

第 5 章

Z80 組譯程式

先前幾章對數目系統、Z 80 微處理器硬體結構、定址法、與指令集之介紹，已為組合語言程式設計提供了大部份基礎。於開始寫作程式之前夕，在此，我們再對 Z80 組合語言，亦即 Z80 微電腦之組譯程式 (assembler)，所使用之規則作一簡介，以便讀者能立即使用。

5-1 機器語言

大家都知道，微處理器僅認得與執行機器碼。因之，任何欲讓微處理器執行之程式，必須先以機器碼（或稱目的碼）之形式，儲存於記憶器內。譬如，若我們欲求 1 至 5 之正整數的和，就必須先寫一機器碼程式，並如圖 5-1 般地儲存於記憶器內。於此一機器碼程式，第一位元組為將累加器 A 之內含清除為零之指令的機器碼。第二與第三位元組代表將 1 加至累加器內含之指令。前一位元組為指令之運算碼，後一位元組為指令之運算元（十進數 1）。程式之第四與第五位元組，代表將 2 加至累加器內含之指令，前一位元組同樣為運算碼，後一位元組為運算元（2），……等等。

5-2 組合語言

顯然，這樣的程式有幾項缺點。第一，程式由無數的 0 與 1 組成，寫來不僅費時，更易發生錯誤。第二，程式設計者必須記住（至少必須具備一張表格，以便查閱）每一指令之運算碼。由於微處理器之指令繁多，通常有數十個至百餘個，加上各運算碼與指令之功能並無任何直接意義關聯，因此，記憶此些運算碼實有困難，更不合實際。

十六進縮寫	機 器 碼	功 能 註 解
A F	10101111	累加器清除為零。
C 6	11000110	累加器內含加 1。
0 1	00000001	
C 6	11000110	累加器內含加 2。
0 2	00000010	
C 6	11000110	累加器內含加 3。
0 3	00000011	
C 6	11000110	累加器內含加 4。
0 4	00000100	
C 6	11000110	累加器內含加 5。
0 5	00000101	

圖 5-1 求 1 至 5 之正整數和的機器碼程式

“需要為發明之母”，人們因而想起，若加法指令能直接寫“ADD”，減法指令直接寫“SUB”，那多方便呀！此即為**組合語言**（assembly language）！於組合語言，計算機所能執行之每一指令皆以一**助憶符號**（mnemonic）代表。此些助憶符號即為每一指令功能名稱的英文簡寫。如上述之ADD 代表加，SUB（SUBtract）代表減，而清除為零就以CLR 代表等。

如此一來，程式設計的工作就方便且容易多了。上述之加法程式即可寫成

```

CLR  A
ADD  A, 1
ADD  A, 2
ADD  A, 3
ADD  A, 4
ADD  A, 5

```

圖 5-2 組合語言程式

此一程式，不僅寫來清徹、簡易，讀起來亦易於了解。可惜，這樣的程式計算機却看不懂，亦無法加以執行。因為，計算機僅能看得懂機器碼與執行機器碼。

5-3 組譯程式

就像中國人講話美國人聽不懂，必須經過翻譯一樣。組合語言寫成之程式在能被執行之前，必須先經過翻譯者的翻譯。如圖 5-3 所示，將組合語言程式翻譯成計算機能立即執行之機器碼程式者，即稱為**組譯程式**（assembler）。注意，組譯程式亦是一套程式，它通常是計算機系統一買來時，製造廠商即已將之寫好，並附於系統內的。有了此套程式，系統就能直接以組合語言作程式設計。否則，程式設計者就必須以機器碼作程式設計。國內全亞電子公司最近以 Z 80 微處理器設計而成之 PA-800 微電腦系統，就是一屬於能直接以組合語言作程式設計之 Z 80 微電腦系統，此系統含有一組譯程式。而其前身，亦是目前許多學校所具有之 Edu-80，則為一僅能以機器碼作

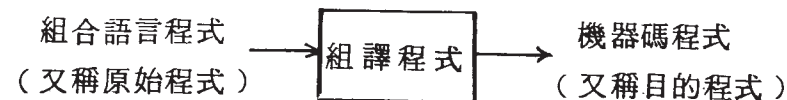


圖 5-3 組譯程式

程式設計之 Z80 系統。此一機器之用者必須親自將所寫之程式，用手（查表）翻譯成機器碼，並透過一計算器（calculator）型之鍵盤，將之存入記憶器內，計算機才能開始執行。

5-4 組譯程式特色

一般之組譯程式除了允許程式設計者能以助憶符號代表每一指令之運算碼外，尚提供了許多其它諸如符號位址，虛指令（pseudo-instruction），算術式運算元等之特色，茲將其分別介紹如下：

5-4-1 符號位址

第一章曾經說過，一般機器指令（所謂機器指令即表示成機器碼形式之計算機指令）均包括運算碼與運算元位址（或運算元）兩大部份。例如，將位址 124 F（十六進制）之記憶位置內含取入 Z80CPU 之累加器 A 的指令

```
LD    A, (124 FH)
```

即是一例。（注意，LD 乃取入指令之助憶符號。124 FH 代表十六進數 124 F）前半部 LD A 為運算碼之助憶符號，後半部 124 FH 即為運算元位址。此種位址稱為**絕對位址**（absolute address）。因為，指令上寫 124 F 就是 124 F 了，不會再變。無論將來指令翻成機器碼後儲存於記憶器之那一區域，指令之運算元“恒”來自位址 124 F 之記憶位置。

使用絕對位址之指令有一缺點，其**無法重新定位**（unrelocatable）。亦即，此種指令儲存在記憶器中之位置無法重新改變。於計算機應用，經常我們必須將程式自記憶器之某一區域，移至另一區域，以引入另一程式或另具其它目的。此時，為確保程式搬動後仍能正確動作，程式必須使用**符號位址**（symbolic address），而非絕對位址。所謂符號位址，即在程式設計時，運算元位址先以一變數名稱代表，而真正之數值位址則留待最後由組譯程式決定。

例如，假若我們欲將某兩記憶位置之內含（此兩記憶位置分別存 03 與 05）相加，結果存於第三記憶位置，那我們即可寫下面的程式。

位 址	內 含
VAR 1	03
VAR 2	05
RESULT	
	⋮

圖 5-4 符號位址

```
LD    A, (VAR1)    ; 取入第一數值。
ADD   A, (VAR2)    ; 加上第二數值。
LD    (RESULT), A  ; 結果存入記憶器。
```

```
VAR1   DEFB    03    ; VAR1 位置儲存 03。
VAR2   DEFB    05    ; VAR2 位置儲存 05。
RESULT DEFS     1     ; 保留一位置以儲存結果。
```

圖 5-5 使用符號位址之程式

組譯程式將來在翻譯此一組合語言程式時，會自動將 03 與 05 分別儲存於兩個記憶位置，然後以儲存 03 之記憶位置的數值位址取代第一指令之 VAR 1，以儲存 05 之記憶位置的數值位址取代 VAR 2。同時

，組譯程式亦會特地保留一儲存結果之記憶位置，並以此位置之位址取代第三個指令之 RESULT。此乃程式最後三個虛指令之功能。注意，程式右邊之中文字是作者額外加上的註解。此外，(VAR1)代表位址為 VAR1 之記憶位置的“內含”。括弧“()”意表“內含”。

程式控制轉移（或跳越）指令將程式控制轉移至某一記憶位置上之指令。此種指令上必須說明控制最後欲抵達之指令，所在之記憶位置的位址。此一指令位址亦可以符號表示。例如，

```

LOOP    ADD    A , 1
      :
      JP     LOOP

```

每當微處理器執行完 JP LOOP 指令後，控制就轉移至 LOOP 處開始之指令，繼續執行 ADD A, 1 指令。此時，LOOP 稱為 ADD A, 1 指令之**標題**（label），其代表 ADD A, 1 指令之第一位元組運算碼所在之記憶位置的位址。若將來程式翻譯後，該指令被存於位址 405A 之記憶位置，則組譯程式會自動將 JP LOOP 指令上之 LOOP 換成 405A。致此兩指令被翻譯成機器碼後，儲存於記憶中的情形，即如圖 5-6 所示。其中，C6 為 ADD A, 1 指令之運算碼。C3 為 JP LOOP 指令之運算碼。

就目前而言，大多數組譯程式皆屬**雙巡迴**（two passes）。此意謂組譯程式分兩階段翻譯組合語言程式。於第一巡迴，組譯程式先翻譯助憶符號，儘可能將整個指令譯成機器碼，計算指令之位元組數，並且為所有在程式內出現過之標題與符號，建立一**符號表格**（symbol table）。此一表格包括每一出現過之標題與符號，以及其所對應之數值位址。於第二巡迴，組譯程式再利用符號表格，將指令上之所有標題與符號，換成數值位址，而結束組譯程序。組譯程式必須具備雙巡迴之主要原因是，有些跳越指令會往前跳（forward reference），當組譯程式翻譯此些指令時，指令所含之標題的數值位址尚未定義。

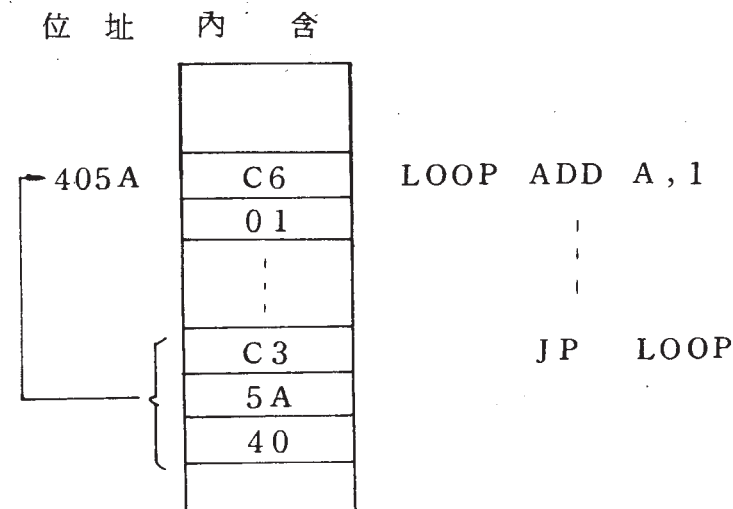


圖 5-6 符號標題

因此，組譯程式必須再回頭一次。

於組合語言程式設計時，運算元之符號位址與指令之標題名稱，是由程式設計者自定的。不過，有些名字是 Z80 系統所保留（reserved），而程式設計者不可再用作其它用途的。此些符號即為各 CPU 暫存器與各狀態旗號之名稱。它們是：

暫存器名稱：A, B, C, D, E, F, H, L

暫存器對名稱：AF, BC, DE, HL, IX, IY, SP, AF'

狀態旗號名稱：C, NC, Z, NZ, M, P, PE, PO。

5-4-2 組合語言格式

當計算機指令寫成組合語言形式時，每一指令（或應稱述句，statement，或行，line）可分成四個欄。如圖 5-7 所示，此四個欄分別為**標題**（label），**助憶符號或運算碼**，**運算元（位址）**，與**註解**。標題與註解可有可無（optional），視需要而加。有些指令則僅有運算碼欄而無運算元欄。

標題欄	助憶符號 或運算碼欄	運算元 (位址) 欄	註解欄
-----	---------------	------------	-----

圖 5-7 組合語言述句之格式

對於各欄之使用，組譯程式通常還有一些規定：

1. 各欄之間至少必須有一空格（作分界用）。
2. 標題欄之標題，長度不可超過六個文數字，且須以大寫英文字母開頭。
3. 註解可單成一行，但必須以一分號（；）開頭。爲了易於辨別起見，於本書中，每一指令後面所附之註解亦皆以分號開頭。

圖 5-8 所示即爲一組合語言程式之典型例子。試著辨別指令之每一欄。

```
; THIS IS AN EXAMPLE FOR ASSEMBLY
; LANGUAGE FORMAT。

START  XOR    A        ; CLEAR A。
        LD     B,10    ; LOAD COUNT。

LOOP   ADD    A,B      ;
        DJNZ   LOOP    ; DONE ?
        RET     ; YES。
```

圖 5-8 組合語言程式之典型例子

以上之格式是人們所寫，並輸入微電腦系統之程式格式。若系統具有列表機（printer），則自列表機印列出之組合語言程式，最左邊通常還會增加三個欄。圖 5-9 所示，即爲一例。最左邊之一欄爲每一述句之行數。左邊第二欄爲機器指令儲存在記憶器內之位址。左邊第

```
105
106      * BXASH-00-00
107      *
108      * FUNCTION: THIS SUBROUTINE CONVERTS AN 8-BIT BINARY VALUE
109      *           IN THE C REGISTER TO TWO ASCII HEXADECEMAL DIGITS.
110      *
111      * CALLING SEQUENCE: (HL)=BUFFER AREA POINTER
112      *                   (C)=8-BIT VALUE TO BE CONVERTED
113      *                   CALL BXASH
114      *                   (R)N W/CHARACTERS IN BUFFER, BUFFER +1
115      *                   AND HL INCREMENTED BY 2
116      *
117      *
118 1036 3EFO  BXASH LD  A,FOH
119 1038 A1    AND  C      MASK 1
120 1039 0F    RRCA
121 103A 0F    RRCA
122 103B 0F    RRCA
123 103C 0F    RRCA
124 103D CD4710 CALL CVERT ALIGN FOR CONVERSION
125 1040 3E0F LD  A,0FH CONVERT
126 1042 A1    AND  C      MASK 2
127 1043 CD4710 CALL CVERT GET SECOND CHARACTER
128 1046 C9    RET        CONVERT
129
130 1047 C630 CVERT ADD A,30H CONVERT TO 0-15
131 1049 FE10 CP     10    TEST FOR 0-9
```

