

第9章

表列與表格處理

本章討論三種簡單之**資料結構**(data structures)：資料串(string)、表格(table)、與表列(list)，以及用以存取此些資料結構之技巧。Z 8 0 指令集中之搜尋指令組，即特別設計以搜尋資料串與表格資料，在本章有詳細之討論。

9—1 資料串

資料串(data string)即一系列之資料，通常的用法是指文數字資料(character data)。因此，你常聽到的名詞是**文字串**(character string)。Z 8 0 組譯程式有一虛指令 TXT，使程式設計者能極便利地產生一系列程式運算所需之 ASCII 文數字資料。譬如，

```
MESG1      TXT      $THIS IS A  CHARACTER
                                STRING$
```

即令組譯程式，將文數字串“ THIS IS A CHARACTER STRING ”之 ASCII 碼，連續儲存於位址 MESG1 起之記憶位置，每一文數字佔一記憶位置。

資料串之運作對**文句處理** (text processing) 以及**編譯程式之作業** (compiler operation) 十分重要。目前已有某些高階語言專門設計以作資料串之處理。

Z 80 能自一系列資料位元組中搜取某一已知位元組。由於此種作業只需找出一與已知搜取值 (search key) 相匹配或吻合 (match) 之資料位元組，故不論文數字或其它資料皆可適用。於 Z 80，搜尋指令之製作與應用 (第七章說過，比較指令亦可用以搜取某一位元組，但手續較繁。)，均類似於其它與區段有關之指令。指令使用前，**八位元搜取值存入累加器 A。HL 暫存器對存資料串之起始位址，且 BC 暫存器對存資料串之長度** (欲搜尋之位元組數)。

若使用 C P I 指令，則搜尋屬“半自動”。換言之，只有一個指令被執行 (一次)。H L 暫存器對所指及之位元組被拿取，並與累加器 A 之內含相比較。然後，H L 之內含加一，B C 之內含減一。若遞減一後 B C 之值不為零，則 P / V 旗號自動置定為 1。根據比較結果 (實際為相減) 是負、相等、或產生半進位，S、Z、與 H 旗號分別被置定為 1。然後，微處理器繼續執行次一緊接指令 (即緊接 C P I 後之指令) C P I 之應用將留待下一節再舉例說明。

若使用 C P I R 指令，則所有的佈建 (setup) 與執行細節均同。唯一差異的是，比較與加一 (及計數值減一) 運算將一直被重複執行至已找到欲搜取元素，或資料串之所有位元組均已搜遍 (位元組計數器 B C 之值為零) 為止。運算過程因何種狀況停止，可由 P / V 與 Z 兩旗號測知。

下面，我們舉一例子說明 C P I R 指令之應用。例 9 ~ 1 之程式，自一長 64 個位元組之文數字串，搜取 “\$” 之文數字。資料串之起始位址為 S T R I N G。

例 9 ~ 1 以 C P I R 作區段搜尋

；該副程式以 C P I R 指令，自一長 64 個位元組之文數字串中，搜

；取文數字 “\$”。若搜取成功 (即找到欲搜取元素)，則程式將文數字所在之記憶位置的位址，存至位址 P N T R 與 P N T R + 1 兩；記憶位置 (低次八位元在前)。若搜取不成，則回返時累加器存 0；0 H。資料串之起始位址為 S T R I N G。

```

;
SRCHD  LD      A, 24H          ; $ 之 A S C I I 碼。
        LD      HL, STRING     ; 取入起始位址。
        LD      BC, 64         ; 取入位元組計數值。
        CPIR                    ; 展開搜尋。
        JP      Z, FOUND       ; 若找到，則跳至 FOUND。
        LD      A, 0           ; 搜取不成，A ← 0。
        RET                    ; 回返。
FOUND  DEC     HL              ; 成功，指示器值減一。
        LD      (PNTR), HL     ; 位址存入記憶器。
        RET                    ; 回返。

```

程式一開始，錢號 “\$” 之十六進 A S C I I 碼先取入累加器 A，H L 暫存器對指至文數字串之起點 (第一個文數字所在之記憶位置)，並且，B C 暫存器對存位元組計數值。C P I R 然後對含 64 個文數字之資料串展開搜索。若找到 “\$” 之文數字，則零值旗號 (Z) 被置定為 1，控制脫離 C P I R，然後執行次一緊接指令 J P Z, FOUND。由於此時 Z 旗號值為 1，故跳越成功。微處理器緊接執行標題 FOUND 以下之指令。將 H L 暫存器對內含減一 (因 CPIR 指令先作比較，然後才將 H L 內含值加一)，求出 “\$” 文數字所在之記憶位置的位址，並將之儲存於位址 P N T R 與 P N T R + 1 之記憶位置，然後回返。

若錢號 “\$” 正巧為文數字串之最後一文數字，控制離開 CPIR 指令時，Z 與 P / V 兩旗號將共同顯示此一終止狀況；Z 旗號置定為

1，以顯示搜尋成功，同時，P/V 旗號清除為 0，以顯示位元計數值已遞減至零。由此可見，於測試 P/V 旗號看資料串是否結束之前，必須先測試 Z 旗號看搜尋是否成功。圖 9～1 所示即為典型搜尋成功例子之終止狀況。

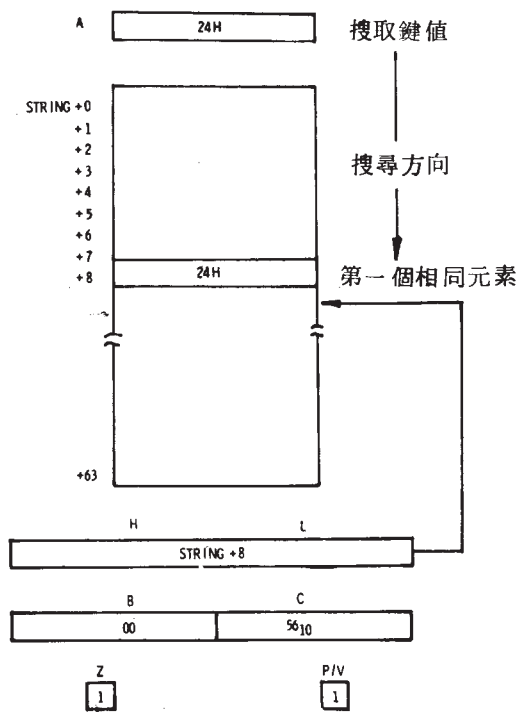


圖 9～1 文數字串搜尋之終止情況舉例

注意，雖然例 9～1 之副程式有兩個出口（exists）；亦即有兩個回返指令。不過，每次副程式被叫用後，控制一定僅由其中之一出口出去（回返）。

正如第四章所說的，除了程式一開始，HL 先指至資料串之末尾（最高點），然後再逐次遞減外，CPDR 區段搜尋指令之動作完全類同於 CPJR（BC 之內含當然還是遞減！）。若採用 CPDR 指

令，則例 9～1 程式之最初四個指令必須改為

```
SRCHD  LD      A, 24H
        LD      HL, STRING+63    ; 指至最後文數字。
        LD      BC, 64
        CPDR                      ; 反向搜尋。
```

搜取值為一個文數字時，CPJR 與 CPDR 非常有效率。若未找到搜取值，則指令重複執行一次所需的時間為 5.25 微秒，致搜尋 64 位元組文數字串平均所需之時間為 $64 \times 5.25 \div 2 = 168$ 微秒。若採用比較指令之方法，先取入、比較、條件測試、然後再調整指示器與迴路計數器之值，則所需時間將兩倍於 CPJR 與 CPDR，而所佔空間則遠超出採用區段搜尋指令之程式。

若搜取值為一個文數字以上，則同樣可採用文字串比較指令，但比較程序即不如單位元組搜尋有效。若搜取值之第一文數字為常出現之文數字，則程式必須浪費大量時間於搜取值之其餘文數字的比較；若第一文數字為甚少見之文字，則所需之比較以及比較失敗之次數就減少，效率即可相對提高。例 9-2 即為自 64 個文數字之資料串，搜則一兩文數字之字串的情形。首先，程式先作搜取值第一文數字之搜尋，若找到了，則資料串之次一文數字與搜取值之第二文數字相比。若相同，則程式結束。若不同，程式自找到第一文數字處重新開始。

例 9-2 搜取兩文數字之字串

；該副程式自一含 64 個文數字之資料串，搜取一兩文數字之字串。
 ；資料串儲存於位址 STRING 起之記憶位置，欲搜取之字串則儲存
 ；於位址 CHAR 與 CHAR+1 兩記憶位置。若搜尋不成，回返時累
 ；加器存 0；若搜尋成功，回返時累加器存 FFH。
 ；

```

BIGSRC  LD    HL, STRING    ; 資料串起始位址。
        LD    BC, 63        ; 最多 63 個位元組。
LOOP    LD    A, (CHAR+0)    ; 取入搜取值之第一文數
        CPIR                ; 字，並開始搜尋。
        JP    NZ, NFND      ; 若失敗，則 NFND。
SECLVL  LD    D, (HL)        ; 資料串次一文數字。
        LD    A, (CHAR+1)    ; 搜取值第二文數字。
        CP    A, D          ; 兩者相比。
        JP    NZ, LOOP      ; 若不同，重新開始。
FOUND   LD    A, FFH        ; 搜尋成功。A ← FF。
        RET                ; 回返。
NFND    LD    A, 00H        ; 搜尋不成，A ← 0。
        RET                ; 回返。

```

若第一文數字搜尋成功，則程式進入標題 SECLVL 處，繼續作第二文數字（第二層次）之搜尋。由於此時 HL 正指至資料串中，與搜取值之第一文數字相配之文數字的次一位置，故次一緊接文數字可直接取入 D，並與搜取值之第二文數字（取入於 A）相比。若兩者相同，則零值旗號為 1，JP NZ, LOOP 跳越不成。微處理器繼續 FOUND 處之指令，將 A 之內含置定為 FF，然後回返。否則，若第二文數字與資料串之次一文數字不相同，則控制跳回 LOOP 處，再繼續往下重作第一文數字之搜尋。注意，當控制於此種情況下再重新進入 CPIR 之比較迴路時，由於指示器與迴路計數器值未曾被破壞，故比較可從原來中止處，繼續往下進行。

若搜尋一直失敗，則於搜遍整個資料串後，控制帶著零值旗號為零之狀態離開 CPIR 指令，繼續執行次一指令。由於此時零值旗號為零，故 JP NZ, NFND 跳越成功，控制轉移至標題 NFND 處，將累加器內含清除為零，然後回返。

例 9 - 2 之程式可再擴展成搜取任一已知長度之文數字串的程式，或將第二層次之比較改成計值（hashing）比較。於計值比較，搜取值之其餘文數字（除第一個外）並不與資料串之文數字一一作比較。此種方法將搜取值之其餘文數字（的 ASCII 值）加在一塊兒，得到一特徵值（hash value）。同時，亦將資料串中緊接之幾個位元組相加，得到另一特徵值。兩特徵值再相比，若值相等，則程式再作真正之比較，否則，程式繼續新一次的第一文數字搜尋。表 9 - 1 舉例說明了特徵值的計算情形。特別注意，當兩特徵值相等時，緊接之一對一真正比較乃是必需的，因為，如表 9 - 1 所示，（BROWN 與 MAUVE）兩不同之文數字串可能產生相同之特徵值。

例 9 - 3 所示即為以計值法，自長 64 個位元組之資料串，搜取一五個文數字之字串的情形。

表 9 - 1 特徵值計算舉例

文數字串	ASCII	特 徵 值
WHITE	57H, 48, 49, 54, 45	181H
GREEN	47 52 45 45 4E	171H
BROWN	42 52 4F 57 4E	17EH
MAUVE	4D 41 55 56 45	17EH

例 9 - 3 計值法搜尋

此副程式以計算特徵值法，自一 64 位元組之資料串中，搜取一已知長五個文數字之文數字串。已知搜取鍵存於位址 CHAR 起之連續記憶位置，而待搜尋資料串存於位址 STRING 起之連續記憶位置。控制回返後，IX 指至匹配文數字串之起點。若搜尋不成，則 IX 含 0。

```

HASHSR  LD    IX, CHAR    ; 五字搜取鍵之起始位址。
          LD    A, (IX+1)  ; 計算搜取鍵後四個文數字之特
          ADD   A, (IX+2)  ; 徵值。
          ADD   A, (IX+3)
          ADD   A, (IX+4)
          LD    D, A        ; 特徵值存入 D。
          LD    HL, STRING ; 待搜尋文數字串之起始位址。
          LD    BC, 60     ; 60 個位元組做完。
LOOP     LD    A, (CHAR+0) ; 搜取鍵第一字母取入 A。
          CPIR              ; 先找出第一字母相同者。
          JP    NZ, NFND   ; 若不成，則跳出。
SECLVL  PUSH   HL          ; 指示器內含 ( 指至第二字母 )
          POP    IX        ; 取入 I X。
          LD    A, (IX)    ; 計算爾後四個文數字之特徵
          ADD   A, (IX+1)  ; 值。
          ADD   A, (IX+2)
          ADD   A, (IX+3)
          CP    D          ; 與搜取鍵之特徵值相比。
          JP    NZ, LOOP   ; 若不等，則繼續搜尋第一文數
                           ; 字相同之字串。

```

; 第一文數字相同，且後面四個文數字之特徵值亦相等。故開始作字
; 母之一一比較。

;

```

MAYBE   LD    IY, CHAR+1 ; 指至搜取鍵之第二文數字。
          LD    A, (IX)    ; 取入資料串之第二文數字。
          CP    A, (IY)    ; 與搜取鍵第 2 文數字比。
          JP    NZ, LOOP   ; 不同則重新再找新字串。

```

```

          LD    A, (IX+1)  ; 第 3 文數字是否相同？
          CP    A, (IY+1)
          JP    NZ, LOOP   ; 若不，則再找新字串。
          LD    A, (IX+2)  ; 第 4 文數字是否相同？
          CP    A, (IY+2)
          JP    NZ, LOOP   ; 若不，則再找新字串。
          LD    A, (IX+3)
          CP    A, (IY+3)
          JP    NZ, LOOP
FOUND   DEC    IX          ; 搜尋成功。I X 指至匹配文數
          RET              ; 字串之起點。
NFND    LD    IX, 0        ; 搜尋失敗。I X 存 0。
          RET

```

例 9-3 之程式分三階段進行。第一階段，標題 LOOP 至 SEC L V L 之前，程式先將搜取鍵之第一文數字取入累加器 A，並從資料串中找尋與此一文數字相同者 (C P I R)。若找尋未果 (即整個資料串中無任一文數字與搜取鍵之第一文數字相同)，則表搜取文數字串必不在資料串內，控制跳至標題 N F N D 處，將指示器值設定為 0 後，回返。若第一階段搜尋成功，則程式進行第二階段——標題 S E C L V L 以下，M A Y B E 以前。

於第二階段，程式計算剛剛匹配文數字之後連續四個文數字的特徵值，並將之與已存於 D 之搜取鍵的後四個文數字之特徵值相比。若兩值不等，則表目前之可能文數字串仍與搜取鍵不同，控制再跳回 L O O P 處，重複第一階段之動作，繼續往下尋找第一文數字相同之字串。否則，若兩特徵值亦相等，則程式進行第三階段——標題 M A Y B E (可能是) 以下，將可能資料串之其餘四個文數字，一一與搜取鍵之後四個文數字相比。若有任一文數字不同，則控制同樣跳回標

題 LOOP 處，重複第一階段之動作。否則，若四個文數字均相同，則搜尋成功。IX 指至匹配文數字串之第一文數字後，控制回返。

9-2 表格作業

資料串屬於表格之一種，為資料項目之連續表（排）列。於資料串，每一元素不折不扣為一 ASCII 文數字。表格（table）則由許多任意長之元素（資料項目）組成，並且元素間可有階級（層次）存在——每項元素又可有子元素。表格之元素可為有序的、無序的、或由某一外部鍵所索引的。每一表格元素之子元素亦可排序。

9-2-1 表格索引

現在，我們舉一由人名與地址構成之表格，說明 Z80 之索引表格的應用。假設愛群公司有 101 個員工，並以電腦作薪資處理。每一員工由 1 至 101 編號。電腦之薪資程式使用一由員工號碼存取（即索引）之表格，表格中每一元素所含之資料如圖 9-2 所示，包括姓名、地址、與身份證號碼等。例 9-4 之程式自表格拿取一員工姓名，並將之送至印字機之暫存區，以便印出。

例 9-4 以索引法存取人事資料表格

；該副程式自一薪資程式所建立的索引表格，取出一員工姓名，送至；印字機之資料暫存區，準備印出。控制進入副程式時，累加器必須；含員工之識別號碼，此一號碼緊接被轉換成資料之索引值。員工資料表格之起始位址為 TABLE，印字機資料暫存區之起始位址為；BUFFER。

；

```
GETNME EQU $
        LD HL, TABLE ;員工資料表格起點。
        DEC A ;身份識別碼換成索引 0 ~
```

```
LD B, A ; 100，並存入 B。
LD C, 0 ;現 BC 存（索引×256）
SRL B
RR C ; BC 現存（索引×128）
ADD HL, BC ;指至員工姓名之起點。
LD BC, 15 ;姓名長 15 個文數字。
LD DE, BUFFER ;指至印字機暫存區起點。
LDIR ;整批傳輸。
RET
```

於 9-4 之程式，累加器所含之員工號碼首先被轉換成索引值 0 ~ 100₁₀。然後，索引值存入 BC 暫存器對，並乘以 128，因為，每一員工之資料（也是表格之每一元素）長 128 個位元組。此一位移求得後，與表格之起始位址相加，即為所要員工之資料（也是姓名資料）的起點。然後，LDIR 將長 15 個位元組之員工姓名，整批搬至印字機之緩衝區。

於上述例子，若表格之資料既無次序且無識別號碼索引，表格資料之存取就必須利用一一比較識別碼之方法。若識別碼值為 1 至 255，且儲存於員工資料之最後位元組，如圖 9-2 所示，則身份識別碼之搜尋可如下反向地進行：

；自人事資料表格搜取身份識別號碼。

```
GETID LD HL, TABLE+128*NENT-1
      LD BC, NENT ;計數值=元素個數。
      LD A, (KEYID) ;取入識別碼鍵。
      LD DE, -127 ;指示器遞減值。
LOOP CPD ;展開搜尋。
     JR Z, FOUND ;若成功，則 FOUND。
     JP PO, NFND ;若已搜過，則表失敗。
```

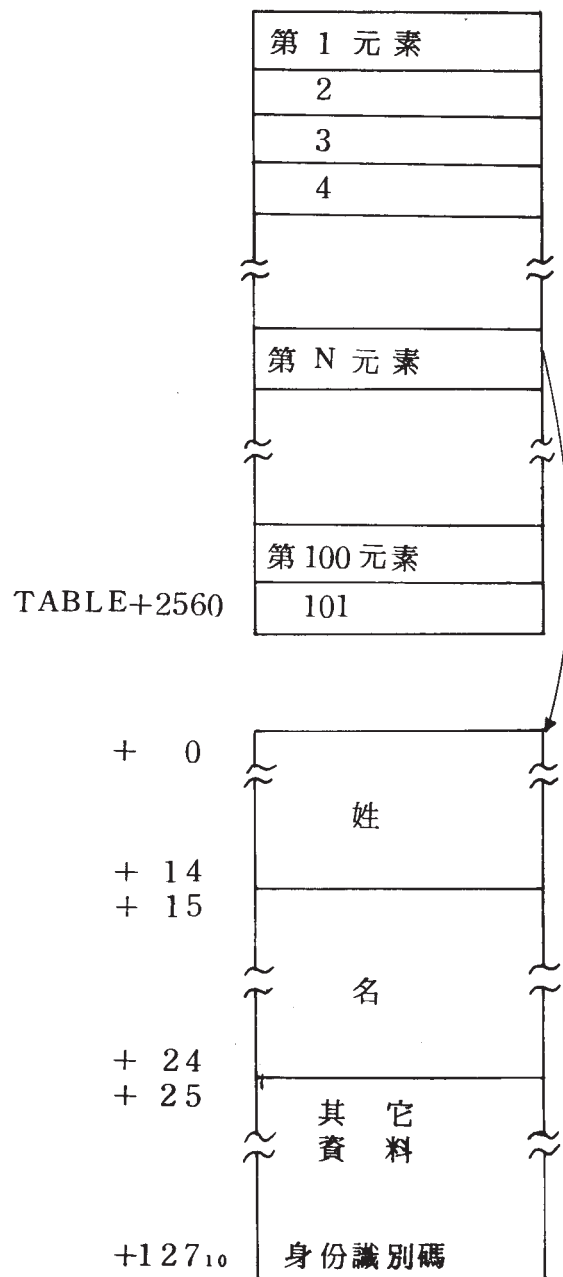


圖 9 - 2 員工資料表格之格式

```

ADD    HL, DE           ; 指至下一識別碼。
JP     LOOP
FOUND  :                 ; 搜尋成功之處置。
      :
      :
NFND   :                 ; 搜尋失敗之處置。
      :
NENT   EQU 101           ; 表格元素個數為 101