圖 5-9 典型之 Z 80 程式列表

三欄爲各指令翻譯成之機器碼。

有關註解欄，有兩點必須再作說明：第一，組譯程式於翻譯的過程中並不理會註解，其直接將之略過，且不產生任何機器碼。第二，由於大多數電腦均屬外國人所設計，此些機器僅能接受英文字，因此，組合語言程式上之註解亦須以英文書寫。不過，爲了使讀者閱讀方便起見。本書所有程式例題之註解都寫成中文。

5-4-3 數底表示

幾乎任一組譯程式皆具備數底 (number base) 轉換之能力。這意謂指令提供之立即運算元可寫成任何最方便之基底。例如，將八位元立即值加至累加器內含的

ADD A , n

指令。指令上之立即值 n 即可表示成二進制，十進制，或十六進制。無論表示成何種基底，程式設計者必須將數目之基底告知組譯程式。於 Z 80，二進數與十六進數分別以在數目後加一 “B” 與 “H” 表示。十進數值則不加任何字尾。

例如，若欲將一百加至累加器 A 之內含，則程式設計者有三種選擇：

ADD A , 100

ADD A , 64H

或 ADD A , 01100100B

第一個指令之 “100” 為十進制寫法，第二種 “64H” 為十六進制表示（十六進數 64 正巧等於十進數 100），第三種則為二進制（ $01100100_2 = 100_{10}$ ）。

5-4-4 算式求值

大多數組譯程式皆不允許程式設計者以一算術式作運算元。但在某些較老練之組譯程式，程式設計者則能在指令之運算元位址欄上寫一算術式。譬如，若欲將位址 ADDR 之記憶位置後第 3 個記憶位置之內含取入累加器 A，其即可寫

LD A , (ADDR + 3)

組譯程式於翻譯過程中碰到此一指令時，會自動將 ADDR 之數值位址加 3，再將結果置為運算元位址。此即稱為**算式求值** (expression evaluation)。倘若 ADDR 之真正數值為 1240H，則上述指令譯

成機器碼後，即如

00111010 (= 3AH)

01000011 (= 43H)

00010010 (= 12H)

。指令長三個位元組，第一位元組 (3A) 為立即取入 A 之運算碼，第二與第三位元組分別為運算元之低次與高次位址：43 與 12。1243 即得自 1240 (ADDR) 與 3 之和。

於較能幹之組譯程式，算術式通常可由符號位址與定字 (literal) 作加、減、乘、除運算組成。譬如，

LD A , 100/4

指令即意謂將 100 除以 4 之結果 (25) 取入累加器 A。

5-4-5 虛指令

於組合語言程式，每一述句之助憶符號欄負責產生運算碼。然而，很快你會發覺，有些助憶符號並不產生機器碼，此些助憶符號用以指使組譯程式，增加程式設計之便利。如圖 5-5 之最後三個指令即是。由於它們並不產生代表真正機器指令之運算碼，因此，特稱之為**虛指令** (pseudo-instruction)。注意，虛指令並非“真正”的機器指令，它並不包括於微處理器之指令集內。其主要功用僅在指揮組譯程式，以產生數據，預留記憶空間，……等等。

Z 80 組譯程式之虛指令有：

ORG

END

EQU

DEFB

DEFW

DEFS

TXT

ORG

就像微處理器在執行程式時，必須建立一程式計數器，以追蹤程式指令一般。組譯程式在翻譯組合語言程式時，亦須設一**位置計數器**（location counter），以追蹤每一述句，並求出每一標題之數值位址。ORG 虛指令位於每一程式之開頭，用以設定組譯程式之位置計數器的起始值。例如，當組譯程式遇及“ORG 1200H”時，其位置計數器之內含會自動置定成 1200H，使得爾後每翻譯一指令，位置計數器之內含能每次增加指令之位元組數。ORG 指令亦可置於程式中任意所想之位置，以使位置計數器由另一新數值開始。

END

END 虛指令置於程式最後，以告訴組譯程式，第二巡迴可以開始，或全部組譯程式可以結束。其告知組譯程式，程式至此全部結束。

EQU

EQU 虛指令令某一標題等於另一標題或某一數值。其主要用以指定某一符號名稱之常數值或算式。例如，假若我們事先定義：

```
COUNT EQU 6
```

則在爾後的程式中，COUNT 就代表 6。組譯程式於翻譯過程中，只要遇到 COUNT，即會以 6 將之取代。

又如，某一表格資料之長度可以 EQU 虛指令定義如下：

記憶位址	組合語言程式述句
0100H	TABLE
0101H	
0102H	
0103H	
0104H	

```

0105H      :
            LENGTH EQU $-TABLE
            :
103FH      LD  IX, LENGTH

```

其中，“\$”代表組譯程式**位置計數器之現有內含值**。

於此例子，假若組譯程式處理標題 TABLE 處之述句時，位置計數器之內含為 0100H，則當其碰上 EQU 虛指令時，位址計數器之內含將為 0106H。組譯程式會自動算出 $\$ - \text{TABLE} = 0106\text{H} - 0100\text{H} = 6$ ，並於符號表格增加一資料項目 LENGTH，以及其對等值 6。稍後，當組譯程式讀到十六位元之立即取入指令 LD IX, LENGTH 時，即會自動查取符號表格，並將 LENGTH 以 6 取代。

DEFB 與 DEFW

DEFB 與 DEFW 虛指令用以定義常數與變數。DEFB 之運算元欄必須為一能表示成八位元之數值或算式，而 DEFW 之運算元欄必須為能表示成十六位元之數值或算式。程式設計者以 DEFB 與 DEFW 建立程式指令運算所需之表格資料，常數，或變數。例如，

```
VAR1 DEFB 1
```

虛指令即告訴組譯程式，將符號位址 VAR1 之記憶位置存 1。組譯程式將來在翻譯過程中碰到此一指令時，會自動預留一記憶位置，將 1 存入該記憶位置，並以該記憶位置之數值位址，取代程式中之所有

```
VAR1。又 VAR1 DEFW 0203H
```

虛指令之意即指，將位址 VAR1 之記憶位置存 02，其次一緊接位置存 03。由此可見，DEFB 與 DEFW 虛指令均用以建立資料。DEFB 用以建立八位元之資料，而 DEFW 用以建立十六位元之資料。於

FORTRAN, COBOL 等高階語言，程式以讀取 (READ) 述句獲得運算所需之資料；於組合語言，運算所需之數據則以 DEFB 與

DEFW等虛指令送給程式（存入主記憶體內）。

下面之虛指令系列即建立起一擁有十個資料項目之表格，此一表格所含之資料項目分別為 1, 2, …… , 9, 10。

```
TABLE    DEFB    1
          DEFB    2
          DEFB    3
          DEFW    0405H
          DEFB    6
          DEFW    0708H
          DEFB    1001B
          DEFB    AH
```

若組譯程式遇及 TABLE 之虛指令時，位址計數器之內含為 0200H，則此一虛指令系列之實際效用即如圖 5-10 所示。

	位 址	內 含
TABLE	0200	01
TABLE+1	0201	02
TABLE+2	0202	03
⋮	0203	04
⋮	⋮	⋮
TABLE+8	0208	09
TABLE+9	0209	0A

圖 5-10 DFFB 與 DEFW 之實際效應

DEFS

時常，我們需預留某些或某一段記憶位置，以儲存運算結果。此些記憶位置不必事先存入任何資料，因為，程式的執行過程會自動將運算結果存入此些位置。DEFS 虛指令即為用以保留此些記憶位置。

DEFS 虛指令之運算元必須為一數值——欲保留之記憶位置的個數。DEFS 虛指令之實際功能即為使位置計數器之內含增加一數值（該數值即等於 DEFS 虛指令之運算元值）。當組合程式之機器碼最後被存入記憶體時，DEFS 所騰出之記憶位置將不受影響，並保持其原有之無意義內含。下面之 DEFS 虛指令保留了自 BUFFER 起之連續 34（=22H）個記憶位置。

```
BUFFER    EQU    $
          DEFS    22H
```

留意，ORG 虛指令亦可用以預留記憶空間。譬如，下面兩虛指令之功能即同於上述兩虛指令。

```
BUFFER    EQU    $
          ORG    $+22H
```

TXT

此地欲介紹之最後一個虛指令，即 TXT。就某種意義而言，TXT 類似於 DEFB 與 DEFW。因為，它們皆產生程式欲用之資料，只不過 TXT 所產的是 ASCII 文數字資料罷了！ASCII 碼是大多數微電腦之輸入／輸出設備所使用之文數字碼。為了能由印字機，CRT 顯示器，或印字機等設備輸出，字母、數字、與特殊文字等必須寫碼成 ASCII 碼。TXT 虛指令即用以產生緊接於 TXT 後之文數字串的 ASCII 碼。TXT 欲產生之文數字串的頭尾，必須分別加一“\$”作識別分號（delimiter）。例如，

```
TXT    $CURSE YOU RED BARON$
```


虛指令，即令組譯程式將 ~~TXT~~ 後緊接之每一文數字（識別分號除外）的 ASCII 碼，依次儲存於位址 TXT 起之連續記憶位置。

以上所介紹之虛指令是一般所最常見的，此些虛指令在爾後程式設計之各章，會經常使用。雖然虛指令隨機器之組譯程式不同而異，但不管在各種不同之 Z 80 系統，或甚至其它之微電腦，其意義都是相同的。

最後，我們舉一簡單之程式例題，綜合說明以上所介紹之虛指令的應用。此一程式將位址 VAR1 與 VAR2 兩記憶位置內含之和加 6，並將結果存於位址 SUM 之記憶位置。

```

; THIS IS AN EXAMPLE PROGRAM TO
; ILLUSTRATE USAGE OF PSEUDO-INSTRUCTION
; AND COMMENTS. THE PROGRAM
; SIMPLY ADDS TWO MEMORY CONTENTS
; AND STORES THE SUM.

    ORG    0000H
    LD     A, (VAR1) ; GET 1ST NUMBER.
    ADD    A, (VAR2) ; ADD 2ND ONE.
    ADD    A, SIX
    LD     (SUM), A ; SAVE RESULT.
    RST    38H      ; EXECUTION STOP.
SIX    EQU    6      ; SIX EQUAL 6.
VAR1   DEFB    60    ; DEFINE 1ST NUMBER.
VAR2   DEFB    8     ; DEFINE 2ND NUMBER.
SUM    DEFS    1     ; RESERVE 1 LOCATION.
        END          ; PROGRAM END.

```

圖 5-11 虛指令用法之舉例

從以上例子，讀者可看出幾些常用虛指令之用法。程式最前端幾

行為對整個程式功能之註解。在組合翻譯時，組譯程式不理這些。緊接第一行為虛指令 ORG，該指令使組譯程式在開始翻譯該程式之前，將位置計數器之初值設定為 0000H，以作為參考點。該指令翻譯後不產生任何機器碼。

緊接五行為程式指令。組譯程式一一將之翻譯，並產生目的碼。最後五行則又皆為虛指令。第一虛指令 EQU 告訴組譯程式，第三程式指令中之 SIX 以 6 將之取代。緊接兩個 DEFB 虛指令，令組譯程式保留兩個記憶位置，分別將其內含設為 60 與 8，並以其所對應之數值位址，分別取代第一與第二程式指令中之相對變數。DEFS 虛指令則令組譯程式預留一記憶位置，並以其位址取代第四個程式指令中之 SUM 變數。最後虛指令 END 告訴組譯程式，程式至此全部結束。組譯程式一看到此一虛指令，即會開始第二巡迴或中止組譯過程。

當微處理器執行此一組譯後之程式時，RST 38H 告訴微處理器，所有程式指令至此結束。微處理器一執行過此一指令後，即會轉而再執行其它之程式。該指令之於微處理器，即如 END 之於組譯程式。

5-5 上 機

程式一經寫好後，緊接的就是送給計算機執行。為了簡化此一工作，系統通常會提供某些系統程式與便捷之輸入／輸出設備。原始程式之述句可由鍵盤一一輸入至微電腦，然後，諸如紙帶錢、磁帶機、或軟性磁碟等輸入／輸出設備，會將原始述句一一記錄。通常，系統均有一編輯程式 (editor)，能將鍵盤之輸入傳至記憶媒體。程式一經記錄於媒體後，系統之組譯程式緊接被取入 (load) 至記憶器，並且開始動作。組譯程式會自記憶媒體一一讀取原始述句，進行第一巡迴之組譯。若記憶媒體為紙帶或卡式錄音帶，則此時必須用手令帶子迴轉至程式開頭；若為其它記憶媒體，系統則會自動回至程式之始。

第一巡迴後，組譯程式開始進行第二巡迴，產生組合語言程式之**目的程式**(object module)，並將原始程式自列表機（或稱印字機，printer）印出。目的程式主要是成特殊存入格式(loader format)之機器碼程式。組譯程式翻譯輸出之目的程式可儲存於紙帶、磁帶、或軟性磁碟等記憶體。目的程式可由系統之**存入程式**(loader)將之取入計算機之記憶體，並令微處理器開始執行。

正如一開始我們就提過的，很少程式會一跑(run)就成功的。錯誤的程式必須再經更改（稱為偵錯），然後再重新組譯，取入，與執行，直至成功為止。圖 5-12 所示即為組合語言程式在系統中之流程圖。