```

程式一開始，HL 暫存器先指至表格之最後位元組——最後一個人之資料的識別碼。由於表格之元素個數為 NENT，每一元素（每一人之資料）之長度為 128 個位元組，故最後一位元組之位址為 $TABLE + (NENT \times 128) - 1$ 。CPD 每次將 HL 所指之識別號碼與累加器所含之已知識別碼（搜取值）相比。若搜尋不成，迴路繼續，HL 內含值再減 127（CPD 指令已將 HL 內含減一），指至次一識別碼。若已知識別碼不在表格資料內，最後 BC 之內含將遞減至零，控制轉移至 NFND 處。

剛剛的例子說明了如何自一資料表格搜取一單位元組之資料。n 位元組資料之搜尋可採取同樣之方法，或可採用前一節所介紹之字串比較技巧。

下面，我們分別各舉一程式例題，說明資料表格幾種最常見之作業。在這些常式中，TABLE 均表表格之起始位址，NENT 均為表格之元素個數，LENT 代表每一表格之長度（以位元組計），而 LASTW 皆代表表格之最後字組的位址加一。搜取鍵假設均為 KEY。表格每一元素之第一位元組均含識別鍵。

9-2-2 剔除一已知元素

第一個程式為剔除（delete）副程式。該副程式自表格中除去某一已知元素。程式先以 CPI 指令作搜尋，自表格中找出該剔除之

元素。找到後，LDIR指令將該元素以下之所有元素，一同往前移動一位置，完成剔除作業。留意，此時必須使用LDIR指令，因為，LDDR之作業會蓋掉欲搬動之資料。

例 9 - 5 剔除副程式

；該副程式自表格中除去某一已知元素。若已知元素不在表格內，則；位址PNTR之記憶位置存FFH（該位置在其它情況下存0）。；表格之起始位址為TABLE，表格所含之元素個數為NENT，每；一元素之長度為LENT，每一元素之第一位元組均為識別鍵。搜；取鍵等於KEY。

；

DELETE	XOR	A	
	LD	(PNTR), A	；指示器先令為零。
	LD	HL, TABLE	；指至表格起點。
	LD	BC, NENT	；取入表格之元素個數。
	LD	A, KEY	；取入搜取鍵。
	LD	DE, LENT	；取入每一元素之長度
	DEC	DE	； 並減一。
LOOP	CPI		；搜尋。
	JP	Z, FOUND	；若找到就跳出迴路。
	JP	PO, NFND	；元素不在表列。
	ADD	HL, DE	；指至表格次一元素。
	JP	LOOP	；繼續搜尋。
FOUND	DEC	HL	；指至欲剔除之位元組。
	PUSH	HL	；算出該移動之位元組數。
	LD	HL, LASTW	；取入被減數LASTW。
	OR	A	；清除進位。
	POP	BC	；取回先前存起之HL值。

	SBC	HL, BC	；LASTW-HL。
	OR	A	；清除進位。
	SBC	HL, DE	；LASTW-HL-LENT。
	DEC	HL	；該移動之位元組數。
	PUSH	BC	；最初HL值存起。
	PUSH	HL	
	POP	BC	；BC存該移動之位元組數。
DESTA	POP	DE	；最初HL值（目的位址）。
	LD	HL, LENT	；算出來源位址。
	ADD	HL, DE	；
	LDIR		；資料整批移動。
	RET		
NFND	LD	A, FFH	；元素不在表格內。
	LD	(PNTR), A	；指示器設為FFH。
	RET		
PNTR	DEFS	1	；預留指示器之空間。
	END		

於例 9 - 5 之副程式，程式先作搜尋以找出欲自表格剔除之元素，然後，絕大部份的工作就是設定搬運所需之參數。此時，指至終點位址加一之HL暫存器，其內含先被減一，然後推入堆疊器存起。最後（於DESTA），再取入DE。必須搬動之位元組數乃為LASTW-HL-LENT，該值經計算求得後，存入BC。欲搬動資料之來源位址由HL+LENT提供；該值於算好後存入HL，以備LDIR使用。圖9 - 3所示即為剔除作業中、元素搬動之情形。

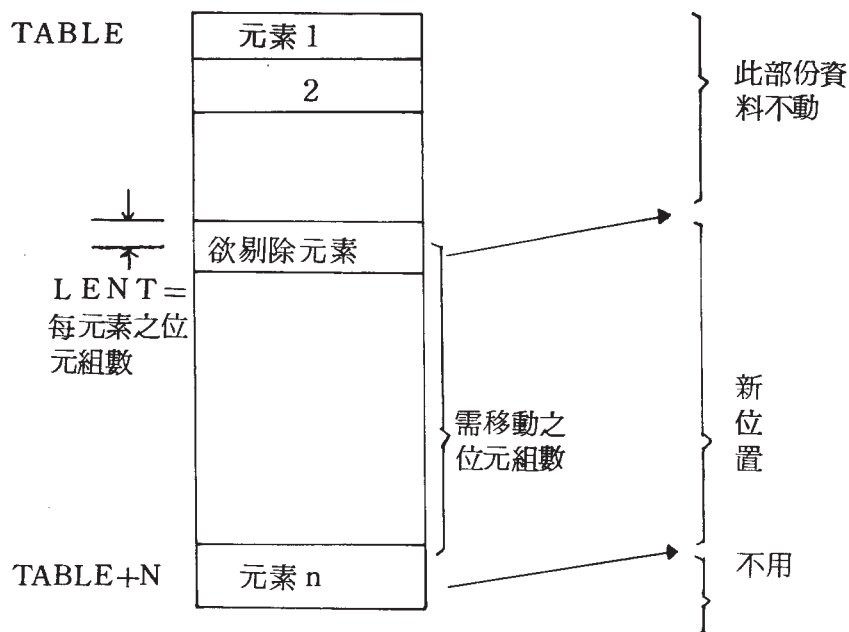


圖 9 - 3 剔除表格元素作業

9-2-3 加入一新元素

第二個常式是在表格加入一新元素。此時，搜尋就不再是找出一等於搜取值之元素，而是必須找出一小於而另一大於搜取值之兩相鄰元素（或小於與等於，或等於與大於兩種情形亦可）。由於在每一次搜尋時，所有搜尋指令均置定符號旗號，以致比較能輕易進行。由於搜尋唯有等於時才終止，故此種搜尋當然必須採用 C P I 或 C P D 指令。

例 9 - 6 加入（新元素）副程式

；該副程式在位址 TABLE 起之表格內，找出一新元素所應插入之位置。新元素之識別鍵為 KEY。表格所含之元素個數 NENT

；，每一元素之長度為 LENT（以位元組數計）。於找到適當位置；後，HL 暫存器對存放新元素插入點之位址，然後控制回返。若新元素該加於表格最後，則回返時，累加器含 0。HL 指至新元素該加入之第一位置。

```

INSERT    LD      HL, TABLE    ; 指至表格起點。
          LD      BC, NENT      ; 取入表格之元素個數。
          LD      A, KEY       ; 取入搜取值。
          LD      DE, LENT      ; 每一元素之位元組數。
          DEC     DE           ; 元素長度減一。
LOOP      CPI                     ; 展開搜尋。
          JP      M, LTHAN      ; 搜取值值小時則跳出。
          JP      PO, END       ; 若已搜遍，亦跳出。
          ADD     HL, DE        ; 否則，指至次一元素之位址。
          JR      LOOP         ; 識別鍵，並繼續。
LTHAN     DEC     HL           ; 指至來源區起始位址。
          RET
END       LD      A, 0         ; 新元素該併於最後。
          RET

```

程式先對表格展開順向搜尋，以找出識別鍵值大於搜取值之第一個元素。（當然，表格元素必須先按識別鍵值之遞升次序排列。）若找不到如此的元素，則新元素就該置於表格之最後。例 9 - 6 之程式並未包括實際加入新元素之指令。若搜找到了一較大值，則自該元素起之後的所有元素，必須集體往後移動等於一元素長度那麼多記憶位置，以納入新元素。因之，程式將 HL 值減一，指至資料移動之來源區的起始位置。副程式自此結束。資料移動之目的區的起始位址，則等於來源區之起始位址加一元素長度。所有必須移動的位元組數

，等於（尚未被搜尋之元素個數+1）×每元素之位元組數，亦即， $(NENT-BC+1) \times LENT$ 。為了避免資料於移開前先被覆蓋而破壞，區段搬運指令必須使用 LDR。圖 9-4 所示即為在表格內加入一新元素的情形。留意，即使新元素必須加於表格之最前端，此一方法仍能正常動作。

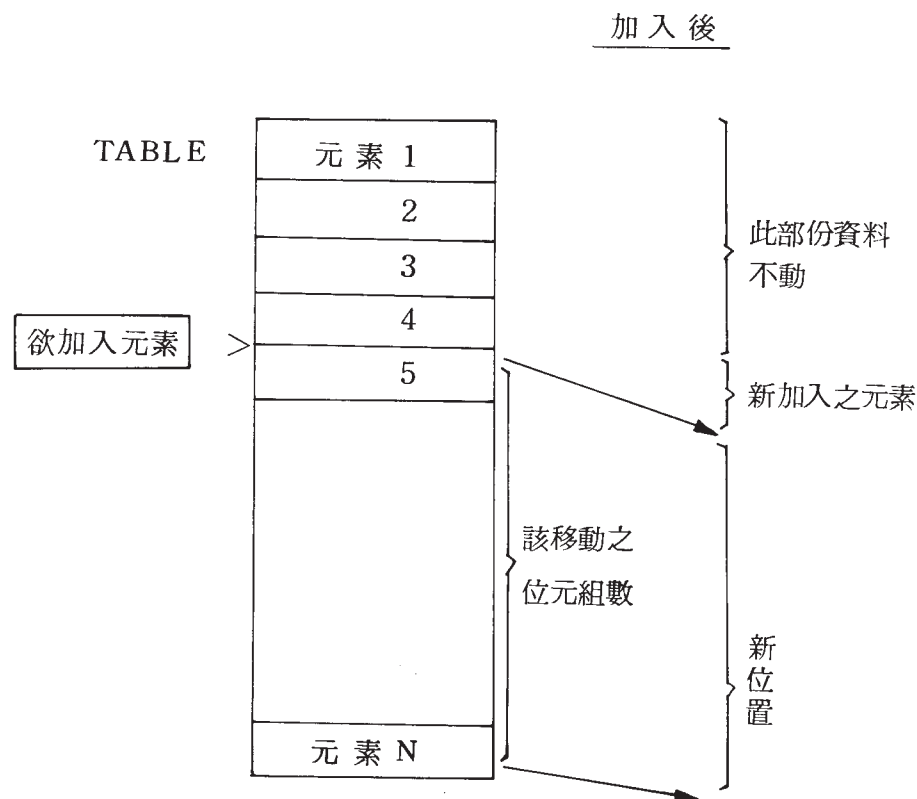


圖 9-4 表格加入一新元素之作業

第三個程式是改變表格之元素的副程式。由於識別鍵值在被更改後，原有之順序已遭破壞。因此，表格之元素必須重新排序（依識別鍵值）。此種作業的方法之一，就是先以剔除副程式除去原來之元素，然後再以加入副程式，將新元素值插入表格內。圖 9-5 所示即為

改變表格元素值之作業情形。

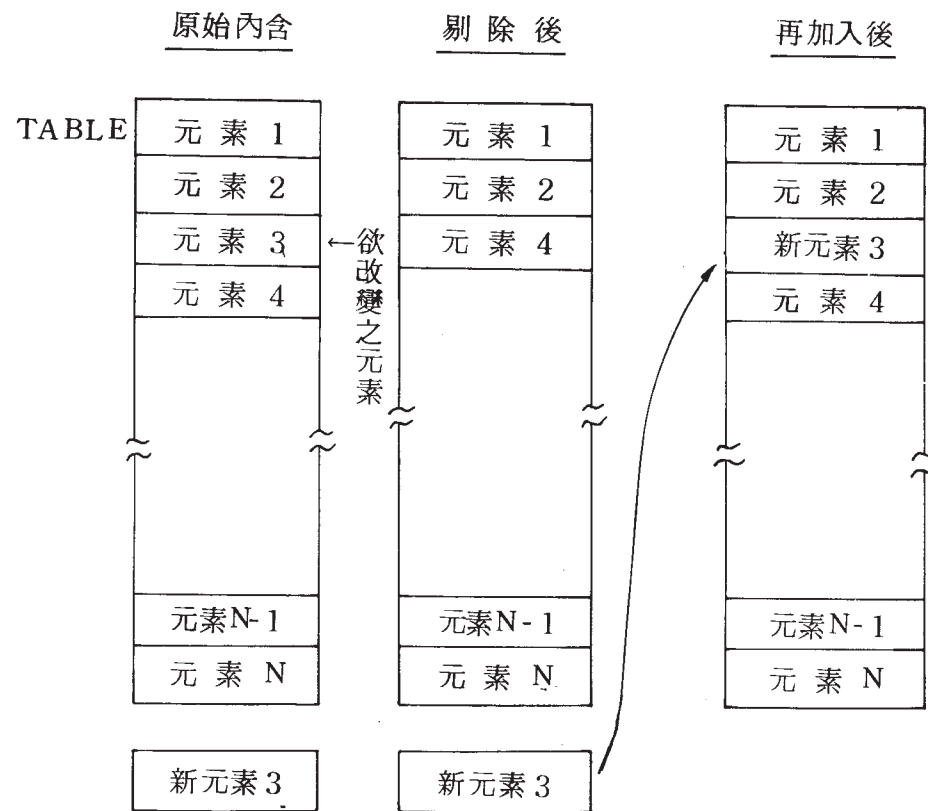


圖 9-5 改變表格元素之作業

9-2-4 二分搜尋

無論向前或向後，Z 80 搜尋指令之作業均屬一種循序搜尋——指令依序讀取每一元素，並加比較。若表格元素已經排成遞增或遞減次序（依某一鍵值），則搜尋尚可用其它方式。對有序表格最常用的一種搜尋方式就是二分搜尋（binary search）。二分搜尋每次將

表格元素區分為二半，將搜取值與中分點之元素的識別鍵值相比，若相等，則找到搜取元素。若不等，則判斷搜取值可能存在於那一半範圍。然後，再以上述之方式搜尋那半段。譬如，於排成遞增次序之表格，若搜取值小於中點元素之識別鍵值，則緊接該搜索的是下半段（識別鍵值較小的一段）；反之，就應搜尋上半段。由於此種方法每次能將必須搜尋（比較）之元素個數減半，因此，效率極高。對於一含有 $NENT$ 個元素之表格而言，程式平均所必須作的比較次數為 $\log_2 NENT + 1$ 次。譬如，若表格所含元素個數為 1000，則二分搜尋法平均所需作之比較次數為 $N = \log_2 1000 + 1 = 9. \dots + 1 = 11$ 次。用循序搜尋法，這平均需要 500 次比較。

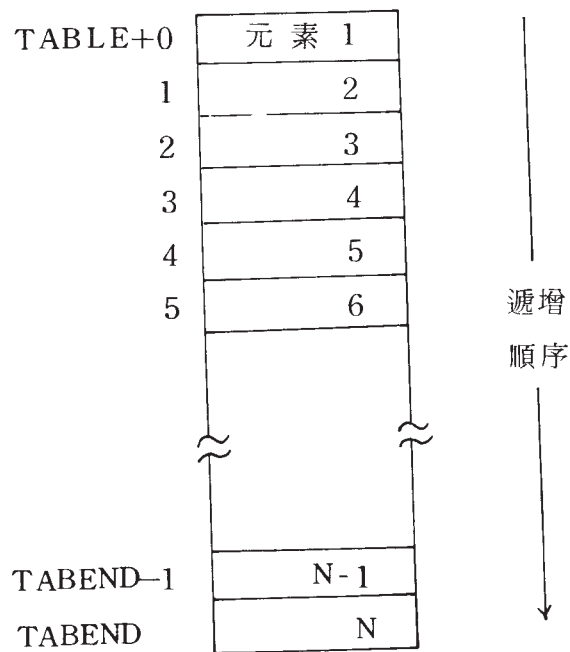


圖 9 - 6 二分搜尋例類之表格

例 9 - 7 之程式自儲存於由位址 TABLE 起至位址 TABEND 之記憶位置的表格，以二分搜尋法搜取一八位元數值。表格最多含 256 個元素，每元素僅長一個位元組，即搜尋鍵本身，如圖 9 - 6 所示。

例 9 - 7 二分搜尋

；此副程式以二分搜尋法，自儲存於位址 TABLE 起至 TABEND ；之記憶位置的表格，搜取一已知八位元值。表格最多含 256 個元 ；素，每元素僅長一個位元組。

```

;
;
BINSRC  LD    B,0                ; 索引低次 = 0。
        LD    C,TABEND-TABLE    ; 索引高 = LAST。
        LD    D,0                ;
        LD    E,C
        LD    IX,KEY             ; IX 指至搜取鍵值。
        JP    JUMP1
LOOP     LD    A,C
        SUB   A,B                ; 高索引 - 低索引。
        SRL   A                  ; (高索引 - 低索引) / 2。
        LD    E,A                ; 中點元素位移。
JUMP1    LD    HL, TABLE         ; 指至表格起點。
        ADD   HL, DE              ; START + (高 - 低) / 2
        LD    A,L
        ADD   A,B                ; 低 + (高 - 低) / 2
        LD    L,A
        JP    NC, JUMP2          ; 若無進位，則略過。
        INC   H                  ; 否則，高位元組加一。
JUMP2    LD    A,(HL)            ; 取得中點元素。

```

```

CP      (IX)           ; 與搜取鍵相比。
JP      Z, FOUND       ; 若相等，則跳出。
JP      M, JUMP 3       ; 元素值小時，亦跳出。
LD      B, E
JP      LOOP
JUMP 3  LD      C, E
JP      LOOP

```

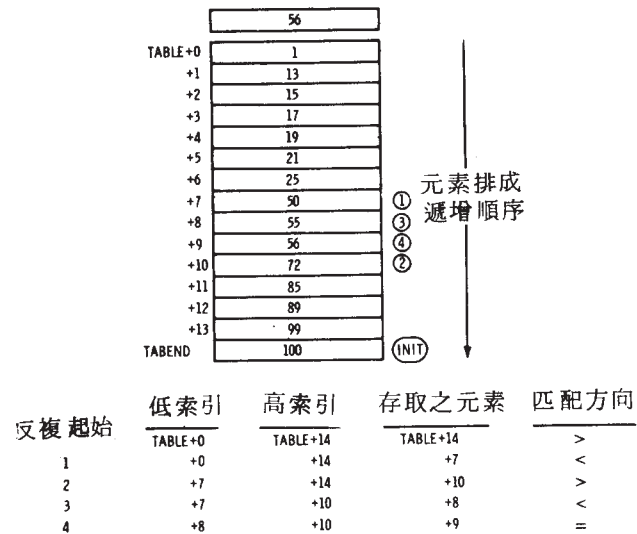


圖 9 - 7 二分搜尋舉例

於例 9 - 7，程式首先將搜取鍵與表格之最後元素相比（以避免於計算次一指標時，產生捨去之錯誤）。若搜尋未成，則控制再進入程式之反覆部份。每反覆一次，新的上限與下限即根據前一比較的結果重新建立。E 中之現有指標，存入 C（高指標）或存入 B（低指標），以作為新上（下）限。欲搜尋區之中點位移然後由（高指標—低

指標）÷ 2 求得，並且加至低指標之位址，求得該搜尋之新位置。搜尋鍵然後與該新位置之值相比，若相等，則搜尋成功；若不成，則算出新的二進搜尋區與新搜尋位置，迴路繼續。於例 9 - 7，搜尋迴路內並未作終止情況之測試。若搜取鍵存在於表格內，則程式無問題，搜尋必成功，並且控制轉移至 FOUND 處之指令。但若搜取鍵不存在於表格內，迴路就變成一無窮迴路，無法結束。為了避免產生無窮迴路，程式可在迴路內加上一將第（i + 1）次之指標值——〔（高一低）/ 2〕與第 i 次之指標值相比的指令。若兩值相等，則表搜尋失敗，二分搜尋就該結束。圖 9 - 7 所示即為當表格元素數為 15 時，例 9 - 7 之程式的動作情形。由圖中可看出，當搜取鍵 = 56 時，二分搜尋法只需作四次比較，即可找到搜取值。若以循序搜尋法，則需十次。

9 - 3 表列作業

表列 (list) 或許是最基本之資料結構了！其包含許多循序安排在記憶體內之資料單元，每一單元稱為一**元素** (element)，每一元素可長一位元組或數個位元組。元素儲存在記憶中之序列可為連續的，每一元素儲存於緊鄰之記憶位置；亦可為連鎖的，每一資料元素含一指至表列之次一元素的指示器。前種情況稱為**循序表列** (sequential list)，後一種情況稱為**連鎖表列** (linked list)。此外，表列之所有元素亦可依某一鍵值 (key value)，順次排列成遞增或遞減次序；或不依任何次序雜亂排列。前一種情況稱為**有序表列** (ordered list)，後一種情況稱為**無序表列** (unordered list)。

循序表列

最簡單且原始之表列即為循序無序表列。此種表列又稱為**區段** (block)。區段通常指一群儲存於記憶體連續緊接記憶位置之有限長

，但內容却無序之資料。此些資料可為一群實驗數據，一串文數字，或其它形式之資料，如磁碟上之一扇形區。區段有時亦用以代表記憶器之某一邏輯區。

圖 9 - 8 所示即為一循序無序表列之例子。此一表列共含 12 個元素，每一元素長一個位元組。元素之內含為數目。表列之起始位址為 SQLIST。顯然，就此等表列而言，若欲搜取某一已知元素，則程式必須從表列之第一項元素開始，將每一項元素與已知搜取值相比，直至元素值與已知搜取鍵值相等為止。此種搜尋法稱為**循序搜尋**（Sequential searching）。循序搜尋之效率極低。為了提高搜尋效率，無序表列通常會經**排序**（sort）成有序表列，以便能使用其它較有效率之搜尋技巧。

本章，除了討論一種較常用之排序方法外，我們還要討論另一種表列——**連鎖表列**，之各種作業。

SQLIST	20
	16
	85
	77
	23
	15
	46
	25
	33
	10
	76
SQLIST+11	44

圖 9 - 8 循序無序表列

9 - 3 - 1 泡浮排序（Bubble Sort）

將表列元素排序之方法有許多，最簡單直接之方法，就是從表列之前端開始，將第一個元素與其它所有元素一一相比，若順序相反，則位置互調。完後，再將第二元素與爾後之所有元素一一相比，同樣遇次序不對就互調。完後，再第三元素，第四元素，……，直至做完最後第二元素為止。此種方法雖然簡單，但却效率不佳。

另一種效率較高，且經常被採用之排序方法，稱為**泡浮排序法**（bubble sort）。

正如氣泡由水底浮升入空中一般，泡浮排序時，表列元素在記憶器中逐次上浮（移）。（雖然表列元素可排序成遞增或遞減兩種順序，但此處我們將先以遞增順序為討論依據。）在泡浮排序時，表列元素由第一元素起循序被讀取，然後與表列之次一元素相比。若讀取元素大於（遞增順序時）表列次一緊接元素，則兩元素對調。然後，次一元素再與次次元素相比，必要時（順序不對時）對調。如此類推。每次兩兩相比，必要時對調。當微處理器抵達表列最後一元素時，表列中最大值之元素也像氣泡般地“**浮至**”表列最後元素之位置。此乃“泡浮”之名的由來。

以上的過程稱為一**巡迴**（pass）。微處理器將表列所有元素依次兩兩比較過一次。根據以上所述，將整個表列排序完畢通常需要一個以上之巡迴。於第一巡迴，最大元素浮至表列最後位置。此後，微處理器必須再進行第二巡迴，以使表列之第二大元素“浮至”表列之最後第二位置。這完了，還必須有第三巡迴，第四巡迴，……等等。既然如此，排序過程究竟何時終止了？答：當表列所有元素均就序時終止。換言之，在均無對調情形發生之巡迴過後，排序作業即可立即終止。讀者很快可發現，所有泡浮排序程式內均會設有一**對調旗號**，以記錄顯示此一終止狀況。

下面的例子舉例了此一過程。試考慮一具有五個元素之表列，其

原始順序為：

05 03 04 01 02

在泡浮排序之第一巡迴過後，表列元素順序變為

03 04 01 02 05

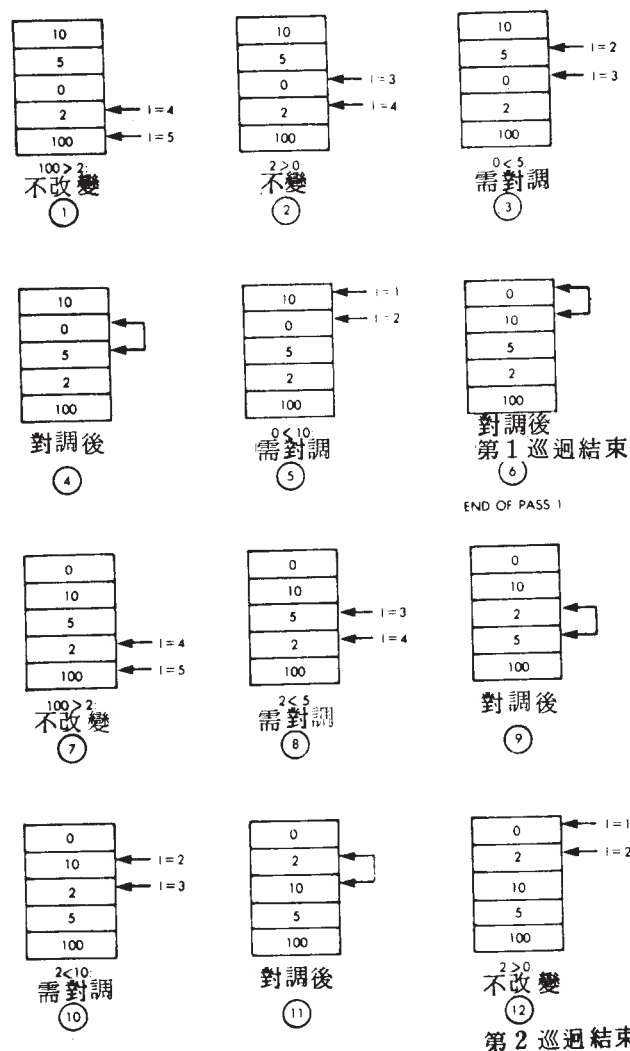


圖 9 - 9

泡浮排序舉例：第 1 至 12 步驟

顯然，最大值元素 (05) 已浮至表列最後位置。除此之外，其它各元素之相對順序不變。第二巡迴過後，表列元素順序又變為

03 01 02 04 05

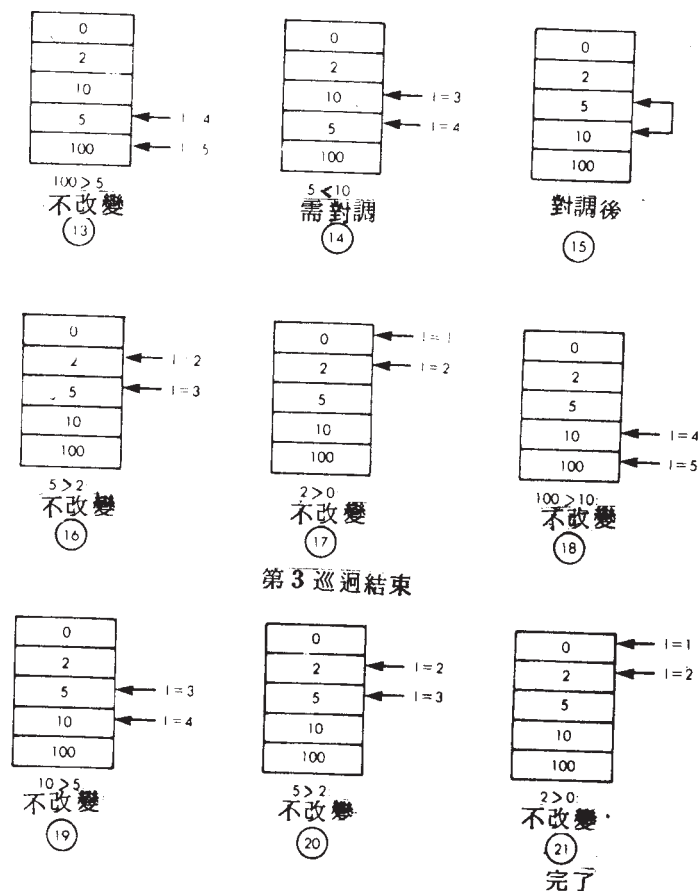


圖 9 - 10

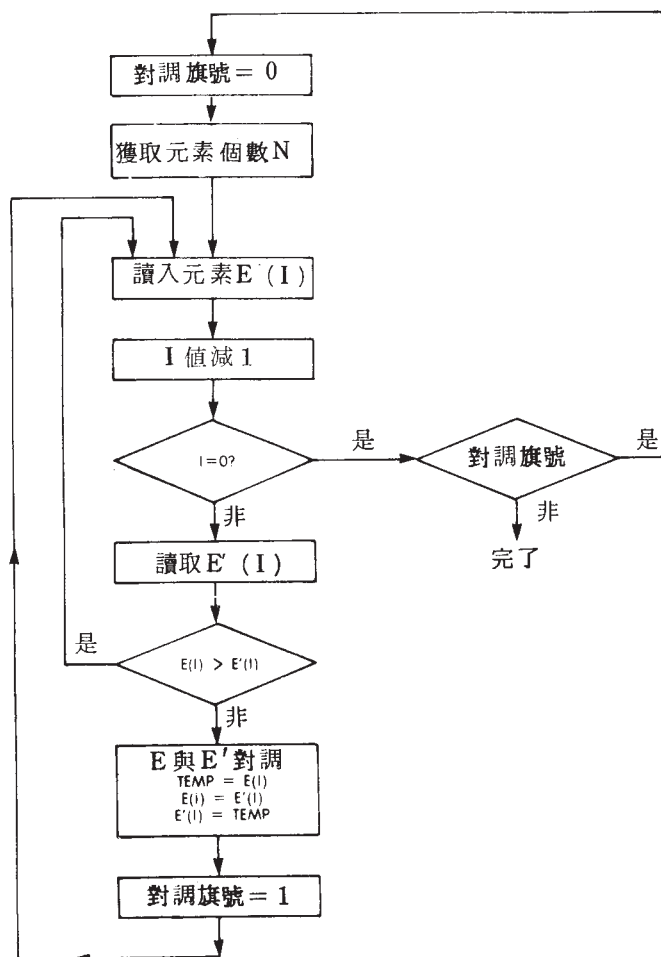
泡浮排序舉例：第 13 至 21 步驟

留意到，此回表列之第二大元素已浮至表列最後第二位置。接着，第三巡迴過後，表列元素順序又變為

01 02 03 04 05

這回，表列之第三大元素又上浮至表列之倒數第三位置。圖 9 - 9 及 9 - 10 所示即為另一五個元素之表列的泡浮排序全部過程。

此一例子不僅說明了泡浮排序如何動作，同時亦顯示了此種方法有何績效。留意，就一含有 n 個元素之表列而言，泡浮排序法最少需 1 個巡迴，最多需 n 個巡迴，平均 $n / 2$ 次巡迴，方能將表列排序完畢。其中， $n - 1$ 個巡迴用以排序，1 個巡迴用以發現所有元素皆已就序之事實。此乃為何即使原始表列皆已全然就序時，此種排序法仍需一巡迴之原故。



泡浮排序流程圖

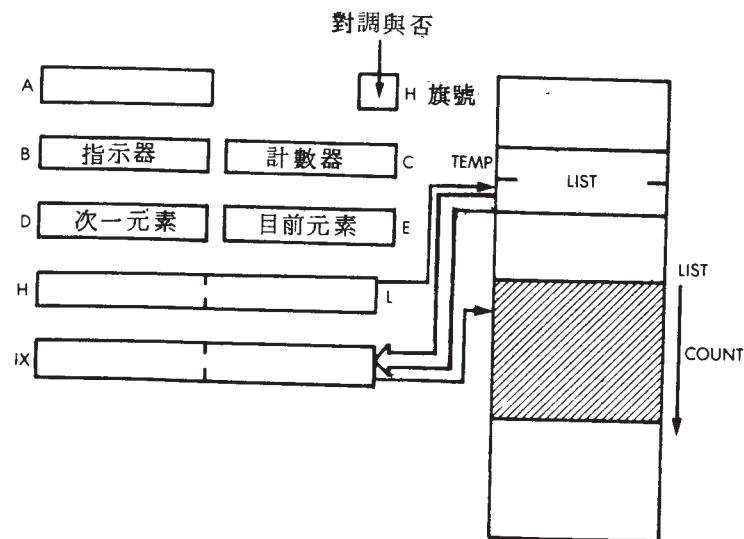


圖 9 - 11 泡浮排序：暫存器及資料佈置圖

例 9 - 8 所示即為一元素為八位元之表列的泡浮排序副程式。該副程式將一元素為八位元之表列的所有元素，以泡浮排序法，排列成遞增順序。叫用該副程式前，H L 暫存器對必須先存表列之起始位址，且 C 暫存器存表列之元素個數值。在整個排序過程中，程式以 I X 暫存器作為表列元素之指示器，H 暫存器之第 0 位元存對調旗號。B 暫存器作為巡迴內之迴路計數器。D 與 E 兩暫存器則存目前欲比較之兩個元素。此一暫存器佈置情形即如圖 9 - 11 所示。而例 9 - 8 之流程圖則如圖 9 - 12 所示。

例 9 - 8 泡浮排序副程式

；此副程式以泡浮排序法，將一資料表列之所有元素，排列成遞增順序。控制進入副程式時，H L 含表列之起始位址，C 暫存器含欲排序之表列的元素個數。排序過程中，副程式使用下列暫存器：

； A 計算用之暫時儲存。
 ； B 表列資料之計數器。
 ； C 欲排序之元素個數。
 ； D 比較之第一元素。
 ； E 比較之第二元素。
 ； H 顯示互換是否發生之旗號（第 0 位元）。
 ； I X 表列之指示器。

```

BBSORT LD      (TEMP),HL      ; 表列指示器換成 I X。
AGAIN  LD      IX,(TEMP)
        RES     FLAG,H        ; 互換旗號先清除為 0。
        LD      B,C           ; 比較次數取入 B。
        DEC     B
NEXT    LD      A,(IX)         ; 取入比較之第一元素。
        LD      D,A           ; 並預留於 D。

```

```

        LD      E,(IX+1)      ; 取入比較之第二元素。
        SUB     E              ; 兩相鄰元素值相比。
        JR      NC,NOEX       ; 若前一元素 > 次一元素，
                                ; 跳至 NOEX。
EXCHGE  LD      (IX),E         ; 否則，兩元素對調。
        LD      (IX+1),D
        SET     FLAG,H        ; 且互調旗號為 1。
NOEX    INC     IX
        DJNZ    NEXT          ; 繼續比次兩相鄰元素。
;
; 一巡迴完了，測試互調旗號，以決定是否再繼續下一巡迴。
;
        BIT     FLAG,H        ; 前一巡迴是否曾互調？
        JR      NZ,AGAIN      ; 若是，則繼續下一巡迴。
        RET                  ; 否則，回返。

FLAG    EQU     0
TEMP    DEFS    2
END

```

9 - 3 - 2 單端連鎖表列

當所存資料必須經常改變，同時，元素經常要剔除、加入、或修改時，改變表格資料所導致之虛功（overhead）就變得相當大。表列即為一種能降低此種虛功之資料結構，其資料項目改變時，不需移動太大批之資料。此節，我們介紹一種資料處理上常用之表列結構——單端連鎖表列（Single-ended linked list）。

單端連鎖表列之元素可儲存於記憶器之非連續位置。每一元素包

括一與表列元素有關之資料，以及一指至表列之次一資料項目的指示器（位址）。由於所有資料項目間均以位址指示器相連，故表列之元素可以任意次序儲存於記憶器內。表列之首項資料由儲存於記憶器內之一變數指著，而最後一項資料則通常為 -1 或其它無效資料，以顯示其為表列之末了。圖 9 - 13 所示即為一典型之單端連鎖表列。表列之每一元素長四個位元組，前兩位元組為資料，後兩位元組即為指至次一表列資料項目的指示器。表列之首由 HEADLS 所指。

單端連鎖表列之作業相當簡易。從連鎖表列中去除某項資料，只要將前一項資料之指示器，換成欲剔除元素之指示器即可。同樣地，將一項新資料加入表列，只要將加入點前一項資料之指示器，換成新

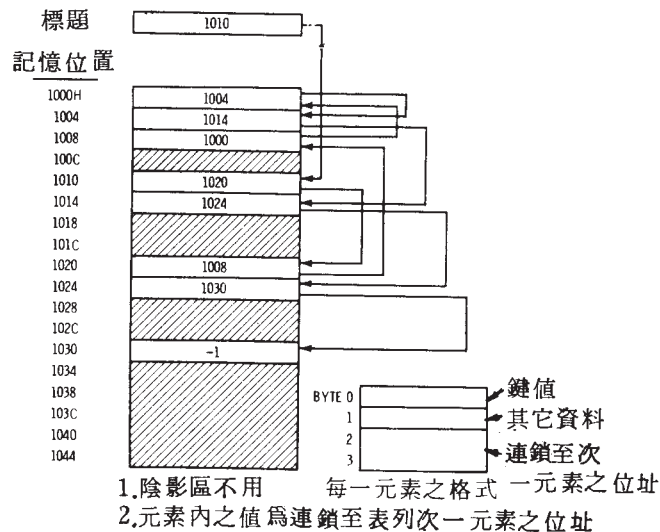


圖 9 - 13 典型之單端連鎖表列

元素之位址，且新資料項目之指示器存插入點後之資料項目的位址即可。圖 9 - 14 所舉例說明的，即為此些動作。

搜尋

例 9 - 9 之程式舉例說明了如何自一單端連鎖表列中搜取一已知值。

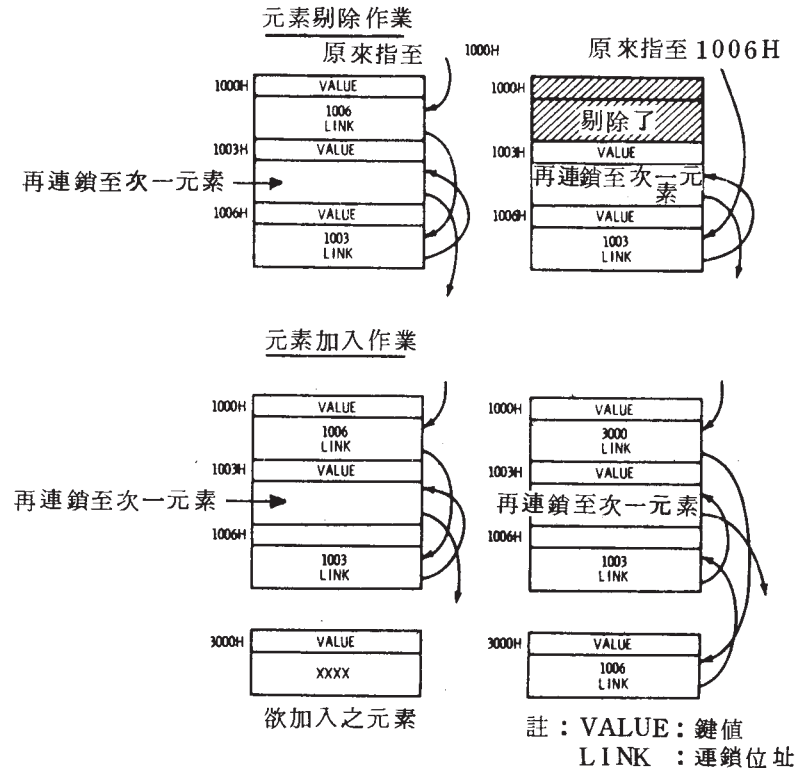


圖 9 - 14 單端連鎖表列之剔除與加入作業

例 9 - 9 單端連鎖表列之搜尋作業

；該副程式自一單端連鎖表列中搜取一已知鍵值 KEY。表列每項元；素含一八位元之資料值以及一兩位元組之連鎖位址。KEY 之實際；值必須以 EQU 虛指令事先定義。表列首項資料之指示器為 HEAD

; D L S , 末項資料之指示器為 - 1 。回返後, 若搜尋成功, 則 H L
; 含匹配元素之位址; 若搜尋失敗, 則 H L 含 0 。

```

SEARCH  LD    HL, HEADLS ; H L 指至表列之首。
        LD    A, KEY      ; 搜取鍵值取入 A 。
        LD    BC, 1       ; 作表列終了比較或清除進位用
                        ; 。
LOOP    ADC    HL, BC      ; H L 指至元素指示器部份。
        JP    Z, END      ; 表列完了, 沒找到。
        DEC   HL          ; 指回資料項目部份。
        LD    D, (HL)     ; 取入次一項資料。
        CP    D           ; 與 A 中搜取鍵值相比。
        JP    Z, FOUND    ; 找到就跳至 FOUND 。
        INC   HL          ; 否則, H L 指至指示器部份。
        PUSH  HL          ; H L 內含取入 I X 。
        POP   IX
        LD    H, (IX+1)   ; 次一元素之指示器。
        LD    L, (IX)     ; 取入 H L 。
        JP    LOOP       ; 再繼續搜尋。
FOUND   RET
END      RET
KEY      EQU          ; 搜取鍵值定義在此。
        END

```

於上述程式, 表列第一項元素之位址即為 HEADLS 。程式一開始, 表列起始位址與搜取鍵值分別取入 H L 與 A 。然後, 控制進入標題 L O O P 以下之迴路。首先檢驗指示器是否為 - 1 (加 1 後變為 0), 若是, 則控制跳至 E N D 處, 表列終了, 搜尋未果。若非, 則控

制往下傳遞。令 H L 指至元素之資料部份, 將資料取入 D 暫存器, 與 A 中之搜取鍵值相比。若兩者相同, 則控制回返, 此時 H L 含匹配元素之位址。若兩者不同, 則 I N C H L 以下之指令, 將表列次一項元素之位址 (即剛剛比較過之資料的次兩位元組) 取入 H L , 然後再繼續。

於表列末了測試時, 例 9 - 9 之程式採用了 A D C H L , S S 指令。特別注意, 除了此一指令之運算結果為零時會影響 C P U 之零值旗號外, 其它所有的 Z 8 0 十六位元算術指令, 均不影響任何旗號。