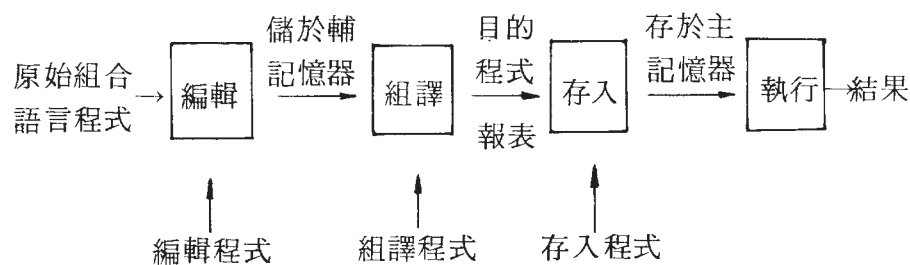


圖 5-12 組合語言程式在系統之流程

第 6 章

資料傳送

在介紹過 Z80 微處理器之定址法、指令集、與 Z80 之組譯程式後，本書自此開始介紹各種組合語言程式之設計。

於從事組合語言程式設計時，隨時記得您有多少“**資源**”(resources)可以利用是非常重要的。於 Z80 微電腦系統，一組合語言程式設計者於設計程式時可運用的所有資源，即為 Z80 微處理器內之暫存器，64K 記憶位置中之讀寫記憶位置，以及輸入/輸出界面電路中之暫存器。圖 6-0 所示即為 Z80 之程式設計者平常寫程式時所用的資源。於程式設計時，讀者應該將此一結構圖牢記在腦海裡。任何程式之設計，就是將所欲處理之資料，妥當地安排於此些設備內，並加以搬運與運算。

首先，本章討論任何計算機系統皆具有之最基本作業——兩 CPU 暫存器間，CPU 暫存器與外部記憶器間，或外部記憶器之兩區域間之資料傳送。此些資料搬運涉及將資料自某一位置傳遞至另一位置（實際是將來源位置之內含抄至目的位置），或兩位置內含之互換。有些搬運則涉及堆疊器資料之存取。於 Z80，每次搬運資料之長度可為八位元或十六位元。區段傳送指令最高則可傳送 64K (1K=1024) 個位元組之資料。

Z80 之資料傳送可分四方面探討：八位元傳送，十六位元傳送，區段傳送，與資料互換。特別注意，除了資料互換外，將資料自某一位置傳送至另一位置之資料傳送作業，均不改變資料來源位置之原有內含。

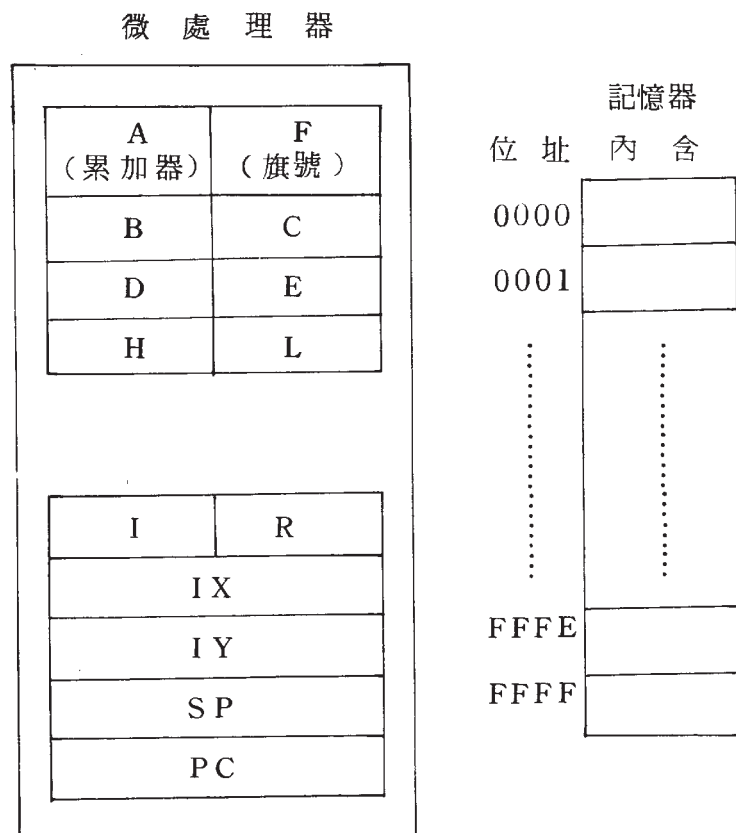


圖 6-0 Z80 程式設計者可用之資源

6-1 八位元傳送

八位元資料傳送主要以八位元之取入 (LD) 指令達成。此一指令使用多種定址法，使資料能自某一 CPU 暫存器搬至記憶體 (或另一 CPU 暫存器)，或自記憶體搬至某一 CPU 暫存器。將資料搬入或搬出累加器 A 乃是其中最重要者，因為，A 暫存器是微處理器所有算術、邏輯、與移位運算涉及之暫存器。下面，我們由最簡單之傳送作業開始，逐一介紹八位元傳送之各種方法。

暫存器傳送，立即與擴展定址

任一一般用途之八位元 CPU 暫存器，可以立即定址之取入指令，定義其內含值。例如，若欲將累加器 A 之內含值設定為 5，則

```
LD  A, 05H
```

可達成任務。又如，若欲將十進數 10 存入 B 暫存器，則

```
LD  B, 0AH
```

亦可使用。該指令執行完後，B 暫存器之內含變為 00001010_2 (等於十進數 10)。

將任一暫存器內含清除為零最直接之方法，即為借用立即定址之取入指令。譬如，

```
LD  C, 00H
```

執行完後，C 暫存器之內含即變為零。將累加器 A 之內含清除則有較多方法。其中，

```
LD  A, 00H
```

```
XOR A
```

```
SUB A, A
```

等任一者皆可達成目的。

於 Z80，任一 CPU 暫存器之內含皆可以暫存器傳輸指令

```
LD  r, r'
```

輕易傳遞至另一 CPU 暫存器。例如，

```
LD  B, A
```

指令執行完後，A 暫存器之內含即被抄入 B 暫存器。此時，A 與 B 暫存器之內含皆變成果加器 A 之原有內含值，B 暫存器之原有內含已遭破壞。

由此可見，欲將任意兩暫存器之內含互換，必須假藉第三個暫存器 (或記憶位置)，將其中某一暫存器之內含先存起。例如，若欲將暫存器 A 與 B 之內含互換，則可將 A 暫存器之內含先寄存於 C 暫存器

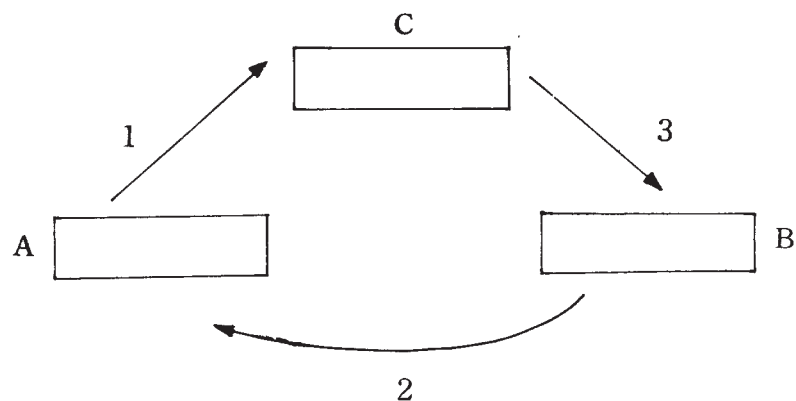


圖 6-1 A與B兩暫存器之內含互換

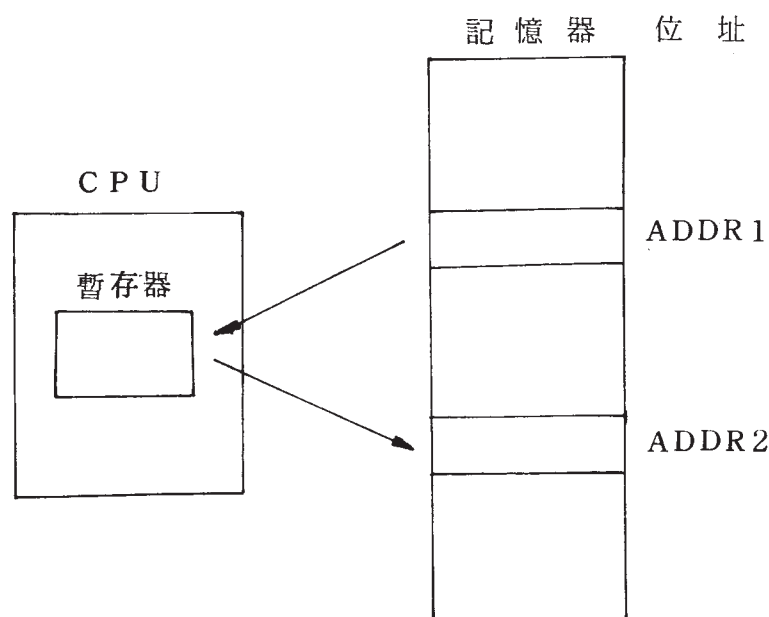


圖 6-2 記憶位置至記憶位置之傳輸

，之後，B暫存器之內含取入A暫存器，最後，A暫存器之內含再由C暫存器取入B暫存器。

```
LD    C, A
LD    A, B
LD    B, C
```

注意，任一記憶位置之內含無法直接傳送至另一記憶位置，而必須假藉某一CPU暫存器作轉運站。譬如，若位址ADDR1之記憶位置的內含欲傳送至位址ADDR2之記憶位置，則可假藉累加器A作轉運站，使用下列擴展定址之指令。

```
LD    A, (ADDR1)
LD    (ADDR2), A
```

注意，ADDR1（與ADDR2）為符號位址，其實際上可能代表1025H，或34FFH，……等等。此外，第二指令執行完後，位址ADDR2之記憶位置的原有內含，已遭破壞。

因此，若兩記憶位置之內含欲對調，則必須使用兩CPU暫存器，一作暫存寄存，一作中間轉運。但是，擴展定址僅能用於累加器A與記憶器間之傳輸，因此，必須利用其它定址方式之取入指令。

將八位元資料自記憶器取入某一CPU暫存器，有四種定址法可使用（反方向之傳送亦然）：

1. 使用HL之暫存器間接定址。
2. 使用IX或IY之索引定址。
3. 擴展定址（僅能取入累加器A）。
4. 使用BC或DE之暫存器間接定址（僅能取入累加器A）。

由於第3與第4種僅能用於取入累加器A，因此，下面我們先討論前面兩種方法。

習題6—1：將0, 1, 2, 3, 4分別存入A, B, C, D, 與E暫存器，然後再將其次序倒反。

HL 暫存器間接定址

下面的程式將位址VAR1, VAR2, VAR3, 與VAR4 等四個記憶位置之內含，分別取入Z80 CPU之A, B, C, 與D四個暫存器。一開始時，十六位元之取入指令將四位元組之起始位址取入HL 暫存器對，將HL 設定為資料指示器。每當有一資料取入，HL 暫存器之值就加一，指至次一項資料。

```

LD      HL, START    ; 指示器指至起點。
LD      A, (HL)      ; 取入第一項資料。
INC     HL           ; 指示器內含加一。
LD      B, (HL)      ; 取入第二項資料。
INC     HL           ; 指至次一項資料。
LD      C, (HL)      ; 取入第三項資料。
INC     HL           ; 指至START+3。
LD      D, (HL)      ; 取入第四項資料。
:               ; 此必須加一停止指令。
START   EQU      $    ; START=VAR1。
VAR1    DEFS     1     ; 此些記憶位置之
VAR2    DEFS     1     ; 內含於程式執行
VAR3    DEFS     1     ; 過程中界定。
VAR4    DEFS     1

```

注意，若四項資料位於**連續緊接**之記憶位置，則上述程式完全動作正常。但若四項資料不位於連續記憶位置，則上述程式便無法達成任務。此時，每取入一項資料後，HL 暫存器就必須再重新存入新的值。下面的程式將A, B, C, 與D 暫存器之內含，分別存出至位址STOR1, STOR2, STOR3, 與STOR4 之記憶位置，此時，這四個記憶位置為任意位置，並不連續。

```

LD      HL, STOR1    ; 位址STOR1 取入HL。
LD      (HL), A      ; 存出第一項資料。
LD      HL, STOR2    ; 位址STOR2取入HL。
LD      (HL), B      ; 存出第二項資料。
LD      HL, STOR3
LD      (HL), C
LD      HL, STOR4
LD      (HL), D
:
STOR1   DEFB      0    ; 最初，DEFB
:               ; 將此些位置之
STOR2   DEFB      0    ; 內含置定為0。
:
STOR3   DEFB      0
:
STOR4   DEFB      0

```

索引定址

於Z80, IX 與IY 兩索引暫存器類似於HL 暫存器——皆作為資料指示器用。不過，有點小差別。以索引暫存器作指示器時，運算元之真正位址等於索引暫存器之內含，與指令所提供之位移值兩者之和。每一索引定址指令必須提供一八位元之位移（2 補數形式），這使指令能存取-256 位元組“區段”（block）內之任一位元組——以索引暫存器所指之記憶位置為基點，向前（位址大之方向）127 個位元組，向後（位址小之方向）128個位元組，如圖6-3所示。

假設我們欲將A, B, C, 與D 四暫存器之內含，分別存至位址BASE-4, BASE, BASE+4, 與BASE+8 等記憶位置，則下列索引定址之指令可達成任務。


```
LD    IX, BASE
LD    (IX-4), A
LD    (IX+0), B
LD    (IX+4), C
LD    (IX+8), D
```

四個索引定址指令之第三位元組（前兩位元組為運算碼）上的位移值，分別是-4, 0, 4, 與8。此時，資料之存取似乎就此前面的使用HL暫存器對之例子更有效率。讓我們將此兩程式作一比較。前一程式使用4個三位元組指令（LD HL, STORX），與4個單位元組指令（LD (HL), D），總計佔16個位元組之記憶空間與17微秒之執行時間。索引定址之程式則使用5個三位元組指令，總計佔用15個位元組之記憶空間與22.5微秒之執行時間。如此看來似乎前者稍較經濟。但這僅是其中一例。於大多數情況，索引定址顯示了其優點。