加入

表列之加入與剔除作業十分類似於在表格作業所討論之循序搜尋。雖然找出加入或剔除點必須費更大的勁, 但一旦此點找到, 改變指示器內含與資料搬動所需之指令就相當有限。例 9 - 1 0 舉例說明了如何將一新資料項目, 加入表列中 I X 所指資料項目之後。

例 9 - 1 0 於單端連鎖表列加入一新元素

; 下面指令系列將一新元素加入單端連鎖表列中, I X 所指元素之後
; 。表列每一元素含一八位元資料值以及一雙位元組之連鎖位址。新
; 元素之位址則存於位址 NEWLKA (低) 與 NEWLKA + 1 (高)
; 兩記憶位置。

```

;
INSERT LD    B, (IX+2)    ; 取得前一元素之連鎖位址
        LD    C, (IX+1)    ; 部份。
        LD    A, (NEWLKA)  ; 新元素之位址存入插入點
        LD    (IX+1), A    ; 前一元素之連鎖位址部
        LD    A, (NEWLKA+1) ; 份。
        LD    (IX+2), A
        LD    IY, NEWLKA   ; 插入點次一元素之指示器

```



```
LD    ( IY+2 ), B      ; 取入新元素之連鎖位址部
LD    ( IY+1 ), C      ;   份。
RET
```

一開始，I X 指至插入點之前一元素。前兩指令將該元素之指示器部份（此部份即為表列次一元素之位址）取入 B C 暫存器。此一連鎖位址應指至緊接插入點後之元素，故最後三個指令將此一位址填入新元素之指示器部份。新資料項目之位址存於位址 NEWLK A（低）與 NEWLKA（高）兩連續記憶位置。第三至六指令將之存至插入點前一緊接元素之指示器部份。

以上我們以 Z 80 之指令集，簡短地討論了表列常見之作業。諸如雙端表列之更高等表列亦經常使用，其有關技巧與上述所言者大同小異。

第 10 章

程式設計技巧

本章討論一些較為高等之程式設計技巧，其中包括跳越表格（Jump table），副程式之種種，與再進入（reentry）等問題。讀者會發覺，部份討論與 Z 80 之控制轉移指令有相當程度之關係。

第四章曾言過，Z 80 微處理器之控制轉移指令包含跳越、副程式叫用、回返、與重始等四種。這些指令能以相對或擴展之定址方式，條件或無條件地跳越至某一記憶位置。若控制轉移出去後不須再回返原先離開之處，則跳越指令 JP（擴展定址）與 JR（相對定址）可用。若控制轉移出去後，緊接指令之位址必須存入堆疊器，以期隨後控制能再回返，則副程式叫用指令 CALL 是唯一入選。CALL 指令用以將控制轉移至儲存於記憶器內另一區段之一系列指令，此一指令系列即稱為副程式。副程式乃僅寫一份，但能在程式內許多不同地方加以重複使用之一段程式指令，使用副程式必須透過 CALL 指令，該指令將緊跟其後之指令的位址推入堆疊器中存起，並將副程式之起始位址（入口位址）取入程式計數器，使微處理器能緊接執行副程式之指令。每一副程式最後必以 RET 指令結尾，該指令將 CALL 指令推入堆疊器中存起之回返位址，自堆疊器中提取，並存回程式計數器，使微處理器能繼續恢復執行原來程式。重始指令 RST 則為一特殊之副程式叫用指令，該指令可用以叫用儲存於記憶器第零頁記憶區（最

初 256 個記憶位置)之副程式，或用於插斷作業。

10-1 表格跳越

跳越指令主要用以形成迴路，或使控制略過某幾個指令。經過第四章之介紹以及前幾章之使用，諒讀者對之已無疑問。此章，於再度簡單摘要此些指令後，主要我們將討論跳越指令之一較高深應用技巧：表格跳越，以及 DJNZ 指令之特性。

Z 80 之跳越指令

跳越指令分條件跳越與無條件跳越兩種。每一種跳越指令於 Z 80 又可使用擴展與相對兩種定址法。表 10-1 與表 10-2 所摘列，即分別為 Z 80 之無條件跳越與條件跳越之指令的情形。於表中，LOC 均代表跳越動作之目的位址。

表 10-1 無條件跳越

擴展定址	相對定址
JP LOC	JR LOC

表 10-2 條件跳越

旗號	擴展定址	相對定址
NZ 非零	JP NZ, LOC	JR NZ, LOC
Z 零	JP Z, LOC	JR Z, LOC
NC 無進位	JP NC, LOC	JR NC, LOC
C 有進位	JP C, LOC	JR C, LOC
PO 奇極性	JP PO, LOC	無
PE 偶極性	JP PE, LOC	無
P 正數	JP P, LOC	無
M 負數	JP M, LOC	無

無條件跳越指令恆使控制轉移至新的目的位址位置。而條件跳越指令則唯有當前述運算結果滿足跳越指令所列之條件時，才發生跳越。條件不滿足時，微處理器繼續執行次一緊接指令。

如表 10-2 所示，Z 80 之條件跳越指令以零值、進位、與極性(溢位)等旗號之狀態，作跳越與否之判斷依據。代表此些條件之組合語言符號隨組譯程式之不同而異。NZ, Z, NC, C, P, 及 M 之意義已不言自明。PO 與 PE 在有些機器則分別以 V (溢位) 及 NV (無溢位) 表示。

擴展與相對定址之取捨，首先需視目的位址之所在而定。若目的位址落於相對跳越之範圍內(即向前 126 個位置，往回 129 個位置)，則可採用相對定址。若目的位址超出此一範圍，則必需採用擴展定址，或擴展定址與相對定址併用。相對定址之跳越指令佔兩個位元組，需 3 微秒之執行時間。而擴展定址之跳越指令則佔三個位元組，但僅消耗 2.5 微秒之執行時間。

除了以上所列者外，Z 80 尚具有其它四個甚為好用之跳越指令：JP (HL)，JP (IX)，JP (IY)，與 DJNZ。底下我們就分別介紹這四個指令之用法。

表格跳越

JP (HL)，JP (IX)，及 JP (IY) 三個指令，分別將 HL, IX, 與 IY 暫存器之內含取入程式計數器，然後引起一無條件之跳越。當程式流程必須作多途分叉，使控制依據多種不同之狀況，分別跳至許多不同點(或不同副程式)時，此些指令甚為好用。舉例而言，假設某一大程式有一“型態”字組，顯示著系統正動作於某一型態，或任何其它之條件組合。如表 10-3 所示，在此假設型態字組 MODE 代表九種系統之不同狀態，其中 0100 與 1000 兩種四位元組合為不可能之狀態，則例 10-1 所示之常式，即根據型態

表 10-3 型態字組之寫碼舉例

其它位元	型態	代表狀況舉例
××××	0 0 0 0	停 電
××××	0 0 0 1	火 災
××××	0 0 1 0	失 竊
××××	0 0 1 1	廁所不通
××××	0 1 0 0	不可能
××××	0 1 0 1	水塔沒水
××××	0 1 1 0	電話不通
××××	0 1 1 1	鄰居失火
××××	1 0 0 0	不可能
××××	1 0 0 1	冷氣機故障
××××	1 0 1 0	室內光亮不足
××××	1 0 1 1	起床時刻

7 6 5 4 3 2 1 0 位元

字組之現有值，分別跳去其所應對應之處理常式。

例 10-1 表格跳越

；位址 JUMPTB 起之記憶位置，儲存有一跳越表格。該表格含有十
；三個跳越至不同處理常式之跳越指令。此一常式即根據表 10-3 之
；寫碼情況，檢驗型態字組之內含，令控制轉移至字組內含所對應之
；狀況的處理常式。型態字組儲存於位址 MODE 之記憶位置。
；

```
TESTMD    LD    BC, JUMPTB    ; 取入跳越表格起始位
                                 ; 址。
          LD    A, ( MODE )    ; 取入型態字組。
```

```
AND    A, 0FH    ; 遮掉字組之高次四位元。
LD    L, A       ; 結果取入 HL 作表格位移。
LD    H, 0
PUSH   HL       ; 位移先存一份於 DE。
POP   DE
ADD   HL, HL    ; 位移值先三倍。
ADD   HL, DE
ADD   HL, BC    ; 然後再加表格起始位址。
JP    ( HL )    ; 實施表格跳越，跳至適當之
                 ; 處理常式。
JUMPTB   JP    MODE0    ; 跳至型態 0 之處理常式。
         JP    MODE1
         JP    MODE2
         JP    MODE3
         JP    ERROR    ; 字組 = 4 為不可能之組合。
         JP    MODE5
         JP    MODE6
         JP    MODE7
         JP    ERROR    ; 字組 = 8 為不可能之組合。
         JP    MODE9
         JP    MODE10
         JP    MODE11
```

於例 10-1 之常式，跳越表格之起始（基底）位址先取入 BC 暫存器對，然後型態字組亦自 MODE 之記憶位置取入累加器 A，型態字組緊接經過一系列之處理，以獲得其所對應之跳越至處理常式之跳越指令，於跳越表格中之位移。首先，AND A, 0FH 先將型態字組之高次四位元皆變為零。其次，此一所得之型態碼被乘 3，並加

至跳越表格之起始位址 (ADD HL, BC)，以求得字組所對應之跳越至處理常式指令的位址，結果存於 HL。最後，JP (HL) 實施表格跳越，使控制跳至適當之處理常式，MODE 0，或MODE 1，……等等。注意，位移值之所以乘 3，乃因為跳越表格之每一跳越至處理常式指令，均佔滿三個連續記憶位置之故。若此些跳越指令為相對定址之 JR MODE 0 等指令，則位移值僅需乘 2 就夠了。

茲舉一實例說明之。若型態字組之最初內含為 F 2 H = 11110010₂ (即表型態 2)。則經 AND A, 0F 指令後，字組內含變為 00000010₂ 存於累加器內。經 LD L, A 及 LD H, 0 後，此一數值存入 HL。HL 內含變成 02H。ADD HL, DE 執行完後，HL 內含被放大三倍，變成 06H。此一數值即為跳越至處理常式指令在跳越表格上的位移。ADD HL, BC 指令執行過後，HL 內含變為 JUMPTB + 6。因此，JP (HL) 指令執行之結果，控制轉移至位址 JUMPTB + 6 之記憶位址上的指令。經拿取後，微處理器解碼與執行該位置起之指令——JP MODE 2。執行結果，控制又轉移至位址 MODE 2 起之指令系列——此即為型態 2 之處理常式。至此大功告成！

若不用例 10 - 1 之表格跳越法，多途分叉亦可採用如下所示之直接跳越法：

```
LD      A, ( MODE )    ; 取入型態字組。
AND     A, 0FH         ; 遮掉高次四位元。
JP      Z, MODE0       ; 若零，則跳至MODE 0。
CP      A, 1           ; 否則，是 1 嗎？
JP      Z, MODE1       ; 若是，則跳至MODE 1。
CP      A, 2           ; 若非，那是 2 嗎？
JP      Z, MODE2       ; 若是，則跳至MODE 2。
:
```

不過，無論就記憶空間與執行時間，或程式之清晰度而言，表格跳越法總是較優越一些。

DJNZ 指令

Z 80 之跳越指令群中最值得一提之指令，該算是長兩位元組之相對定址指令：DJNZ。此一指令主要用於以 B 暫存器作迴路計數器之迴路的終止測試。其將 B 暫存器之內含值減一，若所得 B 之結果為零，則跳越不發生；否則，若 B 之值尚未為零，則跳越發生，控制跳至指令所指明之位置，繼續進行迴路。因之，若迴路之起點為 LOOP 處之指令，則該迴路可以

```
DJNZ    LOOP
```

指令結尾，此一指令之功效即相當於

```
DEC     B
```

```
JR      NZ, LOOP
```

兩指令所達成之效果。

無論何時，只要迴路反覆之次數不超過 256 次，則程式皆可以 B 暫存器為迴路計數器，並採用 DJNZ 指令。例 10-2 所示即為一 DJNZ 應用之典型例子。

例 10-2 DJNZ 應用之舉例

；該副程式使用 DJNZ 指令，計算位址 MEMOP 位置內含之極性，若
；奇極性，則回返時累加器含 1；若偶極性，則回返時累加器含 0。
；

```
PARITY  XOP  A          ; 清除極性及進位旗號。
```

```
LD      B, 8           ; 設定迴路計數器起始值。
```

```
LD      HL, MEMOP      ; 取入記憶運算元位址。
```

```
LOOP    RLC  ( HL )     ; 位元移入進位。
```

```
JR      NC, JUMP 1     ; 若位元為 0，則繼續。
```

XOR	1		; 否則，極性指示器反態。
JUMP 1	DJNZ	LOOP	; 未完繼續。
DONE	RET		; 完了，回返。

若跳越發生，則 DJNZ 需 3.25 微秒之執行時間，若跳越不成，則執行時間僅為 2 微秒。等效之 DEC 與 JR 之作法，於跳越成功與跳越不成兩種情況，則分別需時 4.0 與 2.75 微秒，並且多佔用一位元組之記憶空間。

“副程式”一詞在此之前已屢次提過，本章爾後之篇幅將對此一主題作一較正式化且深入之介紹。

10-2 副程式

10-2-1 何謂副程式

簡言之，**副程式**（subroutine）即為一段僅寫一次，但可無數次叫用（call）之程式。其與一般程式之差別有三：第一，副程式唯有在另一程式將控制傳給它時，才會被執行。將控制轉移至某一副程式之動作，即稱為**副程式叫用**（calling a subroutine），或簡稱**叫用**。第二，每一副程式最後一個被執行之指令，必為**副程式回返**（return from subroutine）指令，或簡稱**回返**（return）指令。實效上，副程式之回返指令將程式控制傳回給叫用自己之程式（該程式稱為**叫用程式**）。第三，副程式必須有一**名字**（name）。此一名字事實上即為副程式被叫用時，第一個被執行之指令的**標題**。此標題所相當之數值位址，常稱為副程式之**入口位址**（entry address）。

10-2-2 副程式之叫用與回返

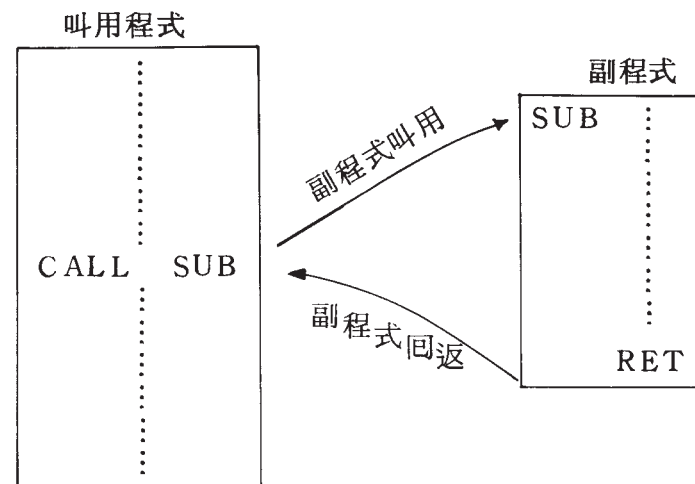


圖 10-1 副程式叫用與回返

於 Z 80，副程式叫用以 **CALL** 指令達成，而副程式回返以 **RET** 指令達成。副程式叫用指令之格式為

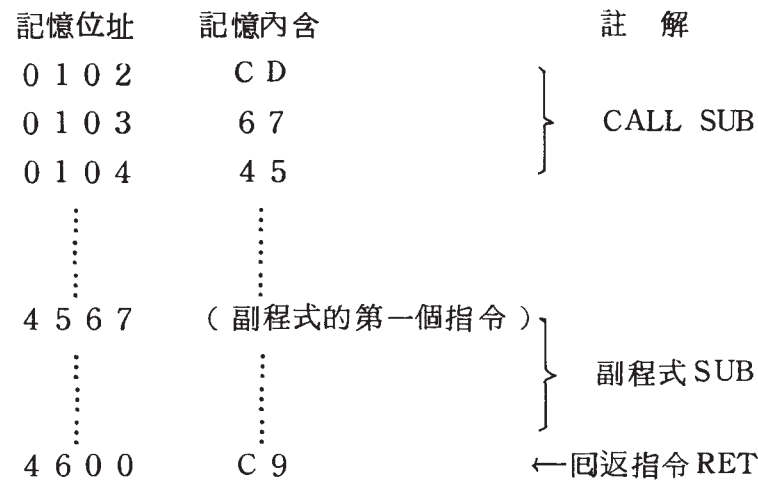
CALL 副程式名稱

。亦即，CALL 後緊接欲叫用之副程式的名稱。此一名稱事實上即為副程式中第一個欲被執行之指令的標題（符號位址）。譬如，若欲叫用一儲存於位址 SUB 起之記憶位置的副程式，我們即可使用指令

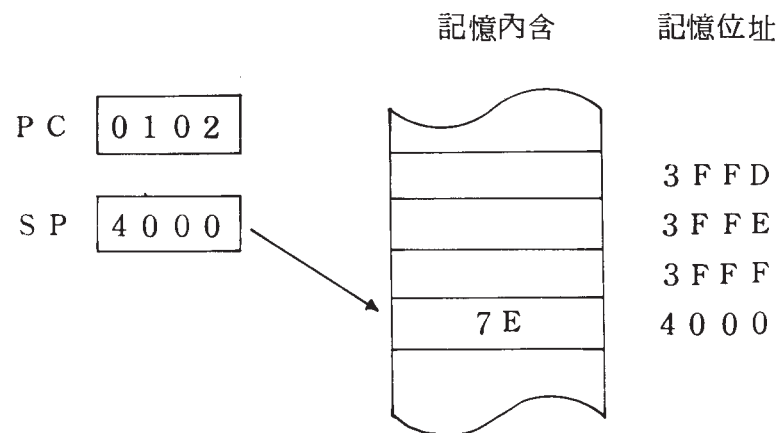
CALL SUB

微處理器執行副程式叫用指令之結果，即將現有程式計數器之內含推入堆疊器存起，然後，將控制轉移至 CALL 指令所標明之位址位置上的指令（此一效應即相當於將緊接 CALL 運算碼後之記憶位址，存入程式計數器）。譬如，若於程式中，我們以 CALL SUB 指令叫用副程式 SUB，而經過組譯與存入後，副程式 SUB 存於位址 4567 H 起之記憶位置（亦即 SUB 之數值位址為 4567 H），副程式叫用指令 CALL SUB 剛好儲存於位址 0102 H 起之記憶位置，則圖 10-2 所示即為 Z 80 微處理器執行副程式叫用指令 CALL

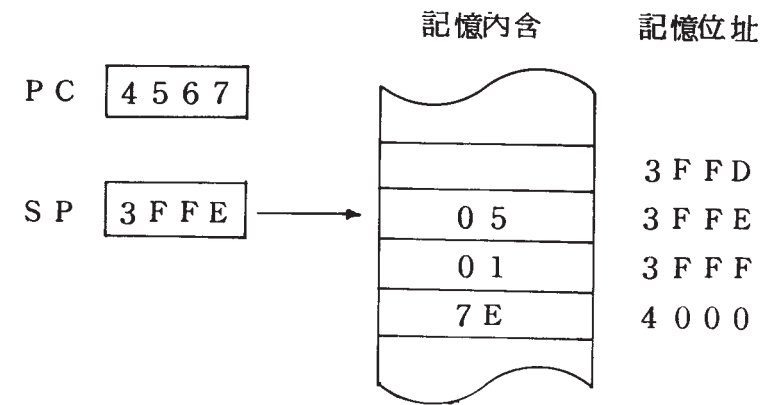
SUB 時，程式計數器 (PC)、堆疊指示器 (SP)、與堆疊器內含之變化情形。



(a) 假想狀況



(b) CALL SUB 指令執行前



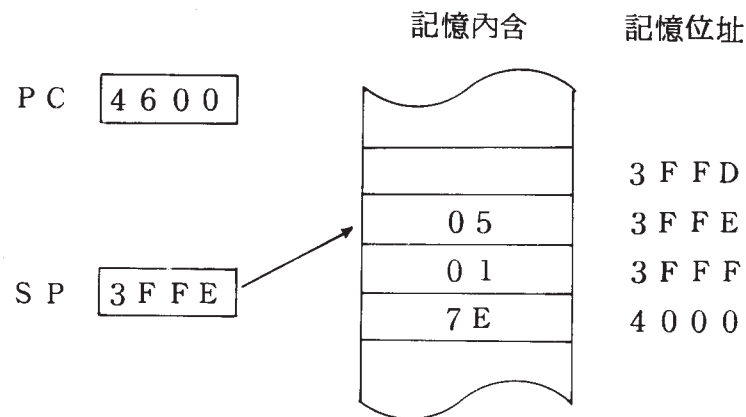
(c) CALL SUB 指令執行後

圖 10-2 副程式叫用指令之效應

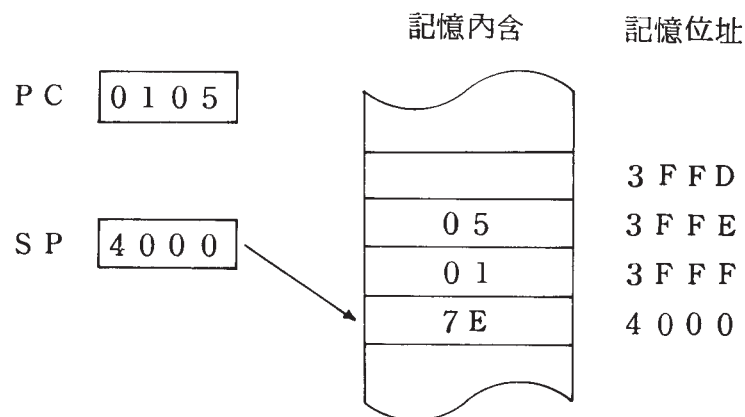
副程式回返指令 (RET) 僅長一個位元組，Z 80 微處理器執行該指令的結果，即將堆疊頂端之兩位元組資料 (即先前存起之程式計數器內含) 取出 (在前例中為 0 1 0 5)，並存入程式計數器，使次一被執行之指令能來自程式計數器新內含所指之記憶位置；亦即為立即緊接叫用指令後之指令。微處理器繼續執行叫用程式未完之部份。若假想狀況同前，則圖 10-3 所示即為 Z 80 微處理器執行副程式 SUB 之回返指令情形。