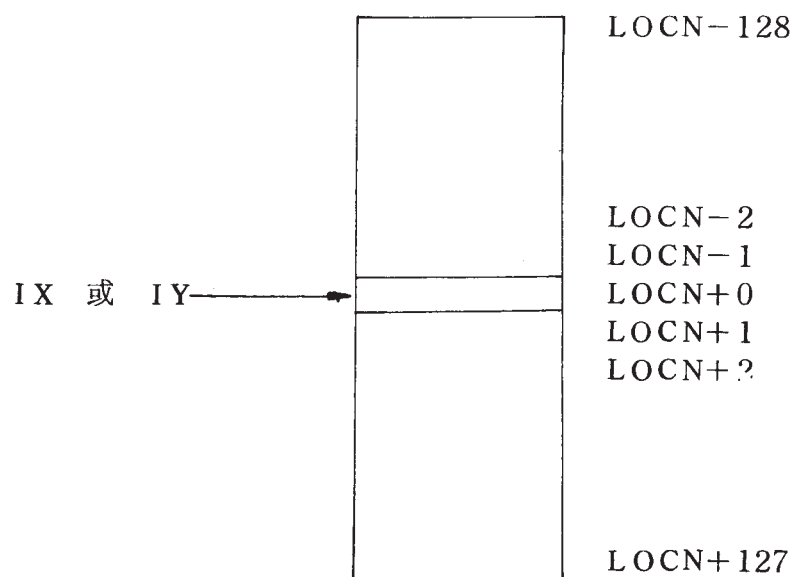


圖 6-3 以索引暫存器作區段存取

IY 索引暫存器之功能與 IX 完全相同，兩者可互相替用。索引定址最見長之處就是**區段搬運或傳送**（block move）。下面之程式即將位址BLK1至BLK1+2之記憶區段的資料，整體移至位址BLK2至BLK2+2之記憶區段。IX 充當來源區段之指示器，IY 則作為目的區段之指示器。程式的作法是由位址最大之位置，反過頭搬運，這可容許來源區段與目的區段彼此重疊而不致發生錯誤。

```
LD    IX, BLK1+2    ; 設定來源區指示區。
LD    IY, BLK2+2    ; 設定目的區指示器。
NEXT1 LD    B, (IX+0) ; 資料取入B暫存器。
LD    (IY+0), B      ; 資料存至目的位置。
DEC    IX            ; 指至次一位元組。
DEC    IY            ; 指至次一目的位置。
NEXT2 LD    B, (IX+0)
LD    (IY+0), B
DEC    IX
DEC    IY
NEXT3 LD    B, (IX+0)
LD    (IY+0), B
```

無論就寫碼時間或程式所佔之記憶空間而言，這樣的寫法都是很不經濟的。假若資料有100項，1000項，……。那寫到什麼時候呢？程式要佔多少空間呢？尤其是，於上述例子中，每一項資料之搬運指令與指示器內含更新指令，完全相同。因此，更無必要逐次重複。於此情況，最佳的辦法就是製作一**迴路**（loop），使同一組指令能反復被執行，有幾項資料就反復執行幾次。不過，由於迴路必須有結束的時候（資料搬完就結束）。因此，迴路內必須增加一**迴路計數器**（loop counter），以計數指令被重複執行的次數，以及決定迴路何時終止。

下面之程式即為以C暫存器作迴路計數器，搬運100項資料之區

段傳送程式。迴路（標題 LOOP 以下）開始前，IX 與 IY 分別先指至資料來源區段與目的區段之起點，同時，資料項目之總個數（即迴路應被重複執行之次數），100，亦存入迴路計數器，C 暫存器。程式被執行時，索引暫存器 IX 與 IY 之內含將每次加一，而迴路計數器 C 之內含每次減一（DEC C）。然後，條件跳越指令 JP

```

LD      IX, BLK1      ; 指至來源區段之始。
LD      IY, BLK2      ; 指至目的區段之始。
LD      C, 100         ; 設定迴路計數值。
LOOP    LD      B, (IX+0) ; 取入資料。
        LD      (IY+0), B ; 存出資料。
        INC     IX      ; 移動指示器。
        INC     IY
        DEC     C       ; 迴路計數器值減一。
        JP      NZ, LOOP ; 若非零，則再迴路。
DONE    ;               ; 完了。

```

IX	IY	C	B	
BLK1	BLK2	100		起 始
BLK1+1	BLK2+1	99	第1項資料	第1次迴路後
BLK1+2	BLK2+2	98	2	2
BLK1+3	BLK2+3	97	3	3
⋮	⋮	⋮	⋮	⋮
BLK1+97	BLK2+97	2	98	98
BLK1+98	BLK2+98	1	99	99
BLK1+99	BLK2+99	0	100	100

圖 6-4 索引之例子

NZ, LOOP 測試 C 暫存器內含減一的結果是否為零，若是，則迴路結束；否則，控制繼續傳回 LOOP 位置之指令，繼續重複執行迴路。結果，迴路內之指令將被執行 100 次，而程式最前面三個指令僅被執行一次。圖 6-4 所示即為迴路執行時，各有關暫存器內含之變化情形。

BC 或 DE 暫存器間接定址

來回傳遞於累加器 A 與記憶器間之資料，除可以 HL 外，尚可以 BC 或 DE 作暫存器間接定址。如此定址之四指令為：LD A, (BC); LD A, (DE); LD (BC), A; LD (DE), A。同樣，此種定址在存取處於連續記憶位置之方塊或表格資料時，甚有效率。前面已舉了一些以索引暫存器作區段搬運之例子。下面的程式所為大體相同。暫存器對 BC 指至來源區段，DE 指至目的區段。暫存器 L 則為迴路計數器

```

LD      BC, BLK1      ; BC 指至來源區段。
LD      DE, BLK2      ; DE 指至目的區段。
LD      L, 100         ; 設定迴路計數值。
AGAIN   LD      A, (BC) ; 取入一項資料。
        LD      (DE), A ; 存出資料。
        INC     BC      ; 更新指示器內含。
        INC     DE
        DEC     L       ; 迴路計數值減一。
        JP      NZ, AGAIN ; 若非零，則再迴路。
DONE    ;               ; 完了。

```

6-2 十六位元傳送

以上所討論的資料傳輸均為八位元傳輸。換言之，每一指令執行僅引起八位元資料的傳遞。Z80 具有數個一次能搬運十六個位元（

即兩個位元組)之指令。此些指令涉及將十六位元資料來回傳遞於兩十六位元暫存器之間，或某一十六位元暫存器與記憶器之間。其共可分為四類：

1. 立即取入 BC, DE, HL, SP, IX, 或 IY。
2. 資料自記憶器傳至 BC, DE, HL, SP, IX 或 IY; 或反之。(僅能使用擴展定址)
3. HL, IX, 或 IY 之資料傳至 SP。
4. BC, DE, HL, AF, IX, 或 IY 之內含推入堆疊器，或自堆疊器拉取。

當然，取入 SP, IX, 或 IY 者，經常為記憶位址。十六位元等於 Z80 之全位址，可選取 64K 個記憶位置。因之，這一組指令特別設計以操縱與記憶位址有關之資料。當然，必要時，其亦可用以存取一般非位址之運算元，諸如十六位元之雙倍長運算元或 ASCII 文數字資料等。

十六位元之立即取入

部份十六位元之取入指令在前面已經用過，此些指令將欲處理之資料區段的起始位址，取入相關之暫存器對 BC, DE, HL 或十六位元之索引暫存器 IX 與 IY。下面的指令系列具有相同之功能。此些指令分別將五個資料區段之起始位址 DATA 1, DATA 2, ………, DATA 5, 取入 BC, DE, HL, IX, 與 IY 暫存器。

```
LD      BC, DATA 1    ; 取入各資料區段。
LD      DE, DATA 2    ; 之起始位址。
LD      HL, DATA 3
LD      IX, DATA 4
LD      IY, DATA 5
⋮
```

```
DATA 1   DEFS    100    ; 第 1 資料區段含 100 位元組。
DATA 2   DEFS     50    ; 第 2 資料區段含 50 位元組。
DATA 3   DEFS     20    ; 第 3 資料區段含 20 位元組。
DATA 4   DEFS     60    ; 第 4 資料區段含 60 位元組。
DATA 5   DEFS    100    ; 第 5 資料區段含 100 位元組。
```

然而，堆疊指示器，SP，恒指至分配作為堆疊器之記憶區域，而非某一預定資料區段。因此，系統一開始時，其內含必須指至堆疊底端。由於堆疊指示器恒指至前一用過位置，同時每次資料存入前其內含值先減一，因此，SP 之起始值應比堆疊器之第一位置的位址值大一，例如，若堆疊器佔據位址 3FFFH 至 3F00H 之 256 個記憶位置，則 SP 之起始值應如下設定：

```
0100          LD      SP, 4000H    ; 設定堆疊起點。
              或
              LD      SP, TOPS
              ⋮
3F00          DEFS    100H          ; 保留 256 個位置作
4000  TOPS  EQU    $                ; 堆疊器。
```

緊接之推入作業（由於此時堆疊器是空的，因此不能拉取），資料在被推入之前，SP 之值將先減一。致第一項資料將存於位址 3FFF 之記憶位置，第二項於 3FFE 之位置，……等等。

來回記憶器之十六位元傳輸

這一組的指令能將某兩連續記憶位置之內含取入 BC, DE, HL, IX, IY, 或 SP 等任一十六位元暫存器，或反之將此些任一十六位元暫存器之內含存出至記憶器。舉個例子說，若 BC, DE, 與 HL 暫存器現欲存入三個記憶區段之起始位址，而其原有之內含值必須保留以便爾後使用，則下列指令系列可用以將其原有內含存起：

```

LD      (SAVB), BC    ; 存起 BC 內含。
LD      (SAVD), DE    ; 存起 DE 內含。
LD      (SAVH), HL    ; 存起 HL 內含。
:
SAVB    DEFS    2      ; BC 內含存於此。
SAVD    DEFS    2      ; DE 內含存於此。
SAVH    DEFS    2      ; HL 內含存於此。

```

留意，每一暫存器對須保留兩個記憶位置。之後，當此些暫存器對之內含欲用到時，下列之指令系列可用以將之自記憶器中取回。

```

LD      BC, (SAVB)    ; 取回 BC 值。
LD      DE, (SAVD)    ; 取回 DE 值。
LD      HL, (SAVH)    ; 取回 HL 值。

```

注意，記憶器與暫存器對間之十六位元傳送，僅能使用擴展定址。

至此，讀者應可分清擴展立即定址與擴展定址之不同記法與不同意義才對。下例中，擴展立即定址之 LD HL, SAVL 指令將十六位元**記憶位址**“SAVL”（實際上是**數值**位址，而非符號位址，在本例中亦即 1000H），存入 HL 暫存器。而擴展定址之 LD HL, (SAVL) 指令，則將位址 SAVL 之**記憶位置的內含**取入 HL 暫存器。

```

LD      HL, SAVL    ; HL 內含變為 1000H。
:
LD      HL, (SAVL)  ; HL 內含變為 1234H。
:

```

```
1000 SAVL DEFW 1234H
```

至 SP 之十六位元傳送

於 Z 80，HL, IX, 或 IY 任一暫存器之內含可傳至 SP。此些指令為

```

LD      SP, HL      ; SP ← HL。
:
LD      SP, IX      ; SP ← IX。
:
LD      SP, IY      ; SP ← IY。

```

堆疊作業

此一小節之標題可能會造成一點誤解，因為，於 Z80，**所有的**堆疊作業均為一次十六位元（或兩位元組）之傳輸。八位元之資料並無法如其它微電腦般地推入（或拉出）堆疊器。但是，這並非嚴重之缺陷，因為，八位元暫存器之內含同樣可推入堆疊器（並從堆疊器中取回），只不過必須做一點虛功（overhead）罷了。

於 Z 80，BC, DE, HL, AF 等暫存器對與 IX 或 IY 暫存器之內含，均可推入堆疊器中存起，並於稍後自堆疊器取回。於推入時，暫存器對之高位元元組先推入，然後再低位元元組。每一位元組推入前，堆疊指示器之值均先減一，以指至次一緊接可用之位置。Z80 之所有推入指令如下：

```

LD      SP, 1000H    ; SP 起始值令為 1000H。
:
PUSH    AF    ; A 存於 0FFFH, F 於 0FFE H。
PUSH    BC    ; B 存於 0FFDH, C 於 0FFCH。
PUSH    DE    ; D 存於 0FFBH, E 於 0FFAH。
PUSH    HL    ; H 存於 0FF9H, L 於 0FF8H。
PUSH    IX    ; (IX)H → (0FF7H), (IX)L → (0FF6H)。
PUSH    IY    ; (IY)H → (0FF5H), (IY)L → (0FF4H)。

```

留意，F（旗號）暫存器同樣被視為一八位元暫存器。

資料自堆疊器拉取時，程序恰巧相反。低位元元組先拉取，並存入暫存器對之低位八位元，**然後**，SP 之值減一，高位元元組被拉取

，並存入暫存器對（或 I X , I Y）之高次八位元。

堆疊器記憶主要有下列幾項用途：

1. 作 CPU 暫存器之暫時儲存。
2. 作為 CPU 暫存器間資料傳送的一種方式。
3. 用於插斷處理。
4. 用於副程式。

插斷處理與副程式之堆疊作業將留待稍後再作討論。其它的兩種用途則較明顯。任何時刻，任一暫存器對，I X , 或 I Y 之內含可以 PUSH 指令，將之推入堆疊器中存起，並於稍後要用時，以 POP 指令將之取回。此即 CPU 暫存器之暫時儲存。自堆疊器拉取之資料並未規定一定要存回原暫存器對，因此，正如下面例子所示的，堆疊器亦可用於暫存器與暫存器間之資訊傳送。

```

PUSH    BC
PUSH    IY
PUSH    DE
PUSH    IX
POP      DE
POP      IX
POP      BC
POP      IY

```

上述之指令系列將 BC 與 IY，以及 DE 與 IX 之內含互換。

雖然必須十分小心，但堆疊指示器亦可用以幫助資料串之處理。舉例而言，設位址 1700H 至 177FH 之連續記憶位置，儲存了一系列列文數字之 ASCII 碼，第一文數字在 1700H 之位置，最後文數字在 177F 之位置（利用加一型之指令，資料可輕易存成此種型式）。只要處理期間堆疊器不作任何其他用途，則下列指令即可用以將此文數字一一取出，加以處理。

```

LD      (SAVP), SP
LD      SP, 1700H
POP      BC
      ⋮
      處理
LD      SP, (SAVP)

```

注意，於上述處理之期間，可罩蓋或不可罩蓋插斷皆不能發生，同時，程式亦不能使用任何用及堆疊器之常式（routine）。

上述之處理雖能正確動作，但並非好辦法，尤其插斷會造成不可收拾之結局。Z80 所特有區段傳輸指令能更完美地達成任務。

6-3 區段傳送

Z80 之區段傳送指令，能自動或半自動地將最多 64K 個位元組之資料，由記憶器之某一區段移至另一區段。雖然如此，但除非無可避免，否則，儘量勿將一大批之資料，自記憶器之某一區域移至另一區域。避免作大批資料移動之方式有幾。資料應直接輸入或輸出至能直接處理之緩衝器，建立表格避免資料重新成形，或使用諸如連鎖表列（linked list）而非連續表格之資料結構等皆是。避免作大批資料傳送之主要著眼乃在必須花費大量之時間。倘若移動每個位元組需時 10 微秒，則移動 1000 個位元組即需時 10 毫秒或 1/100 秒。即令 Z80 移動每位元組之時間僅為三分之一，但比起其它程式運算，區段搬運仍需耗費大量時間。

LDI

記住上述原則，現在我們看 Z80 之區段傳送指令如何應用，首先，先看 LDI。LDI 指令需以 HL 暫存器對指至來源資料區段，DE 暫存器對指至目的資料區段，且 BC 暫存器對含位元組計數。舉例而

Z 80 微電腦軟體硬體

言，若欲將 100H 個位元組之資料，自位址 1000H 位置起之資料區段，移至位址 2000H 位置起之資料區段，則下列指令系列可用。

```

LD    HL, 1000H    ; 建立來源指示器。
LD    DE, 2000H    ; 建立目的區段指示器。
LD    BC, 100H     ; 建立位元組計數器。
LOOP  LDI          ; 傳輸一個位元組。
      JP  PE, LOOP  ; 未完時再繼續。
DONE  :
```

前三個指令為佈建 (initialization) 指令，先安排好各指示器與計數器。佈建妥後，LDI 指令即可開始使用。LDI 指令每被執行一次，兩指示器 (DE 與 HL) 之內含即自動加一，位元組計數器 (BC) 之值自動減一。JP PE, LOOP 指令然後測試 P/V 旗號之值 (若位元組計數不為零，則該旗號值置定為 1)。若 P/V 旗號為 1，亦即位元組計數值尚未達零，則程式控制再跳回 LOOP 處，繼續執行 LDI 指令，傳輸次一位元組。

比起 LDIR，LDI 算是“半自動”的。因為，其每次僅能引起一個位元組之傳送，而 LDIR 每次能引起多個 (最多 64K) 位元組之傳送。因此，LDIR 屬於“全自動”。不過，LDI 亦有其優點。它允許每兩項資料傳送之間能插入某些其它型式之處理。譬如，若遇到資料項目為零時，搬運作業即欲終止，則此時就非使用 LDI 指令不可。下面之指令系列所為同前，不過，當資料為零時，搬運作業立即停止。來源位元組在搬動之前先被測試，若其值為零，則搬運作業立即終止。否則，搬運作業繼續。OR A, A 指令用以測試欲傳送之位元組是否為零。若位元組之所有位元皆 0，則 Z 旗號置定為 1；否則，Z 旗號之值為 0。

```

MOVE  LD    HL, 1000H    ; 設定來源區指示器。
      LD    DE, 2000H    ; 設定目的區指示器。
      LD    BC, 100H     ; 設定位元組計數器。
```

```

NEXT  LDI          ; 傳送位元組。
      JP    PO, DONE    ; 若計數值為 0，則結束。
      LD    A, (HL)     ; 取入欲傳送之位元組。
      OR    A           ; 看其是否為零。
      JP    NZ, NEXT    ; 若非零，則傳送。
DONE  :            ; 若零，則結束。
```

LDI 指令之另一優點為，其可用以搬運非相鄰之資料。假設有一長 256 個位元組之表格資料，每逢第四項資料欲如圖 6-5 所示地搬至另一記憶區域，則總共必須做 $256/4 = 64$ 次搬運。下列之程式可用以達成目的。

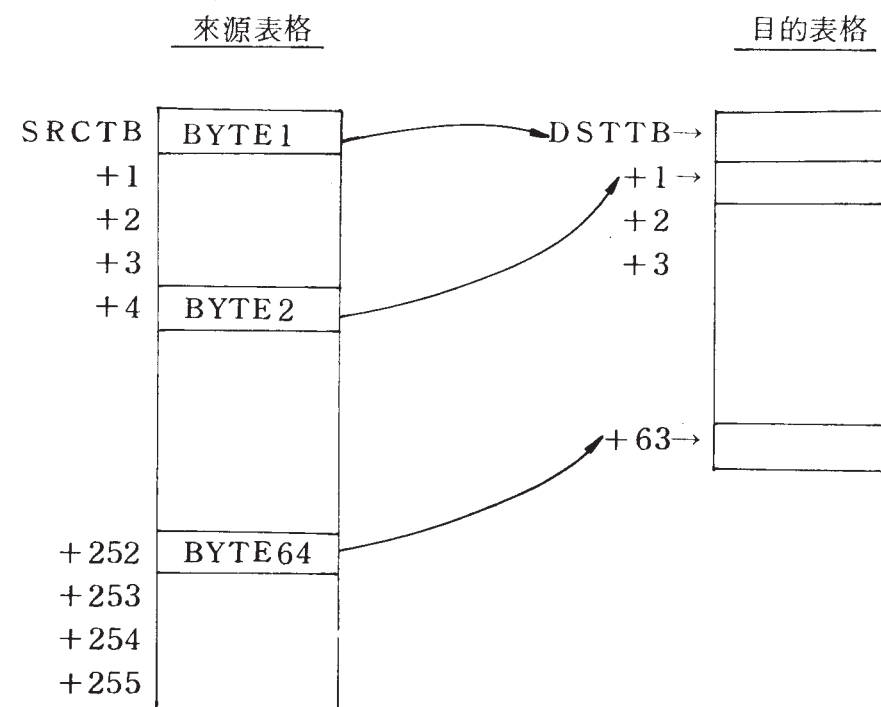


圖 6-5 以 LDI 指令搬運非鄰接資料

```

        LD      HL , SRCTB    ; 設定來源區指示器。
        LD      DE , DSTTB    ; 設定目的區指示器。
        LD      100H/4        ; 設定位元組計數器。
NEXT    LDI                     ; 傳輸位元組。
        INC     HL            ; 指至次一位元組。
        INC     HL
        INC     HL
        JP      PE , NEXT     ; 未完則繼續。
DONE    :
```

上述之指令系列有幾個絕妙之處。算術式 $100H/4$ 在許多組譯程式仍能正常動作，此些組譯程式於**組譯期間** (assembly-time) 會自動計算出算術式之值。此外，LDI 搬完第 I 位元組後，HL 暫存器指至第 $I + 1$ 位元組。緊接三個 INC HL 指令令 HL 指至第 $I + 4$ 位元組。

LDIR

若每兩位元組傳送之間不作任何處理，則區段傳送可以 LDIR 進行。LDIR 指令使用前之佈置，如同 LDI。若有 N 個位元組欲傳送，LDIR 指令將執行 N 次。每一次傳送 LDIR 需費時 5.25 微秒 (LDI 需 4 微秒)。但最後一次傳送 ($BC = 0$) 則僅需 4 微秒。

```

        LD      HL , STRTS    ; 指至來源區起點。
        LD      DE , STRTD    ; 指至目的區起點。
        LD      BC , 64       ; 設定位元組數。
        LDIR
DONE    :
```

上述指令系列將位址 STRTS 記憶位置起之 64 位元組的資料，集體搬運至位址 STRTD 起之連續記憶位置。

LDD與LDDR

於 LDI 與 LDIR 指令，資料順向傳輸。亦即，位址較小之記憶位置的資料先搬動，位址較大者後搬動。這有一個缺點，若資料來源區與目的區有重疊之處，則重疊處之資料在未搬出之前，會先遭破壞。此一障礙可以 LDD 與 LDDR 指令加以克服。LDI 及 LDIR，與 LDD 及 LDDR 之唯一區別是，後兩指令之資料傳送是反向進行——由位址最大之位置上的資料開始先搬，再逐次位址較小之資料。因此，一開始時，HL 與 DE 必須分別指至來源區段與目的區段之**尾端**，然後，每搬動一位元組時，其內含分別遞減。下面之指令系列即是以 LDDR 指令作區段傳輸的情形。

```

        LD      HL , ENDOS
        LD      DE , ENDOD
        LD      BC , 64
        LDDR
DONE    :
        STAB    DEFS    64      ; 來源資料區段。
        ENDOS   EQU     $ - 1
        DTAB    DEFS    64      ; 目的資料區段。
        ENDOD   EQU     $ - 1
```

6-4 資料互換

Z80 之互換指令組有六個指令。其中兩個將目前之 CPU 暫存器組與其互補 (') 組互換。三個將 HL, IX, IY 暫存器之內含與堆疊頂端之內含互換。最後個指令則將 DE 與 HL 之內含互換。

每當 CPU 一起動，八個非補數之暫存器組即變為動作組。另外八個補數組 (A', F', B', C', D', E', H', 與 L') 則為預備組。動作組

與預備組之角色可透過EX AF, AF' 與EXX兩個指令互換。EX AF, AF' 指令將A及F內含，分別與A'及F'之內含互換。下面的指令系列即說明如何暫時將A與F之內含存起，然後於處理後再取回：

```
EX    AF, AF'      ; AF內含存起。
```

```
    :
```

```
    (處理)
```

```
    :
```

```
EX    AF, AF'      ; AF內含取回。
```

同樣地，EXX將BC, DE, 及HL之內含，分別與BC', DE', 及HL'之內含互換。下面之指令系列說明了如何將BC, DE, 與HL之內含暫時存起，俟處理後再取回：

```
EXX                      ; 存起BC, DE, 與HL之內含。
```

```
LD    BC, NEW1
```

```
LD    DE, NEW2
```

```
LD    HL, NEW3
```

```
    :
```

```
    (處理)
```

```
    :
```

```
EXX                      ; 取回BC, DE, 與HL之內含。
```

EX AF, AF' 與EXX最常用於插斷處理時，將CPU之原有“環境”存起。此兩個指令不作其它用途之主要理由是，若以補數暫存器組作臨時資料儲存，則此時若再發生插斷，現有之CPU環境就不知再往那兒擺。若再使用交換指令，則原先寄存之臨時資料將遭破壞。因此，補數組之暫存器最好專門留作插斷處理用，而臨時之資料則以堆疊器或記憶器作寄存。

EX DE, HL指令將DE與HL兩暫存器對之內含互換。由於DE之算術運算極為有限，因此，此一指令在將DE之資料移至HL

上甚為有用。舉例而言，若DE暫存器對之內含欲加倍。則下列指令先將DE之內含移至HL，令HL自加一次，然後再將HL之內含移回DE，以達成目的。

```
EX    DE, HL      ; (DE) → (HL)
```

```
ADD   HL, HL      ; (HL) + (HL) → (HL)
```

```
EX    DE, HL      ; (HL) → (DE)
```

最後三個互換指令則將堆疊頂端連續兩位元組之資料，與HL, IX, 或IY之內含互換。互換的結果並不影響SP之值。顯然，製造廠商提供這個指令是有特殊用意的。此點留作習題，讓讀者自行去發覺。（其中之一就是使回返位址能更改，令控制不必回至緊接叫用指令後的指令。）

6-5 摘要

以上，我們已一一詳盡地介紹了Z80微處理器之資料傳送指令，並說明如何以之構成一般之資訊傳輸程式。於Z80，單一指令可達成八位元或十六位元之資訊傳送。其分別由八位元之LD指令與十六位元之LD指令達成。除此之外，堆疊作業指令（PUSH與POP）與互換指令亦具有特殊之資訊傳輸功能。較長之表格資料的傳輸則須將此些指令置於“迴路”內，或使用Z80所特有之區段傳輸指令達成。茲將以Z80指令作區段傳輸之方法摘要於下。為了統一起見，於下述例子中，我們假設資料皆以反向傳輸，亦即由區段之最高點（位址最大之位置），逐次往區段之最低點（位址最小之記憶位置）作業。同時，FROM與TO均分別代表資料來源區與目的區之最高位置的位址，COUNT為欲傳輸資料之位元組數。

以索引技巧作區段傳輸

區段傳輸作業最基本之方法就是使用索引技巧。索引技巧主要以選取表格內之連續記憶位置。Z80具有兩個長十六位元之索引暫

存器 I X 與 I Y ，其可分別作為資料之來源區段與目的區段的指示器。由於指令僅能提供一八位元 2 補數之位移，因此，使用索引技巧作區段傳輸時，資料區段之長度必須小於 256 個位元組。