正如前面第四章所提過的，Z 80 之副程式叫用與回返指令，除了上述兩者外，尚有條件式的叫用與回返指令。此些指令功用類同，唯一的差別是，微處理器於執行此些指令時，會先測試指令所述條件是否成立，若成立，則叫用 (回返)。若不成立，則叫用 (回返) 不發生，微處理器繼續往下執行次一緊接指令。此外，Z 80 尚具有一特殊之副程式叫用指令 RST P，該指令僅能叫用位於八個固定零頁記

憶位置起之副程式中之任一個。



(a) RET 指令執行前



(b) RET 指令執行後

圖 10-3 副程式回返指令之效應

10-2-3 副程式之使用及優缺點

副程式之使用時機與用法

爲了解題，計算機之用者必須設計一能達成目的之程式，爲了與副程式區別，該程式通常稱爲**主程式** (main program)。於設計程式時，經常我們會發覺，某一指令系列於程式中重複出現數次 (亦即，於解題過程中，某一相同作業必須重複做幾次。) 爲了節省寫碼時間與程式所佔之記憶空間，免於每次需要時就需重寫指令系列一次，我們可將此一重複出現之指令系列寫成副程式，然後，於主程式每一需要此一指令系列之點上，代之以一叫用該副程式之叫用指令。

譬如，假設位址 NUM1，NUM2，與 NUM3 之記憶位置分別儲了三個數目。我們的任務是欲將其每一數放大十倍，結果仍存於原處 (假若最後結果均不超過 256)。則我們有兩種作法。第一種作法即如例 10-3 所示般地寫一直線形 (straight-line) 之程式。將三個數目先後一一取出，放大十倍，再存回原處。全部工作以一主程式完成。

例 10-3 直線形程式舉例

；該程式分別將位址 NUM1，NUM2，與 NUM3 之記憶位置的內含
；乘 10，結果置回原處。
；

```
LD    A, ( NUM1 )    ; 取入第一數目。
SLA   A              ; 乘 2。
LD    B, A           ; 2 倍結果存起。
SLA   A              ; 乘 4。
SLA   A              ; 乘 8。
ADD   A, B           ; 十倍正好。
```

```

LD      ( NUM1 ) , A      ; 結果存回原處。
LD      A , ( NUM2 )      ; 取入第二數目。
SLA     A
LD      B , A
SLA     A
SLA     A
ADD     A , B
LD      ( NUM2 ) , A
LD      A , ( NUM3 )      ; 取入第三數目。
SLA     A
LD      B , A
SLA     A
SLA     A
ADD     A , B
LD      ( NUM3 ) , A
HALT
END

```

顯然，此種方法雖然直截了當，但却稍顯笨拙了些。仔細觀察您會發覺，於剛之程式中，處理每一數目之第二至第六指令皆完全相同。換言之，此一指令系列於程式中重複出現三次。因此，若能將之寫成副程式形式，並於每次需要時叫用之，我們即可節省許多寫碼時間以及程式所佔之記憶空間。此一作法即如例 10-4 所示。

例 10-4 副程式之用法舉例

；此一程式藉著叫用十倍副程式 MUT10，將位址 NUM1，NUM2，
；與 NUM3 三個記憶位置之內含乘以十倍。底下之指令系列為主程
；式。

```

LD      A , ( NUM1 )      ; 取入第一數目。
CALL    MUT10             ; 叫用十倍副程式。
LD      ( NUM1 ) , A      ; 十倍結果存回原處。
LD      A , ( NUM2 )      ; 取入第 2 數目。
CALL    MUT10             ; 叫用十倍副程式。
LD      ( NUM2 ) , A      ; 十倍結果存回原處。
LD      A , ( NUM3 )      ; 取入第三數目。
CALL    MUT10             ; 叫用十倍副程式。
LD      ( NUM3 ) , A      ; 十倍結果存回原處。
HALT                      ; 完了。

```

；
；緊接之指令系列為十倍副程式 MUT10。

```

;
MUT10  SLA     A          ; A之內含兩倍。
        LD      B , A      ; 兩倍結果存起。
        SLA     A          ; A之內含四倍。
        SLA     A          ; A之內含八倍。
        ADD     A , B      ; A之內含十倍。
        RET                      ; 回返。
END

```

注意，原來 22 個指令之程式，已為一 10 個指令之主程式以及一 6 個指令之副程式所取代。當副程式之指令數愈多，或副程式被叫用之次數增加時，此兩種方式之指令數的差距就更大，副程式之效果也愈顯著。

值得注意的是，於第二種方法，指令系列重複出現之處均為一 CALL MUT10 指令所取代。另外，順便一提的是，於第二法，

在主程式執行期間，副程式MUT 10 必須同時駐存於計算機之主記憶體內，方不致使CALL MUT 10 指令“叫不到”副程式。此點可以如例 10-4 所示地，將主程式與副程式一同經過組譯與存入程序之方式，加以避免。

副程式之優缺點

副程式雖然具有節省程式寫碼時間與記憶空間，以及增加程式之結構化之優點，但其却增長了程式之執行時間。至少，每次微處理器必須多執行兩個指令：CALL 與 RET。有時甚至還不止於此。為了免於破壞叫用程式既有之運算結果，副程式在開始從事運算之前，必須先將其所用及之暫存器的內含事先存起，並且於控制回返前再將之恢復。此些作業均需增加額外之時間。

暫存器內含之儲存與恢復

例如，例 10-4 之副程式的作業曾經使用 B 暫存器作為暫時之儲存。若主程式中之運算亦曾用及此一暫存器，並且其內含爾後仍然有用，則此時副程式在開始運算之前，就必須先將 B 暫存器之內含推入堆疊器中存起，並於控制回返前將之自堆疊器中取回。例 10-5 所示即為在MUT 10 副程式加上此一作業的情形。

例 10-5 暫存器內含之儲藏與恢復

；該副程式將其所用及之 B 暫存器的內含，事先推入堆疊器中存起，
；並於稍後將之取回。

；

```
MUT 10  PUSH    BC          ; B 暫存器之內含存起。
          SLA     A
          LD      B, A
          SLA     A
```

```
SLA     A
ADD     A, B
POP     BC          ; B 暫存器之內含取回。
RET
```

特別留意到，此一副程式比例 10-4 之MUT 10 副程式增加了兩個指令——最前端之 PUSH BC 與最末端（RET 之前）之 POP BC。若其它暫存器之內含亦欲存起，則方法完全相同。

10-2-4 參數傳遞

副程式被叫用時，其通常需對某些數據運算，此些數據必須由叫用程式供應。例如，當我們叫用一乘法副程式作乘算時，就必須供給副程式兩個欲相乘之數值。叫用程式供給副程式之數據，即稱為**參數**（parameter）或**真值**（argument）。參數之傳遞方法有三：

1. 經由記憶體。
2. 經由暫存器。
3. 經由堆疊器。

以**記憶體**傳遞參數的方法，就是由叫用程式預留參數之記憶位置，並設定參數之實際數值。當然，此些真值之符號位址必須為副程式所使用者。此種方法之優點為彈性大，參數之個數不受限制。但缺點為效用差，且副程式必須有一段專屬之記憶位置。

暫存器亦可用以傳遞真值。以暫存器傳遞真值的方法，就是當控制轉移至副程式之際，副程式運算所需之數據儲存於副程式所知曉之某幾個暫存器內。此種方法之優點為效率高，副程式儲存之位置不受限制，而缺點為暫存器之數量有限，致所能傳遞之真值個數亦有限。此即為何經常有人特別劃定某一塊記憶區，以專供各副程式間真值傳遞之用的道理。

例 10-4 所採用者即為以暫存器傳遞真值之方法。記得，於叫用副程式 MUT 10 時，累加器 A 所含的即為副程式欲將之十倍之數值。此一參數經由累加器 A 傳遞至副程式。故副程式一開始即可加以使用。特別注意，於此一例子，副程式之運算結果亦以累加器 A 傳回給主程式。

以堆疊器傳遞參數之優點同於暫存器之情況：與記憶位置無關。副程式叫用前，叫用程式將欲傳給副程式之參數值一一推入堆疊器。要用時，副程式只需將之一一自堆疊器拉取即可。自然，這也有缺點：堆疊資料混雜了回返位址與數據，也因而減少了堆疊器可資利用之空間。同時，這亦複雜了堆疊器之使用，且或許必須使用多重堆疊。

至於使用那一者，那就看程式設計者之抉擇了！最好儘可能勿使用記憶位置。若暫存器不敷使用，就改用堆疊器。當然，若想傳遞之參數相當多，那也只有直接藉用記憶位置不可了！克服傳遞大批資料困難之方法，即為僅傳遞一參數值之指示器給副程式。該指示器可以 CPU 暫存器，堆疊器，或某一（二）已知記憶位置傳遞。於叫用副程式時，該指示器即等於真值資料區塊之起始位址。

最後，若以上之方法皆不可行，則叫用程式可與副程式達成默契，以某此固定記憶位置（“信箱”）作真值傳遞。

10-2-5 副程式巢串

叫用程式叫用副程式之次數不受限制。同一叫用程式可重複叫用同一副程式。圖 10-4 所示即為一叫用程式兩度叫用副程式 SUB 的情形。此外，副程式之叫用程式可為主程式，亦可為另一副程式。副程式再叫用副程式的情形，即稱為副程式巢串（subroutine nesting）。理論上，副程式巢串可達“無限多”層（level）。圖 10-5 所示即為單層副程式巢串的情形。主程式叫用副程式 SUB 1，在執行的過程中，副程式 SUB 1 又叫用另一副程式 SUB 2。副程式叫用副程式的情形如同前所述者，微處理器執行叫用指令時，程式計數器之現有內

含值（稱為回返位址，return address）被推入堆疊器中存起。

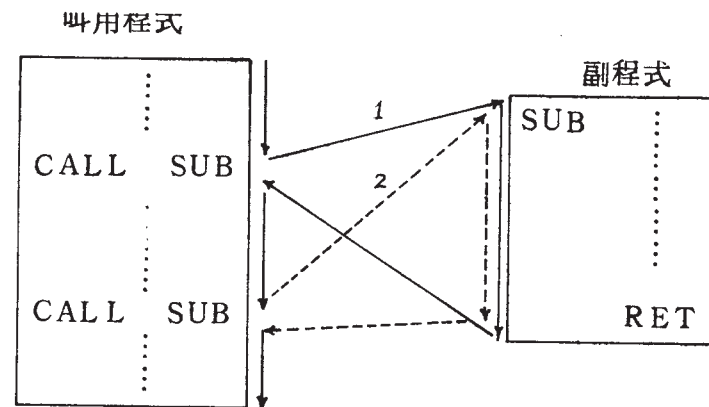


圖 10-4 重複叫用同一副程式

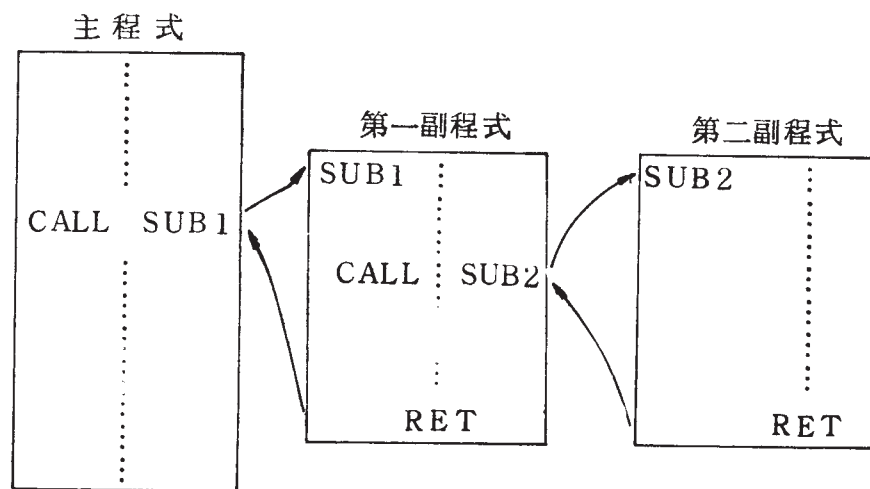


圖 10-5 副程式巢串

爲了方便起見，本書之許多程式都寫成副程式形式。讀者若欲測試此些程式，一則可將副程式之最後指令RET換成手邊機器之停止指令（在Edu-80爲RST 38 H），並加上建立數據資料與預留所需之記憶空間的虛指令即可。不然，您亦可另寫一叫用此些副程式之主程式。

10-2-6 副程式文書

就像程式必須加註解之道理一樣，爲了使他人，甚至連副程式設計者本身，能輕易看懂副程式所爲何事，以及順利將之叫用，副程式通常必須做好文書（documentation）工作。副程式之文書通常必須包括副程式功能，以及叫用系列之說明。此外，有時尚需有暫存器用法，與錯誤情況處理情形之說明。

有關副程式之種種，至此已經都談過了。最後，綜合以上所言，我們舉一副程式例子作爲結論。例10-6之副程式BXOAS，將DE暫存器對內所含之十六位元二進值，轉換成一串八進數字，並將之儲存於一特定之緩衝記憶區。欲傳遞之參數定義於註解欄之叫用系列（calling sequence）中。此一叫喚系列說明了副程式所使用之參數，此些參數如何傳遞，以及副程式如何叫用。叫用系列經常以組合語言指令直接定義。

例 10-6 副程式範例

```

;*****
; 副程式 BXOAS
; 功能：此副程式將DE 暫存器對所含之十六位元二進值，
; 轉換成一含六位數字之ASCII文字串。
; 叫用系列：(HL) = 文字串緩衝記憶區位址 + 5
;          (DE) = 二進值
;          CALL BXOAS
;*****

```

```

; * 回返時，HL 指至第一文數字之前一位置，位 *
; * 址BUFFER - 1。 *
; * 所有CPU 暫存器內含存起。 *
; * 錯誤狀況：無。 *
; *****

```

```

BXOAS  PUSH  DE          ;所有暫存器內含存起。
        PUSH  AF
        PUSH  BC
        LD    C, 6       ;取入反復次數。
LOOP   LD    A, 7       ;面罩(最低三位元爲1)。
        AND   A, E       ;取出目前之八進數字。
        ADD   A, 30 H    ;將之換成ASCII碼。
        LD    (HL), A    ;並存入緩衝記憶區。
        DEC   HL         ;緩衝區指示器內含減一。
        DEC   C          ;迴路計數值減一。
        JR    Z, DONE   ;若六位作完，則準備回返。
        LD    B, 3       ;移位副程式所需之參數。
        CALL MSHRL      ;將DE內含右移三位。
        JP    LOOP      ;繼續下一文數字。
DONE   POP    BC         ;取回所有暫存器內含。
        POP   AF
        POP   DE
        RET

;
; 下面爲右移副程式MSHRL。
;
MSHRL  SRL    D          ;高次八位元先右移一位。

```

```

RR      E      ;再低次八位元右移一位。
DJNZ    MSHRL  ;B 值未零則繼續。
RET

```

關於例 10-6 之副程式範例，有幾點說明如下：

1. 程式 BXOAS 以 RET 指令結尾，故為副程式沒錯。
2. 副程式 BXOAS 之文書井然。程式最前端之註解，不僅說明了副程式之名稱、功能、叫用系列、與錯誤狀況處理情形，而且敘述完整且簡潔。
3. 副程式於運算前先存起所有 CPU 暫存器之內含，並於回返前將之一一復原。（留意推入與取出之次序恰巧相反。）
4. 副程式 BXOAS 叫用了 MSHRL 副程式，MSHRL 無再進一步叫用任何其它副程式，故形成了一單層之副程式巢串。
5. 為了讀者閱讀方便起見，程式文書皆使用中文註解，這在上機之前必須換成英文（若覺必要的話）。
6. 程式之作業情形如下：副程式 BXOAS 之主體為標題 LOOP 以下，至標題 DONE 以前之指令。此一迴路每次將 DE 暫存器內含之最低次三位元取入累加器（LD A, 7 及 AND A, E）。將之轉換成 ASCII 碼（ADD A, 30 H），並存入 HL 所指之緩衝記憶位置。然後，緩衝區指示器之值減一（DEC HL），迴路計數器 C 之值減一。若 C 之值未達零，表 DE 所含之十六位元尚未處理完（因每次處理三位元，故十六位元必須作六次，此乃為何迴路開始前，C 暫存器內含被置定為 6；LD C, 6 之原故。），則 B 暫存器內含置定為 3，並叫用 MSHRL 副程式。

B 暫存器為 MSHRL 副程式內之迴路計數器，該副程式將 DE 暫存器內含右移，右移之位元位置數即為 B 暫存器內含所指明者。（讀者可注意，這又是一個以暫存器傳遞真值的例子。）。DE 內含經右移三位後，JP LOOP 指令使控制跳回 LOOP 處，繼續處理下

一八進 ASCII 數字。

若 DEC C 運算之結果使 C 暫存器之值為零，則表 DE 所含之十六位元皆已處理完畢。JR Z, DONE 指令跳越成功，控制轉移至標題 DONE 處之指令。所有暫存器內含恢復原狀後，控制回返至 BXOAS 之叫用程式。

7. 值得一提的是，由於 LOOP 迴路每次僅處理三個位元，三個位元值最大範圍為 0 至 7，故副程式所產生之文數字串乃稱為八進 ASCII 文數字串。可想像的，若迴路每次處理四個位元，則所產生的必為十六進 ASCII 文數字串。

8. 若 DE 之原先內含為 1101001111011110₂，則副程式執行完後，位址 BUFFER 起之緩衝記憶區之內含將為

位址	內含
BUFFER	3 1 (十六進制)
⋮	3 5
⋮	3 1
⋮	3 7
⋮	3 3
BUFFER + 5	3 6

換言之，此些位置所儲存的即為八進數字串 151736 之 ASCII 碼。

10-3 Z80副程式之特色

以上所談有關副程式之種種，不僅適合於 Z80 微電腦系統，事實上其對任何計算機系統均仍適用。此節，我們介紹一點於副程式作業上，Z80 具有但並非每一系統均有之特色。這包括條件叫用與回返，以及重始指令兩要項。

10-3-1 條件叫用與回返

以上所有副程式之作業均屬無條件叫用與回返。微處理器碰上副程式叫用（或回返）指令時，叫用（或回返）一定發生。於 Z 80，副程式之叫用與回返具有更大之彈性，其可為條件式的。換言之，正如條件跳越指令一般，副程式之叫用與回返指令亦可陳述一條件。當微處理器執行此些指令時。若所述條件滿足，則叫用或回返發生；反之，若條件不滿足，則微處理器繼續執行次一緊接指令。Z 80 之所有條件叫用與回返指令如表 10-4 所摘列，於此一表格中，SUB 代表欲叫用副程式之入口位址。

條件叫用與回返指令之應用如同無條件式。例如，例 10-7 所示即為一條件叫用之情形。該指令系列將某一數值（IX 所指之記憶位置的內含）取入累加器 A。若

例 10-4 條件叫用與回返指令

旗 號	條 件 叫 用 指 令	條 件 回 返 指 令
NZ	CALL NZ, SUB	RET NZ
Z	CALL Z, SUB	RET Z
NC	CALL NC, SUB	RET NC
C	CALL C, SUB	RET C
PO	CALL PO, SUB	RET PO
PE	CALL PE, SUB	RET PE
P	CALL P, SUB	RET P
M	CALL M, SUB	RET M

該數值為負，則程式叫用 ABSVAL（取絕對值）副程式。測試指令 BIT 測試數目之符號位元，若數目為負（符號位元 = 1），零

值旗號（Z）被清除為零。注意，由於 Z 80 取入指令之執行並不影響狀態旗號，故測試指令是必須的。

例 10-7 條件副程式叫用

；該指令系列將某一參數（IX 所指之記憶位置的內含）取入累加器；。若參數值為負，則叫用 ABSVAL 副程式。

；

```
LD      A, (IX)      ; 取入參數值。
BIT     7, A          ; 為負數嗎？
CALL    NZ, ABSVAL    ; 若是，則叫用副程式
        ;             ; ABSVAL。
```

例 10-8 所示則為一條件回返之例子。該副程式求取一八位元數之平方根值的整數部份。控制進入副程式時，數目存於 A 暫存器。控制回返時，平方根值之整數部份存於 B 暫存器。