下面之程式即將位址 FROM 以下之來源區段的資料，搬至位址 TO 以下之目的區段。區段所含之位元組數，亦即區段之長度，為 COUNT。此一數值於程式執行前須先存好。如圖 6-6 所示，I X 與 I Y 分別令為來源區段與目的區段之指示器，而 C 暫存器為迴路計數器。

例 6-1 以索引技巧作區段傳輸

```

BLKMOV  LD      IX, FROM
        LD      IY, TO
        LD      C, COUNT
NEXT     LD      A, (IX)
        LD      (IY), A
        DEC     IX
        DEC     IY
        DEC     C
        JR      NZ, NEXT
        HALT

COUNT  EQU     6          ; 假設資料僅有 6 項。
FROM     DEFW   1112H      ; 定義來源區資料。
        DEFW   1314H
        DEFW   1516H
TO       DEFS    6          ; 目的區預留六個位置。
        END

```

例 6-1 之程式算是一完整的程式了，因為，程式除了完成作業所需之指令外，尚包含了所有必需之資料定義（DEFW 與 EQU）、記憶體空間預留（DEFS）、與告訴組譯程式停止組譯程式（END）

等之虛指令。此外，程式最後亦加了一告訴 Z 80 微處理器，“程式指令至此結束，勿再往下繼續執行”之停止（HALT）指令。這樣的完整程式，就是能直接上機之程式形式。

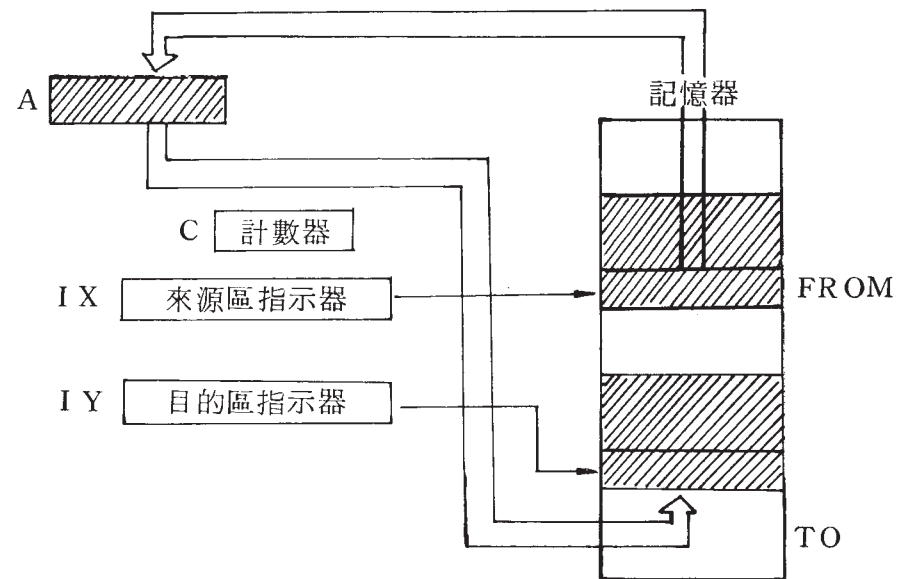


圖 6-6 索引定址之區段傳輸

以 BC 與 DE 作指示器之區段傳輸

於 Z 80，BC 與 DE 暫存器對亦可作為區段傳輸之指示器，不過，此時資料之傳輸僅能以累加器 A 作轉運站。下面之程式，即分別以 BC 與 DE 兩暫存器對作資料來源區段與目的區段之指示器，並以 L 暫存器作位元組計數器（即迴路計數器），之區段傳輸程式。程式一開始，BC 與 DE 分別指至資料來源區與目的區之最高點。

例 6-2 以 BC 與 DE 作指示器之區段傳輸

```

        LD      BC , FROM
        LD      DE , TO
        LD      L , COUNT
NEXT    LD      A , ( BC )
        LD      ( DE ) , A
        DEC     BC
        DEC     DE
        DEC     L
        JP      NZ , NEXT
        HALT

COUNT EQU     4           ; 假設資料僅有四個位元組。
FROM   DEFW    7788H       ; 定義來源區資料。
        DEFW    99AAH
TO     DEFS     4           ; 預留目的區之位置。
        END

```

藉用區段傳輸指令之區段傳輸

於 Z 80，區段傳輸可以區段傳輸指令將之半自動化或全自動化。每一區段傳輸指令均以 H L 暫存器對作資料來源區段之指示器，D E 暫存器對作資料目的區段之指示器，與 B C 暫存器對作位元組計數器。於應用區段傳輸指令前，此些暫存器對必須先佈置妥當。下面之程式即分別以半自動化之 LDD 與全自動之 LDDR 作區段傳輸的情形。注意，於 LDD 之例子，程式在每項資料搬完後，可插入其它型式之處理，再進行搬運次一位元組。

圖 6-7 所示即為區段傳輸之佈置情形。

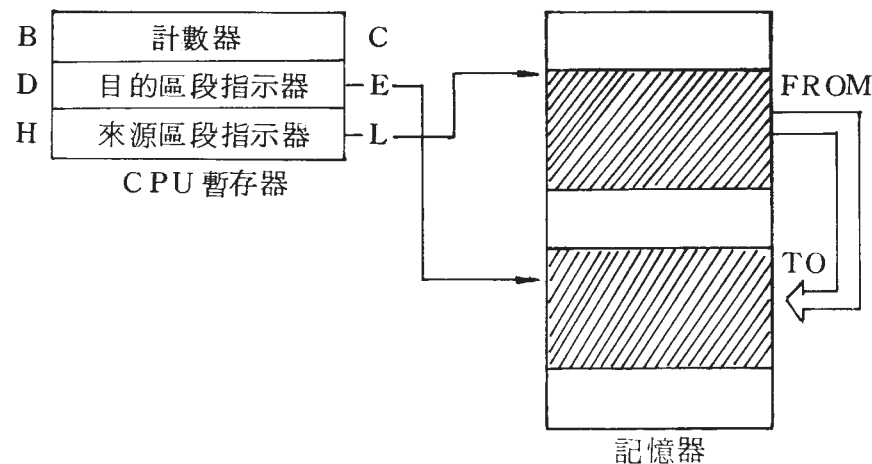


圖 6-7 區段傳送指令使用前所需之佈置

例 6—3 以 LDD 指令作區段傳輸

；該程式以 LDD 指令，將位址 FROM 以下之記憶區段的資料，移至位址 TO 以下之記憶區段。區段之長度為 COUNT。

```

        LD      HL , FROM    ; 設定來源區指示器。
        LD      DE , TO      ; 設定目的區指示器。
        LD      BC , COUNT   ; 設定位元組計數器。
LOOP    LDD                      ; 傳輸一個位元組。
        JP      PE , LOOP
        HALT

COUNT EQU           ; 區段長度填入此。
FROM   DEFW          ; 來源區資料填入此。
        ⋮
TO     DEFS           ; 區段長度填入此。
        END

```


例 6—4 以 LDDR 指令作區段傳輸

；該程式以 LDDR 指令，將位址 FROM 以下之記憶區段的資料，整
；批移至位址 TO 以下之記憶區段，資料區段之長度為 COUNT。

```

LD      HL, FROM    ; 設定來源區指示器。
LD      DE, TO      ; 設定目的區指示器。
LD      BC, COUNT   ; 設定位元組計數器。
LDDR                               ; 整批傳輸。
HALT
COUNT EQU    5      ; 假設區段長度 = 5。
FROM     DEFW   0 0 0 0 H
         DEFW   0 0 0 0 H
         DEFB   0 0 H
TO        DEFS   5
END

```

```

SRLDDR  LD      HL, FROM    ; 加一名稱。
        LD      DE, TO
        LD      BC, COUNT
        LDDR
        RET                               ; 回返指令。
        END

```

讀者可從此例約略看出副程式與一般程式之差別。爲了方便起見，爾後幾章之程式將儘可能寫成副程式形式。讀者若欲單獨測試此些程式，必須先將之修改成一般程式之形式——換掉 RET 指令，以及自己定義運算所需之資料，保留所需之記憶空間等。不然，您就必須另寫一能叫用此些副程式之簡短程式。

6-6 副程式

爲了使更多的人能共享已設計好之程式，完成某一作業之程式通常會寫成“副程式”（subroutine）之形式。事實上，副程式與一般之程式並無多大差別。從表面看，唯一的差異是：一、副程式最後一個被執行的指令是 RET 而非 HALT。二、副程式不能單獨被執行（獨當一面），其僅能被叫用。也因此，其必須具有一“名字”。該名字事實上即爲副程式第一個被執行之指令的標題。三、副程式運算所需之資料通常是“叫用”（CALL）它之程式“送”過來的，而非自己定義的。舉個例子而言，例 6—4 之程式若改成副程式形式，則變成例 6—5 之情形。

例 6—5 以 LDDR 指令作區塊傳輸之副程式

；該副程式以 LDDR 指令，將位址 FROM 以下之記憶區段的資料，
；整批移至位址 TO 以下之記憶區段。資料區段之長度為 COUNT。

第 7 章

算 術 程 式

於介紹過取入指令後，本章，我們開始探討如何以 Z 80 之算術指令，構成各種算術程式。本章採用由淺入深之漸進介紹方式，從最簡單兩數相加之三指令程式，至多段加減程式，逐步發展至軟體之乘算與除算程式。讀完本章以後，讀者將熟悉 Z 80 各算術運算指令之功能與用法，以及各種軟體算術程式如何設計。

本章最後一節順便討論比較指令之應用。

7-1 加 算

7-1-1 八位元加算

最簡單之算術就是兩八位元運算元之相加。假設兩運算元分別儲存於位址 ADR 1 與 ADR 2 之記憶位置，而結果欲存於位址 ADR 3 之記憶位置，則下述之程式可達成目的：

例 7-1 兩八位元運算元相加

；此一程式將位址 ADR 1 與 ADR 2 兩記憶位置之內含相加，結果（
；兩數之和）存於位址 ADR 3 之記憶位置。
；

	LD	A, (ADR1)	; 第 1 運算元取入累加器。
	LD	HL, ADR2	; 第 2 運算元位址取入 HL。
	ADD	A, (HL)	; 第 2 運算元加至累加器。
	LD	(ADR3), A	; 結果存回記憶體。
	HALT		; 結束。
ADR1	DEFB	25H	; 設第 1 運算元為 23H。
ADR2	DEFB	43H	; 設第 2 運算元為 45H。
ADR3	DEFS	1	; 為結果保留一記憶位置。
	END		; 程式完了。

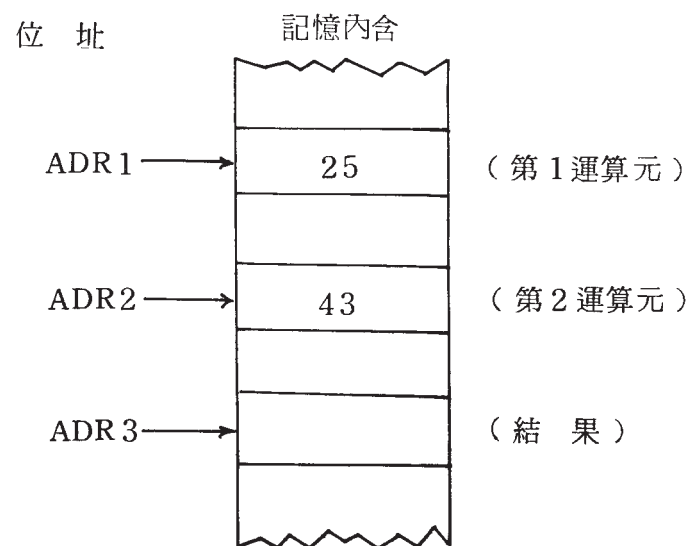


圖 7-1 八位元加算。25H + 43H = 68H

圖 7-1 所示即為例 7-1 之程式所處理之資料儲存在記憶體內之情形。為了解說方便起見，我們假設第 1 運算元與第 2 運算元分別為 25H 與 43H。

圖 7-2，7-3 與 7-4 所示則為例 7-1 程式之執行情形。如圖 7-2 所示，Z80 CPU 執行第一指令時，先透過位址巴士，將運算元位址

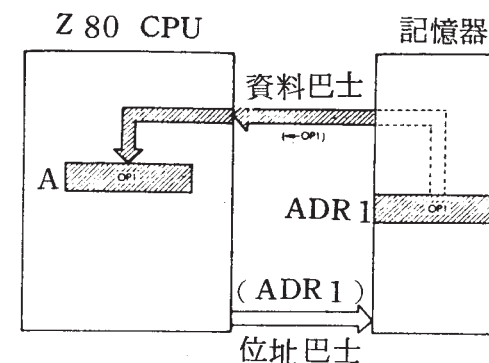


圖 7-2 LD A, (ADR1): 自記憶體取入第 1 運算元。

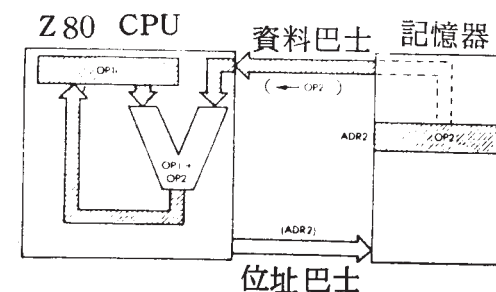


圖 7-3 ADD A, (HL): 第 2 運算元加至第 1 運算元。

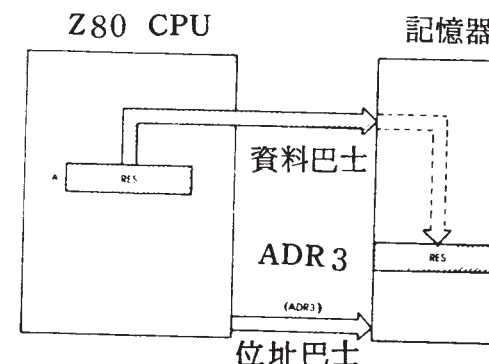


圖 7-4 LD (ADR3), A: 結果存回記憶體。

Z 80 微電腦軟體硬體

ADR1 (事實上是送數值位址) 送給記憶器，該記憶位置之內含然後被讀取 (read)，並送至 Z 80 CPU，寫入 (write) 累加器 A。之後，Z 80 CPU 執行第二指令，將位址 ADR2 (實際為數值位址) 取入 HL 暫存器。

如圖 7-3 所示，Z 80 執行第三指令時，HL 暫存器對之內含先經位址巴士送至記憶器，被選取之記憶位置的內含然後由記憶器送至 ALU 之右輸入，與累加器內含 (ALU 左輸入) 相加，所得之和再存回 (寫入) 累加器 A。最後，Z 80 CPU 執行 LD (ADR3), A 指令，將累加器內含 (兩數之和) 存至 (寫入) 位址 ADR3 之記憶位置。

特別注意，寫入動作改變暫存器或記憶位置之原有內含，但讀取動作並不。因之，於上述程式執行完後，位址 ADR1 與 ADR2 兩記憶位置之內含並未改變，但累加器 A 與位址 ADR3 之記憶位置，兩者之內含皆同變為前述兩位置內含之和。

習題 7-1：位址 LOC1 與 LOC2 兩記憶位置分別存 40 與 73，寫一程式將兩數相加，並將結果存於位址 LOC3 之記憶位置。

7-1-2 十六位元加算

八位元加算僅能作兩八位元數目之相加，所得結果最大僅能為 255 (若採用直接二進數)，這對許多應用而言是不足的。經常，我們必須作兩十六位元 (或更長) 數目之相加。此種多倍長數目之算術，即稱為 **多倍長算術** 或 **多段算術** (multiple-precision arithmetic)。下面，我們介紹一將兩十六位元數目相加之程式，讀者可應用同一原理。作更長數目之加減運算。

如圖 7-5 所示，於此一例子，假設兩十六位元數目分別存於位址 ADR1，ADR1+1 與 ADR2，ADR2+1 之記憶位置，且結果存於位址 ADR3 與 ADR3+1 兩記憶位置。於每一種情況，低次位元組均儲存於位址較小之位置。

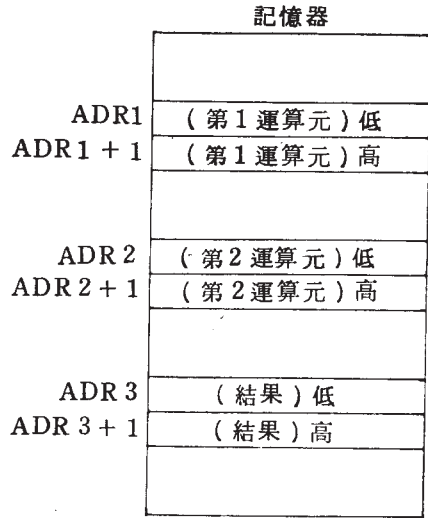


圖 7-5 十六位元加算——運算元擺置。

例 7-2 十六位元加算

；該程式將儲存於位址 ADR1 及 ADR1+1，與 ADR2 及 ADR2+1；之記憶位置的兩十六位元數目相加，結果存於位址 ADR3 與 ADR3+1 兩記憶位置。於每一種情況，低次位元組均儲存於位址較小；之記憶位置。

```
LD    A, (ADR1)      ; 取入第 1 數之低位元組。
LD    HL, ADR2        ;
ADD   A, (HL)         ; 兩低次位元組相加。
LD    (ADR3), A       ; 低次位元組和存起。
LD    A, (ADR1+1)     ; 取入第 1 數之高位元組。
INC   HL              ; 指至第 2 數之高位元組。
ADC   A, (HL)         ; 兩高次位元組相加。
LD    (ADR3+1), A     ; 存出高位元組之和。
HALT
```

```

ADR1  DEFW  2534 H      ; 設定第一數目為 2534 H。
ADR2  DEFW  6503 H      ; 設定第二數目為 6503 H。
ADR3  DEFS   2           ; 為結果預留兩記憶位置。
      END                ;

```

如例 7-2 所示的，於多段加算，最低次之位元組可以 ADD 指令相加，但爾後之各位元組則須以 ADC 指令相加，以計入較低次位元組所產生之進位。有人或許會問，若最高次位元組又產生進位，怎麼辦呢？若預測會有此種情況發生，則程式必須自行以額外之指令測試，並採取適當之措施。

上述之程式是屬於使用累加器之傳統作法。於 Z 80，HL 暫存器對在有限的情況下，亦可作為一十六位元之累加器。下面之例子即以 BC 與 HL 兩暫存器對，作十六位元加算的情形。

例 7-3 以 BC 與 HL 暫存器對作十六位元加算。

；該程式以 BC 與 HL 兩暫存器對，重作例 7-2。HL 暫存器對充當十六位元累加器。

；

```

LD      HL, (ADR1) ; 第 1 運算元取入 HL。
LD      BC, (ADR2) ; 第 2 運算元取入 BC。
ADD     HL, BC      ; 作十六位元相加。
LD      (ADR3), HL  ; 結果存入記憶器。
HALT

```

ADR1 DEFW 2534 H ; 資料定義。

ADR2 DEFW 6503 H

ADR3 DEFS 2 ; 記憶空間保留。

END

注意圖 7-6 所示之十六位元取入的情形。

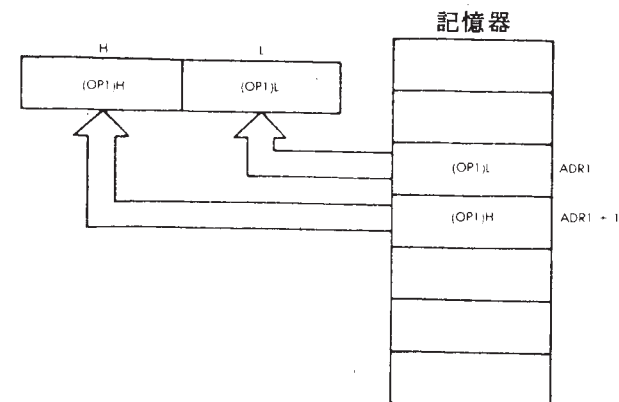


圖 7-6 十六位元取入——LD HL, (ADR1)

習題 7-2：設低次位元組儲存於位址較低（小）之記憶位置，重作例 7-2。

習題 7-3：以剛介紹過之十六位元指令，作兩三十二位元數目之相加。設較低次位元組存於位址較高（大）之記憶位置。

7-2 減 算

減法運算之程式類同於加法運算程式，唯一的差別是，加法指令必須換成減法指令。八位元之減算過於簡單，致直接留作習題。此地，我們介紹一十六位元減算之例子。

習題 7-4：寫一八位元減算程式。

例 7-4 以 HL 與 BC 暫存器對作十六位元減算。

；該程式以 HL 與 BC 兩暫存器對作十六位元之減算。被減數與減數；分別儲存於 ADR1 與 ADR2 起之記憶位置，低次位元組在前（位址小之位置），高次位元組在後。兩數之差則以同樣方式存於位址；ADR3 起之記憶位置。

```

LD      HL, (ADR1) ; 被減數取入 HL。
LD      BC, (ADR2) ; 減數取入 BC。
AND     A           ; 清除進（借）位旗號。
SBC     HL, BC      ; 兩十六位元數目相減。
LD      (ADR3), HL ; 結果存入預留記憶位置。
HALT
ADR1    DEFW                ; 程式執行前，兩數
ADR2    DEFW                ; 目必須填入此些位置。
ADR3    DEFS      2         ; 為結果預留兩個記憶位置。
END

```

例 7-4 之程式主要像十六位元之加算程式。不過，由於 Z80 具有兩個十六位元加法指令：ADD 與 ADC，而僅有一十六位元減法指令：SBC。因此，有兩點不太相同。第一，ADD 指令換成 SBC 指令。第二，減算之前必須加一“AND A”指令，以清除進位。記住，AND A 指令執行之結果並不改變累加器之原有內含，但却將進位旗號清除為零。

習題 7-5：勿使用十六位元指令，重寫一十六位元減算程式。

記得，於 2 補數算術，進位旗號之最後內含並無意義。若減算結果產生溢位，則溢位旗號（V）會置定為 1，並可加以測試。

以上所舉之例子均為二進加算與減算。下面，我們看看 BCD 算術。

7-3 BCD 算術

7-3-1 八位元 BCD 加算

BCD 算術之觀念於第一章已經提過，在此，讓我們再回憶一下。於 BCD 表示，每四位元用以儲存一十進數字（0 至 9）。因此，每八位元就能儲存兩位 BCD 數字（這稱為**濃縮 BCD**）。

由於 BCD 表示僅使用四位元二進碼之前十種組合，留下後面六種組合未用。因此，若將兩 BCD 數目相加，就有問題產生：祇要兩數字之和的大於 9，其結果就必須再加 6，方能使之越過六個無用之電碼，使之達到十進算術之進位。例如，

8 之 BCD 碼為	1000	
3 之 BCD 碼為	+ 0011	
	1011	← 無定義之 BCD 碼
	+ 0110	← 加 6
	0001 0001	← 正碼結果 11 ₁₀ 。

於計算機內部，算術 / 邏輯單元永遠將任何數目按二進數相加（或相減）。因此，兩 BCD 數目在以二進算術指令運算後，所獲結果必須立即加以校正，方能正確。於一般處理器，此一工作即由 DAA（Decimal Adjust Accumulator，十進調整）指令達成。DAA 指令會根據進位與半進位兩旗號，自動檢視剛剛運算之結果，並於必要時將結果加 6，使之符合十進算術之結果。此一指令通常緊接加法指令之後。

例 7-5 八位元 BCD 加算

；程式將位址 ADR 1 與 ADR 2 兩記憶位置所含之兩數作 BCD 相加，
 ；結果存於位址 ADR 3 之記憶位置。DAA 指令用以調整二進加算之
 ；結果，使之符合 BCD 結果。

```

LD      A, (ADR1)      ; 取入第一 BCD 數目。
LD      HL, ADR2       ; HL 指至第二 BCD 數目。
ADD     A, (HL)        ; 兩 BCD 數相加。
DAA                      ; 調整運算結果。
LD      (ADR3), A      ; 十進結果存入記憶器。
HALT                    ; 完了
ADR1    DEFB    39 H   ; 設第一數為 39 H。
ADR2    DEFB    18 H   ; 設第二數為 18 H。
ADR3    DEFS    1      ; 兩數之和存於此。
END

```

假若位址 ADR 1 與 ADR 2 兩記憶位置之內含分別為 39 H 及 18 H，
 則上述程式執行完後，最後位址 ADR 3 之記憶位置的內含必為 57 ($0101\ 0111_2$)，而非直接二進算術之結果 51 ($0101\ 0001_2$)。

BCD 減算

BCD 減算類似於 BCD 加算。DAA 指令緊接置於減法指令之後，
 以調整減算所獲之結果。實際上，減算結果之十進調整與加算不同。
 當某一數位減算所獲結果大於 9 時，其結果必須再 **減 6**，而非加 6。
 Z 80 旗號暫存器之減法旗號 (N)，即用以記錄 Z 80 CPU 剛剛所進
 行的是加法或減法運算，以作 DAA 指令調整之依據。

7-3-2 十六位元 BCD 加算

十六位元 BCD 加算之程式同於二進加算之情況，唯一不同的是，
 每一加法指令之後必須增加一 DAA 指令。下面即為一例。

例 7-6 十六位元 BCD 加算

；該程式將兩十六位元之 BCD 數目相加，結果存於位址 ADR 3 與
 ；ADR 3 + 1 兩記憶位置。欲相加之兩數目分別儲存於 ADR 1 及
 ；ADR 1 + 1，與 ADR 2 及 ADR 2 + 1 之位置。低次位元組均位於位
 ；址較低 (小) 之記憶位置。

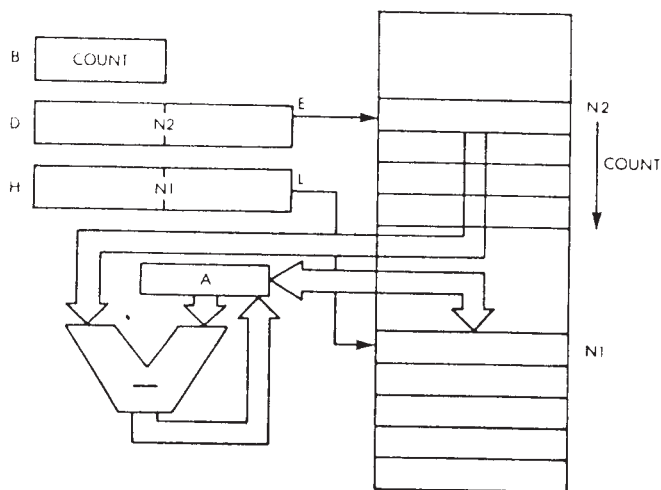
```

LD      A, (ADR1)      ; 第 1 數之 LSB 取入 A。
LD      HL, ADR2       ; HL 指至第 2 數之 LSB。
ADD     A, (HL)        ; 兩數之 LSB 相加。
DAA                      ; 調整運算所得結果。
LD      (ADR3), A      ; 結果存入預留記憶位置。
LD      A, (ADR1+1)    ; 第 1 數之 MSB 取入 A。
INC     HL             ; HL 指至第 2 數之 MSB。
ADC     A, (HL)        ; 兩數之 MSB 相加。
DAA                      ; 調整運算所得結果。
LD      (ADR3+1), A    ; MSB 結果存起。
HALT                    ; 完了。
ADR1    DEFW          ; 設定第 1 數目(用者自填)。
ADR2    DEFW          ; 設定第 2 數目。
ADR3    DEFS    2      ; 儲存結果之位置。
END