例 10-8 條件副程式回返（平方根副程式）

；該副程式以連減連續奇數法，求取一八位元數目之平方根值的整數；部份。控制進入副程式時，數目存於 A 暫存器。控制回返時，結果；存於 B 暫存器。

；

```
SQRT    LD      B, 0      ; 平方根值先令為 0。
        LD      C, 1      ; 第一奇數設定為 1。
LOOP     SUB     A, C      ; 數目減奇數。
        RET     M          ; 若已不夠減，則結束。
        INC     C          ; 否則，求取下一奇數。
        INC     C
        INC     B          ; 且平方根值加一。
```

J P LOOP ; 繼續

特別注意，所有控制轉移指令（包括跳越、叫用、與回返）的執行均不影響狀態旗號之內含。這意謂副程式可以某些**狀態旗號**作為**參數**，並將之**傳回**給叫用程式。例 10-9 所示即為一例。該副程式作兩四位元組數目之相加，副程式回返時，最高位元組之進位，與其它旗號一起傳回至叫用程式。

例 10-9 以狀態旗號作參數傳遞

；該副程式將分別儲存於位址 NUM1 與 NUM2 起之連續記憶位置的
；兩四位元組數目相加，結果存於位址 NUM1 起之記憶位置。控制
；回返時，最高位元組之進位與其它旗號，一同傳回至叫用程式。
；

```
ADD 4   LD    BC, ( NUM1+2 )    ; 先取入低次兩位元組。
         LD    HL, ( NUM2+2 )
         ADD   HL, BC                ; 相加。
         LD    ( NUM1+2 ), HL    ; 結果存起。
         LD    BC, ( NUM1 )        ; 再取入高次兩位元組。
         LD    HL, ( NUM2 )
         ADC   HL, BC                ; 連進位一併加入。
         LD    ( NUM1 ), HL        ; 結果存起。
         RET                        ; 回返。
```

10-3-2 重始指令

Z80 之重始指令 RST，主要是力求與 Intel 8080 吻合之努力下的產物。此一指令主要用於插斷之處理。Z80 微處理器具有兩支插斷輸入接腳： $\overline{\text{NMI}}$ 與 $\overline{\text{INT}}$ 。插斷請求輸入 INT 又有三種作業型態：型態 0，型態 1，及型態 2。對小型微電腦而言，NMI 及型

態 1 插斷能力即已夠用。型態 2 插斷主要則用於具有多個不同插斷層次之較大系統。

RST 指令主要使用於型態 0 插斷（此為 8080 唯一僅有之插斷型態），於此一插斷型態，外部週邊設備於認知微處理器為接受插斷請求所送出之 $\overline{\text{IORQ}}$ 與 $\overline{\text{MI}}$ 信號後，會立即將一重始指令置於資料巴士上。

若 RST 指令不用於外部插斷（型態 0），其亦可用作一長一位元組之特殊叫用指令。該指令之功能完全相同於副程式叫用指令，只不過其僅能跳至下列八個零頁位置中之任一個：00 H，08 H，

10 H，18 H，20 H，28 H，30 H，38 H。使用 RST 指令之優點為，若常用之副程式之向量均取自第 0 頁，則程式可因採用一位元組之 RST 取代三位元組之 CALL 指令而節省許多記憶空間。此處的重點是“常用”。若於程式過程中總共有一百次不同之副程式叫用，就節省了 200 個記憶位置。對小系統而言，此一數量已相當可觀。雖然此八個可叫用之位置嚴重地限制了副程式之功用（因每一副程式僅有八個記憶位置），但事實上有些副程式就給八個記憶位置即已夠用。若不，則我們亦可採用類似本章第一節所介紹之表格跳越法，於此八個零頁位置上，存放跳至另外其它記憶區域之跳越指令。例 10-10 所示即為 RST 之典型用法。

例 10-10 RST 之範例

```
ORG    0
LD     ( HL ), A                ; 存起累加器內含。
INC    HL                        ; 加一。
RET
ORG    08 H
LD     A, ( HL )                ; 取入累加器 A。
```



```

        INC     HL           ; 加一。
        RET
        ORG     10 H
        RLCA                ; 累加器內含左旋轉四位。
        RLCA
        RLCA
        RLCA
        RET
        :
        :
STRING EQU 0
LDINC  EQU 08 H
R04LF  EQU 10 H
        :
        :
        RST     LDINC       ; 取入第一位元組。
        RST     STRING      ; 存起。
        RST     LDINC       ; 取入第二位元組。
        RST     STRING      ; 存起。
        RST     LDINC       ; 取入第三位元組。
        RST     R04LF       ; 左旋轉四位。
        AND     A, FH       ; 取下BCD 數字。

```

上述位址 0，08 H，及 10 H 起之三個常式，乃是能置於第 0 頁之常用程式的最典型例子。於上例中，副程式之叫用以含有事先經定義過之參數的 RST 指令為之。例如，RST LDINC 指令即等於 RST 08 H。

10-4 再進入

副程式叫用、回返、與堆疊指令簡易了可再進入 (reentrant) 程式之設計。“再進入”一段程式意表由於插斷原故，某一副程式執行至中途時又需重新執行一次。若收到插斷時能將程式之“環境”(各暫存器之現有內含)存起，並且被再進入之常式不改變共同記憶位置之內含，則再進入毫無問題。若常式改變某些共用記憶位置之內含，則再進入的動作將破壞被再進入程式之前一次執行的結果或部份結果。

試看下面例子。例 10-11 之常式將累加器 A 之內含儲存於一稱為 TEMP 1 之記憶位置。沒有任何插斷動作。儲存動作完後，過了幾個指令，插斷發生。由於此時插斷致能，故插斷開始動作，微處理器開始進入插斷處理常式。於插斷處理過程中，插斷處理常式叫用 FINDIT 副程式。由於剛插斷發生時，微處理器所正在執行的即為此一副程式，故 FINDIT 副程式被重新進入。在此次重進入期間，又有一新數值被存入位址 TEMP 1 之記憶位置。隨後，副程式執行完畢，控制回返至插斷處理常式。過會兒，最後插斷處理常式亦執行完，將環境恢復後，微處理器執行插斷處理常式之最後一指令 RETI 或 RETN，控制回至先前插斷點——BACKHR 之位置，繼續往下執行未完之部份。爾後之指令 LD A，(TEMP 1) 又將 TEMP 1 位置之內含取回，以便作進一步運算。可是，此回取到的已是副程式被再進入時所儲存之數值，而非原先所儲存者了！因此，像此一情況，再進入就破壞了先前之運算結果。換言之，副程式 FINDIT 是無法再進入的。

例 10-11 無法再進入之副程式舉例

```

;
再進入點→ FINDIT :
                :
                :
                :

```


LD (TEMP1), A ; A內含存起。
 LD A, (HL) ; 取次一參數。
 LD B, (IX+30H) ; 取第二數值。
 ADD A, B
 LD (HL), A ; 結果存起。
 LD A, (TEMP1) ; A內含復原。

 RET ; 回返。
 TEMP 1 DEFS 1 ; 臨時記憶區。
 TEMP 2 DEFS 2
 END

插斷發生於此——→
 ↗ BACKHR
 插斷處理常
 式回返至此

有多種方式可解決再進入之問題。其中最簡單的就是避免改變副程式共用記憶區之內含。欲達此目的，最方便之方式就是**以堆疊器作臨時儲存，而勿使用記憶位置**。這不但可免於破壞先前之運算結果，更具備了記憶區容量不受限制之優點。使用堆疊記憶，副程式可再進入之次數毫無受限制。

倘若非使用記憶位置作臨時儲存不可，則再進入仍然可能。不過，此時每一次的再進入必須使用一不同之記憶區。舉個簡單例子而言，假設有五個用者會引起五個不同之插斷。每一插斷之插斷處理均叫用 GETCH 副程式，以讀取次一由鍵盤輸入之文數字，並將之存入用者自己之次一緊接緩衝記憶位置。則爲了使用各自不同之記憶區，插斷處理常式可以一用者號碼叫用 GETCH 副程式。

例 10-12 例 10-13 副程式之叫用

; 下面指令系列乃例 10-13 副程式之叫用系列。叫用前，累加器 A 先存
 ; 用者號碼。該號碼於副程式中被用以計算用者個人所屬緩衝記憶區
 ; 之位址。

```

;
LD      A, 3
CALL    GETCH
OR      A
JP      NZ, AVAIL
  
```

然後副程式 GETCH 再以送來之用者號碼，找到該用者預留之記憶區，以儲存資料或變數。

例 10-13 可再進入副程式舉例

; 該副程式利用叫用程式以 A 暫存器所送來之用者號碼，自鍵盤讀取
 ; 一輸入按鍵，並將之存入該用者之次一緊接緩衝記憶位置。真正讀
 ; 取鍵盤之動作以叫用 READK 副程式達成。

```

;
GETCH  PUSH  AF      ; 用者號碼先存起。
        SLA   A      ; 用者號碼乘 2，然後存入
        LD    C, A    ; BC。
        LD    B, 0
        LD    HL, BUFTB ; 緩衝位址表格起始位址，
        ADD   HL, BC  ; 加表格位移。
        LD    E, (HL) ; 拿取緩衝區位址。
        INC   HL
        LD    D, (HL)
  
```

```

CALL READK      ; 讀取鍵。
JP Z, OUT       ; 若未備好，則離開。
LD (DE), A      ; 讀取文數字存至緩衝區。
OUT RET
BUFTB DEFW BUFF0 ; 用者 0 至用者 4 之緩衝區
      DEFW BUFF1 ; 位址表格。
      DEFW BUFF2
      DEFW BUFF3
      DEFW BUFF4
END

```

該副程式首先將叫用程式以 A 暫存器傳遞過來之用者號碼推入堆疊器存起（注意 Z 80 只有十六位元之推入指令），然後將用者號碼乘 2，變成表格位移（表格中每項緩衝區位址長兩位元組），存入 BC，並加至 HL 暫存器所存之表格起始位址，以指至用者所對應之緩衝記憶區位址。緊接 ADD 後之三個指令即將該緩衝記憶區位址取入 DE 暫存器。副程式然後再叫用 READK 副程式，以讀取自鍵盤輸入之文數字。控制回返至 JP Z, OUT 指令。若輸入文數字未備好，則此時零值旗號為 1，控制轉移至 OUT 處。否則，讀取文數字（於累加器內）被存至緩衝記憶區。

其它避免發生再進入破壞之方法尚有“拒絕”正在使用中之副程式被叫用，於執行某些關鍵指令時將插斷禁能，以及分別對每一不同層次之用者複製一份不同之程式等等！不過，上述之技巧仍是最常用的，而且，以堆疊器作臨時記憶之方法最乾淨俐落。

常用副程式

於介紹過如何以 Z 80 指令集之指令構成各種應用程式，以及與副程式有關之各種問題後，本章，我們整理出一些日常最常用之副程式。一方面可供讀者參考之便，它方面可作為前面幾章對程式設計討論之結論。

本章所收容之副程式包括比較副程式、計時迴路、乘除算副程式、多段算術常式，ASCII 文數字串至二進、十進、十六進、與 BCD 數間之轉換，資料填補常式，資料串比較常式，找最大值常式，排序副程式，與搜尋副程式等。雖然這並不包括所有之常式，但它們的確代表了計算機應用上經常反覆要作之常事，並且可作為系統之**庫存副程式**（library routine）的一部份。本章所討論之每一副程式皆以一個別功能**單元**（module）之觀點加一考慮，而非以視為一程式片段之觀點。系統設計可採用諸如此類之副程式，配合系統之功能要求，以達成一種“由上而下”（系統要求）與“由下而上”（較低功能階層之副程式）兩種技巧組合之設計。

讀後，讀者可發覺此些副程式採用了一切能最有效達成任務之技巧，因此，每一副程式皆顯得相當精簡。此外，每一副程式在指令開始之前，皆加有一段詳細之註解，使讀者看過此一段註解後，不僅能立即了解每一副程式所為何事，並且知如何加以叫用。

11-1 比較副程式

許多決策經常皆根據某一運算元與另一運算元相比之結果而定。最簡單的比較副程式就是將一八位元之無號數 A 與另一八位元之無號數 B 相比，然後顯示 $A < B$ ， $A \leq B$ ， $A = B$ ， $A \geq B$ ，或 $A > B$ 之結果。假設副程式叫用指令後緊接的即為根據比較之不同結果，使控制分別轉移至適當之指令系列的相對跳越指令，則例 11-1 之比較副程式的叫用系列即為

```
LD      A, (OPERA)    ;
LD      B, (OPERB)    ;
CALL    LMPARE        ; 叫用比較副程式。
JR      LTHAN         ; A < B 時回返至此。
JR      EQUAL         ; A = B 時回返至此。
JR      GTHAN         ; A > B 時回返至此。
```

正如某些高階語言一樣， $A \leq B$ 或 $A \geq B$ 之等式亦可經適當地使用回返點加以實現。譬如，下面之叫用系列則為當 $A \leq B$ 時，使控制轉移至 LTEQL 處，而當 $A > B$ 時，控制轉移至 GTHAN 處之指令系列的雙途分叉式決策。

```
LD      A, (OPERA)    ; 取入無號數 A。
LD      B, (OPERB)    ; 取入無號數 B。
CALL    CMPARE        ; 叫用比較副程式。
JR      LTEQL         ; A < B 及 A = B 時，
JR      LTEQL         ; 均跳至 LTEQL 處。
JR      GTHAN         ; A > B 時跳至 GTHAN。
```

兩八位元無號數之比較副程式如下。

例 11-1 兩八位元無號數之比較副程式

；該副程式將累加器 A 與暫存器 B 所含之兩八位元無號數相比。依 A ； $< B$ ， $A = B$ ，及 $A > B$ 之不同結果，控制分別回返至叫用指令之 ；後第一、三、及五個記憶位置。

```
；
CMPARE  CP      B          ; A 比 B
        EX      (SP), HL   ; 提取回返位址。
        JR      Z, EQUAL   ; 若 A = B 則跳出。
        PUSH    AF        ; 否則，A 與旗號存起。
        XOR     B
        JP      P, SAME    ; 若同號，則跳出。
        POP     AF        ; 恢復 A 與旗號值。
TEST    JP      C, LESST   ; A < B。
        JP      GREAT      ; A > B。
SAME     POP     AF
        CCF
        JP      TEST
GREAT    INC     HL        ; 若 A > B，則回返
        INC     HL        ; 位址加 4。
EQUAL    INC     HL        ; 若 A = B，則回返位址加
        INC     HL        ; 2。
        EX      (SP), HL   ; 修正後之回返位址存回堆
        ; 疊器。
        RET              ; 回返。
```

兩十六位元有號數相比

實際應用所處理之數目通常超乎八位元。例 11-2 所示即為比較兩十六位元有號數 N1 與 N2 之副程式。若 $N1 < N2$ ，則副程式將進位旗號置定為 1；若 $N1 = N2$ ，則零值旗號置定為 1。若當 $A < B$ ， $A = B$ ，及 $A > B$ 時，控制分別欲轉移至 LTHAN， EQUAL，及 GTHAN 等記憶位置，該副程式之叫用系列即如

```
LD      IX, NUM1    ; IX 指至第一數目。
LD      IY, NUM2    ; IY 指至第二數目。
CALL    COMP 16      ; 叫用比較副程式。
JP      C, LTHAN     ;  $N1 < N2$ ，跳至 LTHAN。
JP      Z, EQUAL     ;  $N1 = N2$ ，跳至 EQUAL。
JP      GTHAN        ;  $N1 > N2$ ，跳至 GTHAN。
```

其中，如圖 11-1 所示地，NUM1 與 NUM2 分別代表兩十六位元數目所在位置之地址。

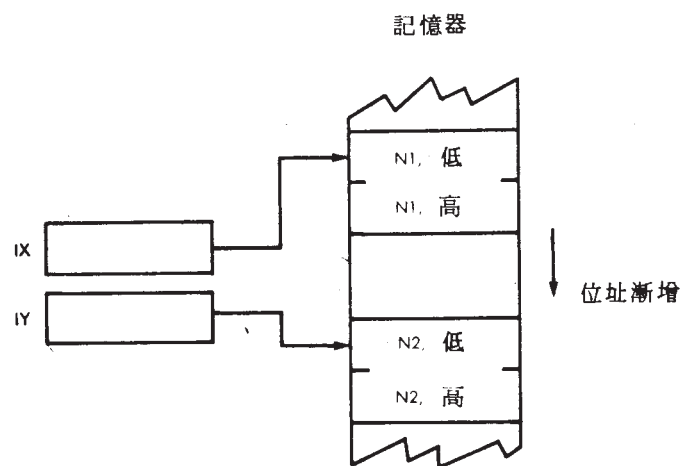


圖 11-1 兩十六位元有號數相比

例 11-2 兩十六位元有號數相比之副程式

；該副程式將兩十六位元之有號數 N1 與 N2 相比。若 $N1 < N2$ ，進位旗號被置定為 1；若 $N1 = N2$ ，零值旗號被置定為 1。控制進入前，索引暫存器 IX 與 IY 必須分別指至 N1 與 N2。

```
COMP 16  LD      B, (IX+1)    ; 取得 N1 之符號。
          LD      A, B
          AND     80H          ; 並加測試，清除進位。
          JR      NZ, NEGN1    ; 若負，則跳至 NEGN1。
          BIT     7, (IY+1)    ; 測試 N2 之符號位元。
          RET     NZ           ; N1 正，N2 負，則回返。
          LD      A, B          ; 兩者皆正。
          CP      (IY+1)       ; 兩高次位元組相比。
          RET     NZ           ; 若不等，則大小已分明，
                                ; 故回返。
          LD      A, (IX)       ; 否則，低位元組再比。
          CP      (IY)          ;
          RET           ; 結果記錄於 C 及 Z 旗號。

;
NEGN1    XOR     (IY+1)        ; 兩符號位元 XOR。
          RLA              ; 結果移入進位旗號。
          RET     C           ; 若兩數異號 (N1 負, N2
                                ; 正) 則回返。
          LD      A, B          ; 兩數均為負。
          CP      (IY+1)       ; 兩高次位元組相比。
          RET     NZ           ; 若不等，則大小已分明，
                                ; 回返。
          LD      A, (IX)       ; 兩低次位元組再比。
```

CP (IY) ; 結果記錄於 C 及 Z 旗號。
RET

程式首先測試 N1 與 N2 之正負號。若 N1 為負，控制跳至 NEG-N1，否則，微處理器執行上半部之程式指令。於兩種情況，唯有當 N1 與 N2 同號時，程式才真正將兩數相比。高位元組相比，若兩高次位元組不等，則回返，此時 N1 與 N2 之大小關係已記錄於進位旗號中。（譬如，若 $N1 < N2$ ，則比較相減結果不夠減，進位旗號之值為 1。）否則，若兩高次位元組相等，程式再將兩低次位元組相比，然後再回返。兩數之大小關係同樣記錄於進位旗號與零值旗號中。

11-2 計時副程式

計時迴路可用於產生輸入 / 輸出處所需之可變時間延遲，提供即時時鐘計時所需之時限，或提供等候操作員反應（如按下一鍵）所需之延遲。日常所需之計時迴路最好能產生從幾毫秒至大約 30 秒左右之時限。若平均指令執行時間為 2.5 秒（以 4MHz 之時序計），則 30 秒延遲必須執行 12000000 個指令。若每一層迴路能延遲約 3000 計數，或其中一迴路延遲 300 計數而另一迴路延遲 30000 計數，那雙層之巢串迴路即已夠用。

例 11-3 延時副程式

；該副程式以雙層巢串計時迴路，產生 0.1 秒至 25.6 秒之時間延遲；
；調整差距為 0.1 秒。外層迴路之計數器為 B 暫存器。B 初值為 1；
；時，副程式產生最短延遲 0.1 秒。B 之值每加 1，延時即增加 0.1
；秒。控制進入副程式前，B 值必須先設定好。更短或更長之延遲可
；由調整 H L 之初值達成，其值每加一，可增加 5.75 微秒之延遲。
；

DELAY	LD	DE, -1	；作遞減用。
LOOP1	LD	HL, 14814	；內層迴路計數值。
LOOP2	ADD	HL, DE	；內層迴路計數減 1。
	JR	C, LOOP2	；非零則繼續。
	DJNZ	LOOP1	；外層迴路計數減 1，非零 ；則繼續。
	RET		；計時完畢，回返。

內層迴路以 H L 暫存器對為計數器，其初值設定為 14814。H L 自此一初值遞減至零需費時 0.1 秒。外層迴路以 B 暫存器為計數器，其初值由叫用程式設定。副程式所產生之時間延遲為

時間延遲 = B 之初值 \times 0.1 秒（近似）

。因此，最短延遲（B 初值為 1）為 0.1 秒；而最長延遲（B 初值為 0）為 25.6 秒。更長或更短之時間延遲可由調整 H L 之起始值達成。該暫存器對之初值每增加（減少）一，即可增加（減少）5.75 微秒（以 4MHz 時序計）之時間延遲。

11-3 乘除副程式

兩八位元無號數之乘算已於第七章介紹過。不過，八位元運算元對多數應用而言，似乎較拘限了些。十六位元對十六位元之乘算較能涵蓋大多數微電腦之應用；若這一長度還不夠，那就得採用浮點乘算了。於下面所舉之十六位元乘算副程式中，乘數（無號數）由 D E 暫時器輸入，被乘數（無號數）由 B C 暫存器輸入，四位元組之結果則由 D E 及 H L 兩暫存器對傳回至叫用程式。

例 11-4 十六位元乘算副程式

；該副程式將叫用程式由 D E（乘數）及 B C（被乘數），兩暫存器
；對送來之兩十六位元無號數相乘。所得結果由 D、E、H、及 L 等

; 暫存器送回至叫用程式。

```

;
MULT16  LD    A, 16      ; 反複次數。
        LD    HL, 0      ; 乘積低半部先令為零。
LOOP    BIT    7, D      ; 測試次一乘數位元。
        JP    Z, JUMP1   ; 若零，則略過加算。
        ADD   HL, BC     ; 否則，被乘數加至部份積。
        JP    NC, JUMP1  ; 若無進位，略過次一指令。
        INC   DE         ; 進位至乘積之高半部。
JUMP1   DEC    A         ; 反複次數減一。
        RET    Z         ; 若零，則完畢，回返。
        EX    DE, HL
        ADD   HL, HL     ; D E之乘數左移一位。
        EX    DE, HL
        ADD   HL, HL     ; HL之部份積左移一位。
        JP    NC, LOOP   ; 若無進位，則繼續。
        INC   DE         ; 否則，進位加至上半乘積。
        JP    LOOP      ; 然後繼續。

```

以上為兩十六位元無號數之相乘。若欲求**兩有號數**之乘積，程式必須增加核對符號位元以及取運算元之絕對值（若數目為負時）之指令。此一指令系列可緊接叫用上述無號數之乘算副程式MULT16，求得絕對值乘積。然後再將所得乘積轉換成適當之正負號。一負數之絕對值可以NEG指令求得。而符號位元之核對可由將兩運算之符號位元互作XOR運算輕易達成。例如：

; 下面之指令系列決定BC與DE 兩暫存器對所含之兩十六位元有號數之乘積究應為正或為負。若正，則位址SIGN之記憶位置存0；

; 若負，則該記憶位置存-1。

```

;
        LD    A, D      ; 取入第一運算元之符號。
        XOR   B         ; 兩符號位元XOR。
        LD    A, 0      ; 指示旗號值先令為0。
        JP    P, JUMP2  ; 若正乘積，則略過。
        CPL                     ; 否則，指示旗號值為-1。
JUMP2   LD    (SIGN), A ; 指示旗號值存至SIGN處。

```

八位元無號數除十六位元無號數之除算例子已於第七章舉過。正如八位元對八位元之乘算一般，就實際應用而言，此些運算元亦顯過小了一點。下面我們所舉的例子為十六位元之無號數除三十二位元之無號數。

例 11-5 32 對 16 位元之除算副程式

; 該副程式將叫用程式以H，L，D，及E暫存器送來之三十二位元無號數，除以以BC暫存器對送來之十六位元無號數。十六位元之商由DE暫存器對送回至叫用程式。餘數亦由HL暫存器對送回；

```

DVDE16 LD    A, 16      ; A為迴路計數器。
LOOP   ADD   HL, HL     ; HL左移一位。
        EX    DE, HL    ; DE亦左移一位。
        ADD   HL, HL
        EX    DE, HL
        JP    NC, JUMP1 ; 無進位時則略過INC。
        INC   HL        ; 進位加至高次兩位元組。
JUMP1  OR     A         ; 清除進位。
        SBC   HL, BC    ; 被除數減除數。

```

	INC	DE	; 先設商為 1。
	JP	P, JUMP2	; 若真，則略過。
	ADD	HL, BC	; 否則（不夠減），恢復。
	RES	0, E	; 被除數，且商令為 0。
JUMP2	DEC	A	; 迴路計數值減一。
	JP	NZ, LOOP	; 未完則繼續。
	RET		; 除完了，回返。

除算結束時，十六位元之商存於 D E 暫存器對，而餘數（若有的話）存於 H L 暫存器對。

有號數之除算可以類似於有號數乘算之方法進行。其主要步驟為：

- 1 兩運算元之符號位元作 XOR 運算，以求得商數之正負號。
- 2 兩運算元分別取絕對值。
- 3 叫用無號數除算副程式求商數之絕對值。
- 4 將商數與餘數轉換成適當之符號。注意，餘數恒與被除數同號。

11-4 多段算術常式

此節，我們介紹一般性之多段加 / 減副程式。運算元之長度可為幾段：一個位元組至數個位元組。

多段加法常式將 D E 所指位置起之多倍長來源運算元，加至 H L 所指位置起之多倍長目的運算元，結果存於目的運算元之位置。

例 11-6 多段加法常式

; 此副程式將 D E 所指位置起之 n 倍長來源運算元，加至 H L 所指位
; 置起之 n 倍長目的運算元，結果存於目的運算元位置。運算元之長
; 度 n（以位元組計）含於 C 暫存器。回返時，進位旗號顯示最後一
; 次（最高次位元組）加算之進位情況。
;

MPADD	LD	B, 0	; 長度 n 存於 B C。
	ADD	HL, BC	; 指至最低次位元組加 1。
	DEC	HL	; 指至最低次位元組。
	EX	DE, HL	; 亦使 D E 指至來源運算元
	ADD	HL, BC	; 之最低次位元組。
	DEC	HL	
	EX	DE, HL	
	OR	A	; 進位清除為零。
LOOP	LD	A, (DE)	; 取入來源位元組。
	ADC	A, (HL)	; 與目的位元組相加。
	LD	(HL), A	; 結果存回目的位置。
	DEC	HL	; 目的指示器值減一。
	DEC	DE	; 來源指示器值減一。
	DEC	C	; 位元組數減一。
	JR	NZ, LOOP	; 未完時繼續。
	RET		; 回返。

除了來源運算元減去目的運算元外，多段減算常式與多段加算常式無異。

例 11-7 多段減算常式

; 此副程式將 D E 所指位置起之 n 倍長來源運算元，減去 H L 所指位
; 置起之 n 倍長目的運算元，結果存於目的運算元位置。運算元之長
; 度 n（以位元組計）含於 C 暫存器。回返時，進位旗號反映了最後
; 一次（最高次位元組）減算之借位。
;

MPSUB	LD	B, 0	; 長度 n 存於 B C。
	ADD	HL, BC	; 指至最低次位元組加一。
	DEC	HL	; 指至最低次位元組。
	EX	DE, HL	; 亦使 D E 指至來源運算元
	ADD	HL, BC	; 之最低次位元組。
	DEC	HL	
	EX	DE, HL	
LOOP	OR	A	; 清除借位。
	LD	A, (DE)	; 取入來源位元組。
	SBC	A, (HL)	; 減去目的位元組。
	LD	(HL), A	; 結果存於目的位置。
	DEC	HL	; 目的指示器值減一。
	DEC	DE	; 來源指示器值減一。
	DEC	C	; 位元組數減一。
	JR	NZ, LOOP	; 未完時繼續。
	RET		; 完了回返。

11-5 ASCII 至基底 X 之轉換

由於目前計算機（尤以微電腦）系統所使用之輸入／輸出設備幾乎都採用 ASCII 寫碼，以致，計算機之應用經常須將代表一項二進、十進、十六進、或 BCD 數目之一串 ASCII 文數字，轉換成某一適當基底之數目。因此，若有此些轉換副程式，使用上將甚為方便。藉著使用此些副程式，資料可直接由鍵盤輸入，並經轉換成程式運算所需之數據（適當基底）。

本節以下所舉之常式。主要觀點皆為：將 HL 所指位置起之一串 ASCII 文數字（或許剛從鍵盤輸入），轉換成適當之內部運算所需的格式。代表二進數之 ASCII 文數字將每次八個一起轉換，因為，八個位元（轉換所得結果）正好可容納於一八位元之暫存器。代表十

六進數之 ASCII 文數字每次將兩個一起轉換，因為，八位元正巧可容納兩個十六進數字。同理，ASCII BCD 文數字每次亦兩個一起轉換。若輸入資料代表 ASCII 十進數字，則問題就變成一轉換結果所佔長度的問題了。亦即，轉換結果所得十進數之值究竟有多大？八位元僅能容納 0 至 255 之十進數，十六位元所能容納之最大十進數為 65536。由於欲轉換之 ASCII 十進數的大小未知，致此一情況就不如二進、十六進、及 BCD 之情況單純。因此，底下之十進轉換副程式假定輸入資料長六個 ASCII 文數字。因為，這代表了輸入至程式之數值的較合理範圍。於每一種情況，每次轉換後，文數字串之指示器值的累增值即等於已轉換之文數字的個數，使得再轉換的是次一文數字串的第一個文數字。

下面之常式 ASXBIN，將八個 ASCII 文數字（假設均為 0 與 1），轉換成一八位元之二進值，存於累加器 A。

例 11-8 ASCII 至二進數之轉換

；該副程式將八個 ASCII 文數字（假設均為 0 或 1），轉換成一八位元之二進值，存於累加器 A。控制進入時，HL 指至欲轉換文數字串之第一個文數字；回返時，HL 指至文數字串之第九個文數字；

ASXBIN	LD	B, 8	; 設定迴路計數值。
	LD	C, 0	; 結果先清除為零。
LOOP	SLA	C	; 結果先左移一位。
	LD	A, (HL)	; 獲取數字之 ASCII 碼。
	INC	HL	; 先指至次一文數字。
	SUB	30H	; 轉換成二進數 0 或 1。
	OR	A, C	; 與舊結果合併，新數字位
			; 於最低次位元。
	LD	C, A	; 新結果存入 C。

```

DJNZ    LOOP    ; 未達八次時則繼續。
RET      ; 完了，回返。

```

下面副程式為ASCII 十六進數字之轉換。

例 11-9 ASCII 至十六進數之轉換

；此副程式將兩位 ASCII 十六進數字，轉換成一八位元值存於累加器 A。第一數字值存於高次四位元，第二數字值存於低次四位元。
；控制進入時，HL 指至文數字串之第一文數字；回返時，HL 指至文數字串之第三個文數字。

```

;
ASXHEX  LD      C, 0          ; 結果先清除為零。
         LD      A, (HL)      ; 拿取第一文數字。
         CALL    CVERT        ; 轉換。
         INC     HL
         LD      A, (HL)      ; 拿取第二文數字。
         CALL    CVERT        ; 轉換。
         INC     HL           ; 指至第三文數字。
         RET
CVERT   SLA      C            ; 先調整結果，以空出低次
         SLA      C            ; 四位元。
         SLA      C
         SLA      C
         SUB     30H          ; ASCII 轉換成十六進。
         CP      A, 10        ; 數字為 A 至 F 嗎？
         JP      M, JUMP1     ; 若非，則結果正確。
         SUB     A, 7         ; 否則，必須再減 7。
JUMP1   ADD      A, C         ; 兩數字值合併。

```

```
RET
```

緊接為 ASCII BCD 轉換副程式。

例 11-10 ASCII 至BCD之轉換

；此副程式將兩位 ASCII 文數字（假設均為 ASCII BCD 數字 0 至 9），轉換成代表兩 BCD 數字之八位元值，存於累加器 A。控制；進入前，HL 指至文數字串之第一文數字；回返時，HL 指至文數字串之第三個文數字。

```

;
ASXBCD  LD      A, (HL)      ; 拿取第一文數字。
         INC     HL          ; 先指至第二文數字。
         SUB     A, 30H      ; 第一數字轉換成BCD。
         RLCA              ; 並調整至第 4 至 7 位元。
         RLCA
         RLCA
         LD      C, A        ; 部份結果先存起。
         LD      A, (HL)     ; 取入第二文數字。
         SUB     A, 30H
         ADD     A, C        ; 兩數字合併，結果於 A。
         INC     HL          ; 指至次一文數字。
         RET                ; 回返。

```

最後一個為十進轉換常式。

例 11-11 ASCII 至十進數之轉換

；此副程式利用移與加技巧作十倍運算，將一五位之 ASCII 十進數

; (包括前導零), 轉換成其對等之十進數值, 存於 HL 暫存器對。
 ; 控制進入時, IX 指至文數字串之第一文數字; 回返時, IX 指至文
 ; 數字串之第六個文數字。

```

;
ASXDEC  LD      B, 5      ; 設定迴路計數器初值。
        LD      HL, 0     ; 結果先清除為零。
LOOP    ADD     HL, HL     ; 結果兩倍。
        PUSH    HL        ; 然後存起。
        ADD     HL, HL     ; 結果四倍。
        ADD     HL, HL     ; 結果八倍。
        POP     DE
        ADD     HL, DE     ; 結果十倍。
        LD      A, (IX)    ; 拿取次一 ASCII 數字。
        SUB     A, 30H     ; 轉換成 0 至 9。
        LD      E, A       ; 並存入 DE。
        LD      D, 0
        ADD     HL, DE     ; 加至部份結果。
        INC     IX         ; 先指至次一數字。
        DJNZ    LOOP      ; 未完時繼續。
        RET              ; 完了, 回返。
  
```

圖 11-2 所示即為以上四個 ASCII 至基底 X 之轉換副程式的作業說明圖。

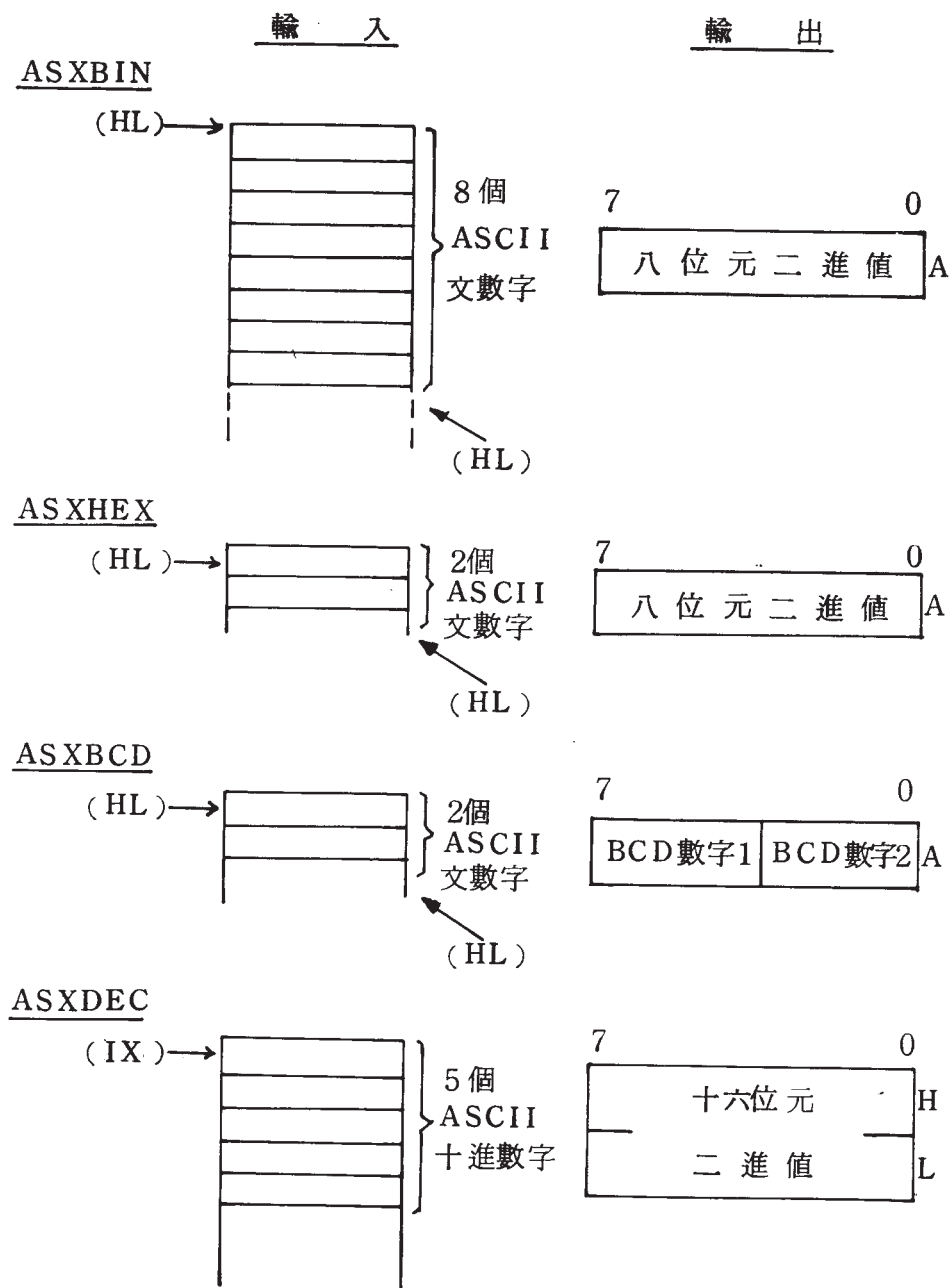


圖 11-2 ASCII 至基底 X 之轉換

11-6 基底 X 至 ASCII 之轉換

此一組之轉換與前一節所述之 ASCII 至基底 X 轉換正巧相反。此回，一二進數值經轉換成 ASCII 二進數、十六進數、BCD 數、或十進數。控制進入副程式之際，指示器指至 ASCII 結果所欲儲存之緩衝記憶區。控制回返時，指示器指至緩衝記憶區次一可用之位置。

下面常式為二進數至 ASCII 二進文數字串之轉換。

例 11-12 八位元二進值轉換成 ASCII 二進文數字串

；此副程式將 C 暫存器所含之八位元二進值，轉換成八個 ASCII 二進文數字，每一位元對應一文數字。所得結果存於 HL 所指之緩衝記憶區。控制進入副程式時，HL 指至緩衝區第一位置；回返時，HL 指至緩衝區第八位置。

；

```

BXASB  LD    B, 8          ; 設定迴路計數值。
LOOP   LD    A, 30H        ; “ 0 ” 之 ASCII 碼。
        BIT   7, C          ; 測試 C 之最高次位元。
        JP    Z, JUMP1      ; 若 0，則略過。
        INC   A             ; “ 1 ” 之 ASCII 碼。
JUMP1  LD    (HL), A        ; ASCII 文數字存起。
        SLA   C             ; 準備一下位元。
        INC   HL            ; 指至次一緩衝記憶位置。
        DJNZ  LOOP          ; 未完則繼續。
        RET                   ; 完了回返。

```

此一系列之第二副程式為八位元二進值至兩位 ASCII 十六進數字之轉換。

例 11-13 八位元二進值轉換成兩位 ASCII 十六進數字

；此副程式將 C 暫存器所含之八位元二進值，轉換成兩位 ASCII 十六進數字。控制進入副程式時，HL 指至儲存結果之緩衝記憶區的第一位置；回返時，HL 指至緩衝區第三位置。

；

```

BXASH  LD    A, F0H        ; 面罩。
        AND   A, C          ; 獲取第一數字 (C 之 4MSBs)
        RRCA                      ; 調整至低次四位元以便轉換
        RRCA                      ;
        RRCA                      ;
        CALL  CVERT         ; 轉換並儲存。
        LD    A, 0FH        ; 面罩。
        AND   A, C          ; 獲取第二數字 (C 之 4LSBs)
        CALL  CVERT         ; 轉換並儲存。
        RET                   ; 完了，回返。
CVERT  CP    A, 10           ; 數字介乎 A 至 F 嗎？
        JP    M, JUMP1      ; 若非，則略過次一指令。
        ADD   A, 7           ; 若是，則先加 7。
JUMP1  ADD   A, 30H          ; 轉換成 ASCII 碼。
        LD    (HL), A        ; 結果存入緩衝記憶區。
        INC   HL            ; 指至次一緩衝記憶位置。
        RET

```

下面副程式將一八位元數值轉換成兩 ASCII 十進數字。

例 11-14 濃縮 BCD 轉換成兩位 ASCII 十進數字

；此副程式將 C 暫存器所含之八位元數值（假設為一濃縮 BCD 數）
 ；，轉換成兩位 ASCII 十進數字 0 至 9。控制進入副程式時，欲轉
 ；換之數值存於 C 暫存器，HL 指至欲儲存轉換結果之緩衝記憶區的
 ；起點。控制回返時，HL 指至緩衝記憶區第二記憶位置。

```

;
BXBCD    LD      A, F0H      ; 面罩。
          AND      A, C       ; 取得第一 BCD 數字。
          RRCA          ; 調整至低次四位元以便轉換
          RRCA          ; 。
          RRCA
          RRCA
          ADD      A, 30H     ; 轉換成 ASCII 十進數字。
          LD       (HL), A    ; 然後存起。
          LD       A, 0FH     ; 面罩。
          AND      A, C       ; 取得第二 BCD 數字。
          ADD      A, 30H     ; 轉換成 ASCII 碼。
          LD       (HL), A    ; 並且存起。
          INC      HL         ; 指至次一緩衝位置。
          RET

```

最後一個常式將 HL 暫存器對所含之十六元二進值，轉換成一對等之五位 ASCII 十進數。控制進入副程式時，IX 指至 ASCII 文數字記憶區之起點；回返時，IX 指至緩衝區起始位址加 5 之位置。為了避免 16 位元對 8 位元之除算產生餘數，程式採用查表法以獲得 10 之乘幂。查取表格含 10^4 ， 10^3 ， 10^2 ， 10^1 ，以及 10^0 等五項元素。並以迴路之現有反複計數值作索引存取。各數位（如十位，百位，千位等）之係數值乃以連減 10 之乘幂值求得。夠減之次數即為該相對數位上之係數值。

例 11-15 十六位元二進值轉換成五位 ASCII 十進數

；該副程式將 HL 暫存器對所含之十六位元二進值，轉換成一對等之
 ；五位 ASCII 十進數。控制進入副程式時，IX 暫存器指至儲存轉
 ；換結果之記憶區的起點。程式以連減 10 之乘幂值，獲得十進數各
 ；數位之係數值。各乘幂值則由查取表格，以迴路之現有反複計數值
 ；作索引取得。

```

;
BXDEC    LD      IY, P10TAB ; 指至十乘幂表格。
LOOP0    XOR      A         ; 數位係數值 = 0。
          LD       D, (IY)
          LD       E, (IY+1)
LOOP1    OR        A         ; 清除進位旗號。
          SBC      HL, DE     ; 減去十乘幂。
          JP       NC, JUMP1 ; 不夠減時就跳出。
          INC      A         ; 數位係數值加 1。
          JP       LOOP1
JUMP1    ADD      HL, DE     ; 減過頭了必須加回。
          LD       (IX), A    ; 數位係數值存起。
          INC      IX         ; 緩衝區指示器加一。
          INC      IY         ; 指至次一十乘幂值。
          INC      IY
          CP       E, 1       ; 已作完五位數字了嗎？
          JP       NZ, LOOP0 ; 若非；則繼續。
          RET
P10TAB   DEFW     10000
          DEFW     1000
          DEFW     100
          DEFW     10

```

DEFW 1

圖 11-3 所示即為以上四個基底 X 至 ASCII 之轉換副程式的作業說明。

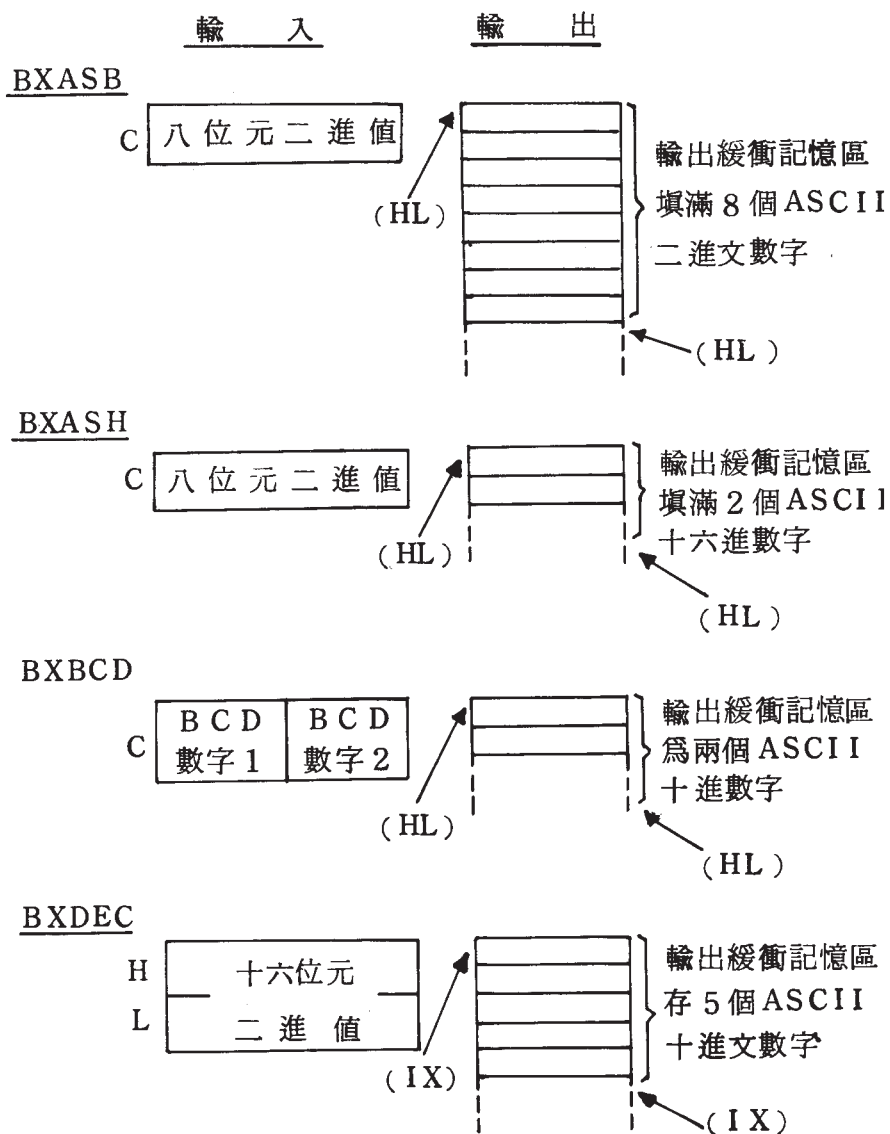


圖 11-3 基底 X 至 ASCII 轉換

11-7 資料填補常式

下面常式將一特定八位元資料，填入某一段記憶區域。該常式在將某一表格或某一系列記憶位置清除為零，或將某一已知資料值填入緩衝記憶區（例如，將印字機之緩衝記憶區皆填成空白）時，甚為好用。

例 11-16 資料填補常式

；該副程式將叫用程式以 A 暫存器送來之數值，填入 HL 所指位置起；之連續一系列記憶位置。欲填補之記憶位置數存於 B 暫存器。控制；進入副程式時，A，B，及 HL 等暫存器之值皆須先設定好。

```
FLDATA LD (HL), A ; 填入位元組資料。
        INC HL      ; 指至次一記憶位置。
        DJNZ FLDATA ; 未完時繼續。
        RET         ; 完了就回返。
```

11-8 資料串比較常式

以下常式比較兩個資料串。資料串之長度可為任意，但最高 256 個位元組。典型上，資料串可為一表格之搜取鍵或一段文章。回返時，副程式顯示了資料串 A 是否小於、等於、或大於資料串 B。於叫用時，資料串 A 與資料串 B 之位址分別以 DE 及 HL 暫存器對傳遞至副程式。兩資料串之位元組數（當然相等）則以 B 暫存器傳遞。若 B 之初值為 0，則副程式總共比較 256 個位元組。回返後，若 $A < 0$ ，則表資料串 A < 資料串 B；若 $A = 0$ ，則資料串 A = 資料串 B；若 $A > 0$ ，則表資料串 A > 資料串 B。所有比較皆為八位元無號數之比較。

例 11-17 資料串比較常式

；此副程式比較兩個資料串A與B。所有比較皆為八位元無號數之比較。控制回返時，依據資料串A是小於、等於、或大於資料串B，累加器A之內含分別為負、零、或正。控制進入副程式時，資料串A與B之位址分別以DE及HL暫存器對傳遞至副程式。資料串之長度（位元組數）則以B暫存器傳遞。若B之初值為0，則副程式總共比較256個位元組。

```
COMSTR    LD      A, (DE)      ; 取入資料串A。
           SUB     (HL)        ; 減去資料串B。
           RET     NZ          ; 若不等，則回返。
           INC     DE          ; 指至次一位元組。
           INC     HL
           DJNZ    COMSTR      ; 未完時繼續。
           RET              ; 完了則回返。
```

11-9 找最大值副程式

於許多應用上，諸如統計、實驗資料之處理或排序（sorting），經常我們必須找出一系列資料（稱為表列）之最大值一項。此等應用副程式之一般作法皆是：設定一起始最大值，將表列每一項元素與之相比，若元素值較小（或相等），則繼續比次一元素；否則，元素值令為新最大值，再繼續比次一元素。圖11-4所示即為此一演算法之流程圖。

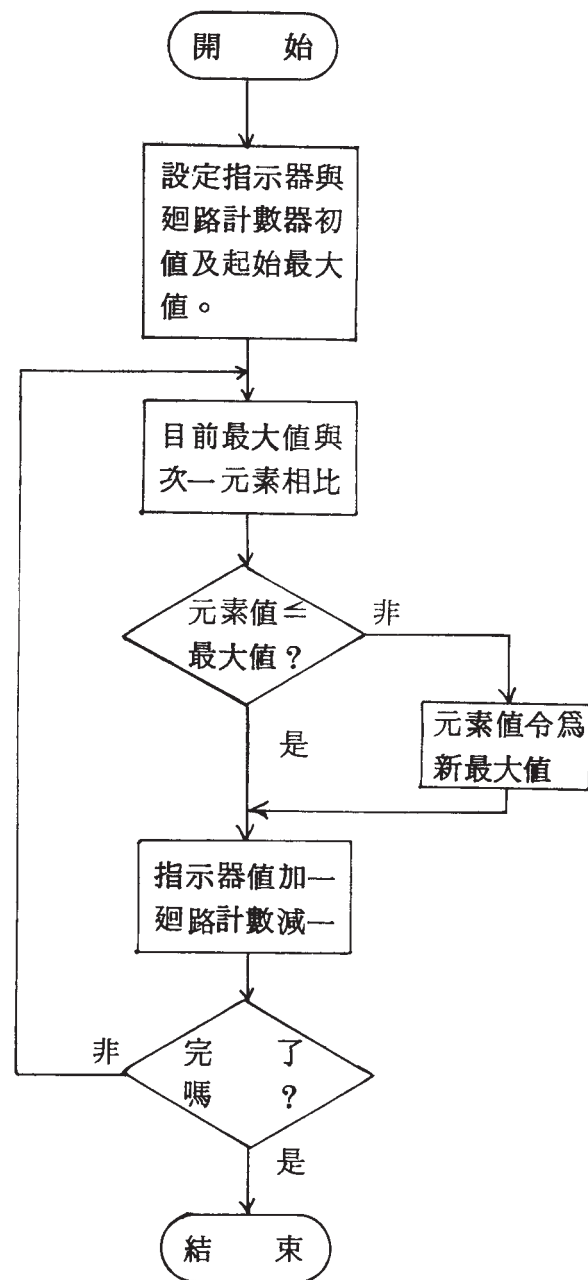


圖 11-4 找最大值元素之演算流程圖

以下即為找最大值之副程式。該副程式之資料佈置圖如圖 11-5 所示。

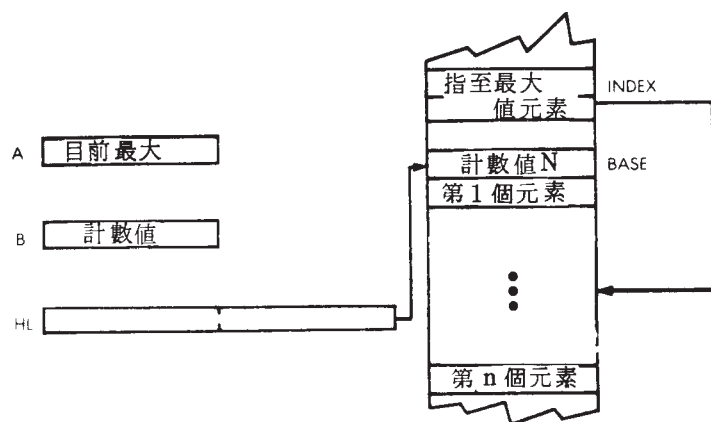


圖 11-5 找出表格之最大值元素——資料佈置圖

例 11-18 找最大值副程式

；該副程式找出位址 BASE 起之表格資料中，數值最大的一項。控
；制回返時，此一最大值存於累加器 A，其記憶位址存於位址 INDE
；X 起之記憶位置。表格之第一項資料即為表格所含之位元組數。該
；副程式使用 A，B，H，及 L 等暫存器。

```

MAX      LD      HL, BASE      ; 取入表格起始位址。
          LD      B, (HL)      ; 取入表格元素個數。
          LD      A, 0         ; 起始最大值設為零。
          LD      (INDEX), HL  ; 設定索引初值。
          INC     HL           ; 指至第一項資料。
LOOP     CP      (HL)          ; 最大值與元素值比。
          JR      NC, NOCHGE   ; 元素 ≤ 最大值則繼續。

```

```

LD      A, (HL)      ; 否則，元素值令為新最大值。
LD      (INDEX), HL ; 且其位址存起。
NOCHGE  INC     HL     ; 指至次一元素。
DEC     B            ; 位元組個數減一。
JR      NZ, LOOP    ; 未完時繼續。
RET                               ; 完了就回返。

```

資料處理上有兩項非常常用且重要之工作，那就是**排序**（sorting）與**搜尋**（searching）。排序是將一系列雜亂無序之資料項目，排成遞增或遞減順序。搜尋是自一系列經排序過之資料項目中，找出自己想要之項目。當然，未經排序過之資料表格亦可做搜尋，但搜尋有序資料表格則更見效率。

排序與搜尋最簡單笨拙的方式就是循序逐項比較，但此種方法效率甚低。最常用之排序方法有**泡浮排序**（bubble sort），而最常用之搜尋方法有**二分搜尋**（binary search）與**計值搜尋**（hashing search）等。本章最後將討論這兩個主題。不過，二分搜尋與計值搜尋前面表列處理一章已介紹過，故此處不再重複。這兒所介紹的乃是一般性之循序搜尋副程式。

11-10 泡浮排序副程式

泡浮排序之原理於前面表列處理一章已舉例說明過，這兒，我們再將此一副程式列出，以供讀者參考之便。副程式所運算之資料擺設與暫存器使用之情形，如圖 11-6 所示。

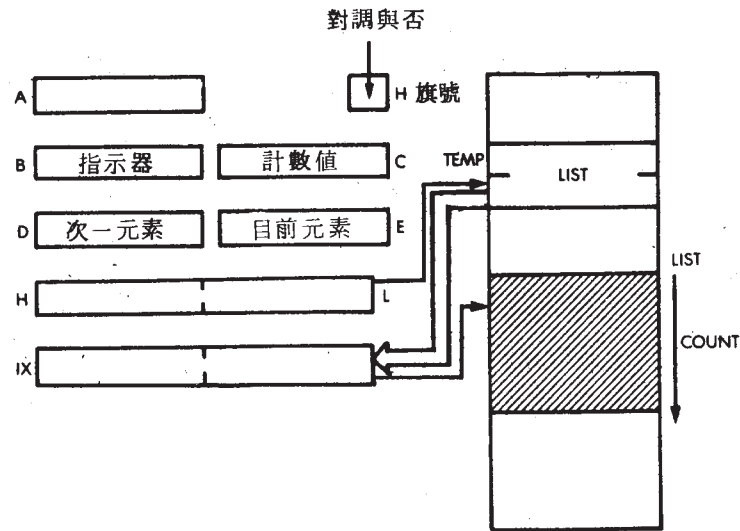


圖 16-6 泡浮排序之資料擺設及暫存器使用

例 11-19 泡浮排序副程式

；此副程式以泡浮排序法，將一資料表列之所有元素，排列成遞增順序。控制進入副程式時，H L 含表列之起始位址，C 暫存器含欲排序之元素個數。排序過程中，副程式使用下列暫存器：

- ； A 計算用之暫時儲存。
- ； B 表列資料之計數器。
- ； C 欲排序之元素個數。
- ； D 比較之第一元素。
- ； E 比較之第二元素。
- ； H 顯示互換是否發生之旗號（第 0 位元）。
- ； IX 資料表列之指示器。
- ；

```

BBSORT  LD    (TEMP),HL    ; 表列指示器換成 IX 。
AGAIN   LD    IX,(TEMP)
        RES   FLAG,H      ; 互換旗號清除為 0 。
        LD    B,C
        DEC   B
NEXT     LD    A,(IX)      ; 取入比較之第一元素。
        LD    D,A
        LD    E,(IX+1)    ; 取入比較之第二元素。
        SUB   E            ; 兩相鄰元素值相比。
        JR    NC,NOEX     ; 若前一元素 > 次一元素，
                           ; 跳至 NOEX 。
EXCHGE   LD    (IX),E      ; 否則，兩元素互調位置。
        LD    (IX+1),D    ;
        SET   FLAG,H      ; 且互調旗號為 1 。
NOEX     INC    IX
        DJNZ  NEXT        ; 若還有，則繼續比較次兩
                           ; 相鄰元素。
;
; 一巡迴完了。測試互調旗號，以決定是否繼續再次一巡迴。
;
        BIT   FLAG,H      ; 前一巡迴是否曾互調。
        JR    NZ,AGAIN    ; 若有，則再次一巡迴。
        RET              ; 否則，排序完畢。回返。

FLAG     EQU    0
TEMP     DEFS   2
END

```

11-11 表格搜尋常式

下面之副程式為一般性之表格搜尋常式。表格中含幾個元素，每一元素長 m 個位元組。搜取鍵值為八位元，同時假設表格每一元素之搜取值均位於元素之最初八位元。叫用程式輸入（傳遞）至副程式之參數值有：累加器 A 含搜取鍵值，IX 暫存器含表格起始位址，B 暫存器含表格之元素個數，而 E 暫存器則含每一元素之位元組數。圖 11-7 所示即為這些暫存器使用以及表格元素安排之情形。

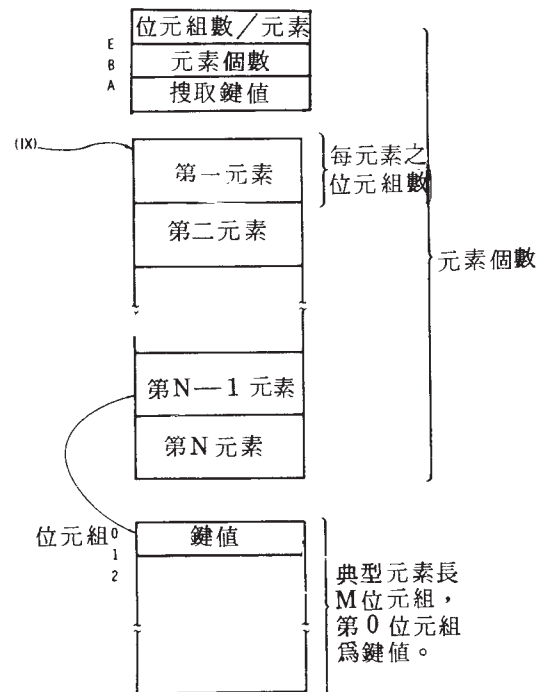


圖 11-7 表格搜尋常式——資料佈置圖

搜尋程序由表格上端往下搜尋。若找到搜取值，則第一個匹配元素之位址存於 IX 暫存器，然後控制回返。若找不到匹配元素，回返時 IX 暫存器含 -1。A，B，與 E 暫存器所含參數之安排，使得副程式能立即“再進入”，以搜取第二個匹配之元素；搜取鍵值與每元

素之位元組數值維持不變，B 所含之元素個數值會改變，以顯示表格中所餘之元素個數。再進入位址即為常式中之第二進入點。副程式之第一次進入點為 SRTAB，而“再進入”點則為 SPTAB1。

例 11-20 一般性之表格搜尋常式

；此副程式為一般性之表格搜尋常式。待搜尋表格含 n 個元素，每一元素 m 個位元組，但搜取值假設均位於第一位元組。控制進入副程式時，累加器 A 含搜取鍵值，IX 暫存器含表格起始位址，B 暫存器含表格之元素個數，而 E 暫存器含每一元素之位元組數。

；此副程式安排成可“再進入”，以搜取第二個匹配之元素。副程式之第一次進入點為 SRTAB，而“再進入”點為 SPTAB1。

```

;
SRTAB  LD      D, 0          ; 每元素位元組數存於 DE。
LOOP   CP      A, (IX)      ; 搜取鍵值與元素搜取值比。
      RET      Z            ; 找到了則回返。
SPTAB1 ADD     IX, DE        ; IX 指至次一元素。
      DJNZ     LOOP         ; 若非最後元素，則繼續。
      LD      IX, -1        ; 若搜尋失敗，IX 存 -1。
      RET                     ; 然後回返。

```

文件名稱： Z80 微電腦軟體硬體第 9、10、11 章(掃描版)

文件分類	I
文件編號	00028
文件批號	05

製作群	原稿掃描	文稿編輯
	原稿圖文分離	文稿整合
	原稿辨識	文稿校對
	文稿成品輸出	特別感謝名單

文件完成日期	初版	2007-02-10	其他
	再版		加註

文件出處	原圖書書名	Z80 微電腦軟體硬體
	原圖書作者	陳金迫
	原圖書出版者	儒林圖書有限公司
	原圖書出版日	民國 70 年 8 月

DDSC 文件 版權宣告	本文件版權屬原輸出公司、出版社、圖書公司或原著作人所有，作商業用途者請自行洽上述公司，本文件僅可在非商業上流傳或供私人收集資料用。另由於資料老舊 DDSC 不對原書內的內容負責，且除了更正原書內的錯字、漏字之外一切照原書內容所用的文字顯示。
-----------------	--------------------------------------------------------------------------------------------------------------------------

檔名格式說明：

DDSC — 文件分類 — 文件編號 — 文件批號 — 文件名.PDF

以 DDSC 為起頭，加上 1 個字母為分類代碼，再加上以 5 位數由 00001 起的編號，加上 2 位數由 01 起的編號，加上完整的文件名稱而成的。
其中分類代碼詳見下面列表。文件批號指該文件為非合訂版的，可能因書的內容過多而分批完成的，此項可有可無。

文件分類代碼說明	
代 碼	說 明
A	小說／文學類文章類
B	娛樂類
C	天文類
D	科學類
E	古文明事物類
F	自然界類
G	古怪事物類
H	動／植物類
I	電子類
J	電腦類
K	教育教學類