```

7-3-3 濃縮BCD減算

基本之BCD加算與減算皆已討論過，但實際應用上，BCD數目可能包括數個位元組。下面，我們介紹一一般化之濃縮BCD數目減算程式，以結束BCD算術之討論。任何長度之BCD加算或減算，皆可同理推之。於此例子，假設欲相減之兩濃縮BCD數含有同等數量之位元組，並分別儲存於位址N1與N2起之記憶位置，低次位元組在前（位址小之位置）。COUNT代表兩數目之長度（以位元組數計）。暫存器與記憶器之分配情形如圖7-7所示。

圖 7-7 濃縮BCD減算： $N1 \leftarrow N2 - N1$

為方便起見，例7-7之程式寫成副程式形式。讀者若欲真正測試此一程式，可自行設定COUNT之值，並且以DEFB虛指令定義兩BCD數目，將副程式變成一般程式形式（RET指令換掉）。

例 7-7 通用之濃縮BCD減算程式

；該程式將兩分別儲存於N1與N2起之記憶位置的濃縮BCD數相減； $(N2 - N1)$ ，結果存回N1起之記憶位置。假設兩數所含之位元組數同為COUNT。

```

;
BCDPAK  LD      B,COUNT      ; B 為迴路計數器。
        LD      DE,N2        ; DE 為被減數指示器。
        LD      HL,N1        ; HL 為減數指示器。
        AND     A            ; 清除進位。
MINUS    LD      A,(DE)       ; 取入被減數。
        SBC     A,(HL)       ; 減去減數。
        DAA                     ; 十進調整。
        LD      (HL),A
        INC     DE            ; 指至次一位元組。
        INC     HL
        DJNZ    MINUS        ; B 值減 1，迴路
                                   ; 至 B = 0 為止。

        RET
        END

```

習題 7-6：試問二進算術程式與BCD算術程式有何不同？

習題 7-7：於例7-7，BC與HL之角色能對調嗎？（提示：注意SBC之定址法）。

習題 7-8：寫一十六位元BCD減算程式。

順便一提的是，於BCD型態，在加算時，進位旗號顯示結果是

否大於 99，這與 2 補數之情況不同，因為，BCD 數字並非以真正二進數表示。反之，在減算時，進位旗號代表借位。

7-4 乘 算

接著，我們探討一更複雜之算術問題：二進數之乘算。為得二進乘算之演算法，首先，我們檢視日常之十進乘算。就以 12×23 為例

$$\begin{array}{r}
 12 \quad (\text{被乘數}) \\
 \times 23 \quad (\text{乘數}) \\
 \hline
 36 \quad (\text{部份積}) \\
 24 \\
 \hline
 276 \quad (\text{最後結果})
 \end{array}$$

乘算首先以乘數最右邊之數字乘被乘數，即 3×12 ，得部份積 36。之後，乘數之次一較高次數字再乘被乘數，亦即 2×12 ，得部份積 24。兩次部份積相加，所得即為最後結果。

但是，尚有一個運算：24 左移了一位。或者我們亦可說，第一次部份積 (36) 於相加前右移了一位。不論何者，祇要部份積正確移位，然後相加，所得即為正確之最後乘積。二進乘算就是以與此完全相同之方式作業的！如，

$$\begin{array}{r}
 (5) \quad 101 \quad (\text{被乘數, MPD}) \\
 (3) \times 011 \quad (\text{乘數, MPR}) \\
 \hline
 101 \quad (\text{部份積, PP}) \\
 101 \\
 000 \\
 \hline
 (15) 01111 \quad (\text{最後結果, RES})
 \end{array}$$

圖 7-8 所示即為一基本乘數演算法之流程圖。前面說過，流程圖就是演算法的符號表示。每矩形代表一欲執行之命令，可翻成一個或一個以上之機器指令。而菱形則代表測試，其即為程式流程之分支點

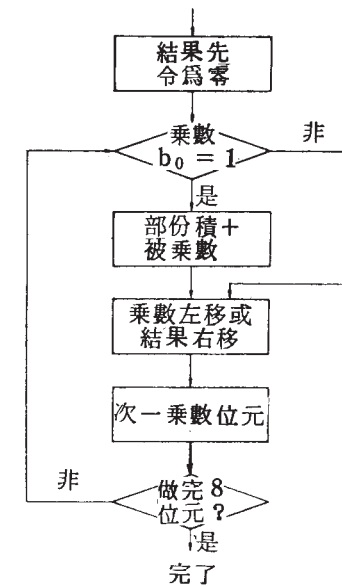


圖 7-8 乘算演算法之基本流程圖

。若測試成功（指明之條件滿足），則程式控制將緊接跳至某一指明之位置；若測試失敗，則控制不發生跳越，繼續執行次一緊接指令。分支之觀念在爾後程式中會再作更詳細之說明。就讀者而言，此刻最重要的，就是詳細閱讀圖 7-8 之流程圖，並看其是否真正代表了我們欲表示之演算法。注意，最後一菱形有一箭頭流出，回至上一菱形之頂。此乃因為同一部份之流程必須重複執行八次之原故，乘數之每位元一次。這種由同一點再從頭開始執行之情形，稱為**程式迴路**（program loop）

習題 7-9：依據圖 7-8 之流程圖，以二進制做“ 4×7 ”，並證明所得結果為 28。若不，則再試，唯有當您獲致正確結果時，您才有辦法將流程圖轉換成程式。

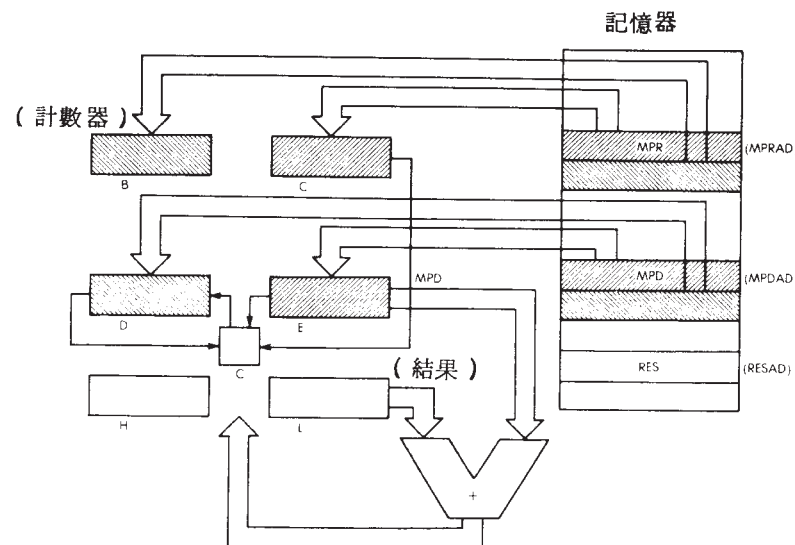
7-4-1 8×8 乘算

現在，我們將圖 7-8 之乘算流程圖，轉換成 Z80 程式。此一程式即如例 7-8 所示。

例 7-8 8×8 乘算程式

；該程式履行兩八位元數目之乘算。乘數由位址 MPRAD 之記憶位置
；取入 C 暫存器，被乘數則自位址 MPDAD 之記憶位置取入 E 暫存器
；。最後乘積置於 HL 暫存器對，並存至位址 RESAD 之記憶位置。
；

MPY88	LD	BC, (MPRAD)	；乘數取入 C 暫存器。
	LD	B, 8	；B 為位元計數器。
	LD	DE, (MPDAD)	；被乘數取入 E 暫存器。
	LD	D, 0	；清除 D。
	LD	HL, 0	；結果先令為零。
MULT	SRL	C	；乘數位元移入進位旗號。
	JR	NC, NOADD	；測試進位，若零， ；則略過加法指令。
	ADD	HL, DE	；否則，被乘數加至累加部份積。
NOADD	SLA	E	；被乘數左移一位。
	RL	D	；位元存入 D。
	DEC	B	；位元計數值減一。
	JP	NZ, MULT	；若未零，則再來。
	LD	(RESAD), HL	；否則，最後乘積存起。
	HALT		
MPRAD	DEFB	8	；數據設定：乘數 = 8。
MPDAD	DEFB	7	；數據設定：被乘數 = 7。
RESAD	DEFS	2	；記憶空間預留。
	END		

圖 7-9 8×8 乘算——暫存器佈建。

標題 MULT 以前之五個指令為佈建指令，對應於流程圖之第一方塊。如圖 7-9 所示，此些指令於正式乘算開始前，事先安排好各暫存器之內含。乘數與被乘數原來分別儲存於位址 MPRAD 與 MPDAD 之記憶位置，其取入暫存器的情形分別如圖 7-10 與 7-11 所示。注

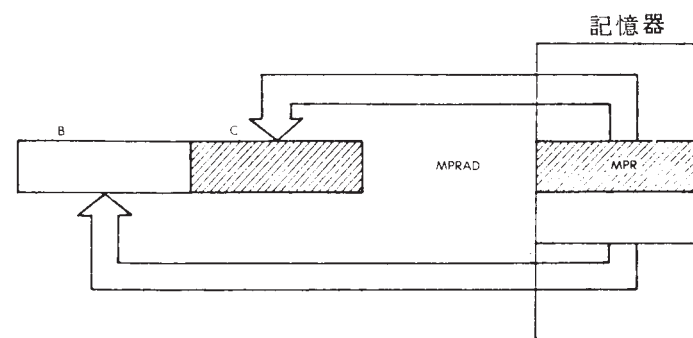


圖 7-10 八位元乘數取入 C 暫存器：LD BC, (MPRAD)

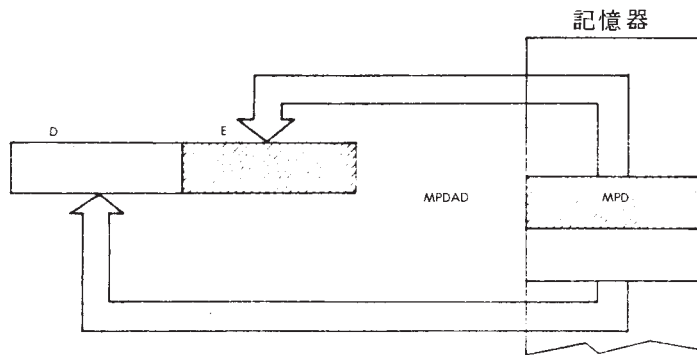


圖 7-11 八位元被乘數取入 E 暫存器：
LD DE, (MPDAD)

意，由於乘數與被乘數分別僅有八位元，但擴展定址之取入指令必須採用十六位元。因此十六位元取入後，B 與 D 暫存器必須再分別置定為“8”與“0”。此外，八位元與八位元之乘積為十六位元，並且部份乘積必須逐次累加，因之，部份（與最後）結果必須儲存於 HL 暫存器對。

乘算開始時，程式先測試乘法位元是否為 1（SRL C 與 JR NC, NOADD）。若是，則部份積直接加至最後結果（ADD HL, DE）；若非，則控制略過加法指令（因乘數位元為零，部份乘積必零，故不必累加）。然後，被乘數左移一位（SLA E 與 RL D）。位元計數值減一（DEC B），若其值為零，則表示八位元皆已乘完，最後乘積存至位址 RESAD 之記憶位置（最後一指令）；否則，控制傳回 MULT 處之指令，繼續次一位元之乘算。

有幾點必須指出：第一，乘數位元值之測試亦可採位元測試指令 BIT。然而，由於此一指令上必須標明欲測試第幾位元，致每一位元之測試勢必個別寫一 BIT 指令。因此，鑑於其無法寫成迴路形式，我們未加採用。第二，部份積之累加必須使用十六位元加算指令，同時，被乘數在被加至累加積之前必須先左移一位。由 Z80 並無十六

位元移位指令，因之，DE 暫存器對之內含（被乘數）左移一位，必須先將 E 暫存器作算術左移（使最高次位元進入進位，最低次位元補 0），然後，再對 D 暫存器左旋轉（RL D），使剛移出 E 暫存器最高次位元者（即進位之值），進入 D 暫存器之最低次位元。此一運算即如圖 7-12 所示。

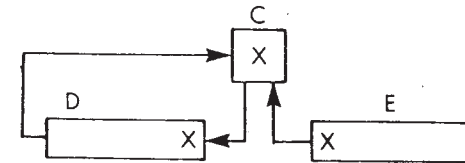


圖 7-12 十六位元 DE 暫存器對左移一位

第三，被乘數（或部份積）第一次加至最後結果前不必左移，故左移運算擺在加算之後。特別注意的是，即令被乘數並無加至最後結果，其值仍須左移一位。最後，將迴路計數器 B 之值減一，並測試迴路是否可結束之運算，於 Z80 可以一十分便捷之 DJNZ 指令達成。依此，原來程式之 DEC B 與 JP NZ, MULT 兩指令，可以 DJNZ MULT 一指令取而代之。

習題 7-10：您能以 BIT 指令取代 SRL C 指令，重作例 7-8 嗎？這樣作法的缺點為何？

習題 7-11：檢視例 7-8 程式開頭之 LD D, 0 與 LD HL, 0 指令，可否以

```
XOR    A
LD     D, A
LD     H, A
LD     L, A
```

之指令系列將之取代呢？若可，那對程式所佔記憶空間與執行速度有

何影響呢？

自我測試

由於截至目前為止，例 7-8 之程式算是“稍”較複雜的一個。因此，作者建議讀者，自己假設兩個數目，開列一張能記錄各有關暫存器內含之表格，將自己當成電腦，自程式最開始，一一執行程式之每一指令，並隨時記錄每一運算之結果。直至程式完全結束後，再核算最後之結果是否正確。如此做，不但可使您更深刻地認識每一指令，同時亦可使您更加熟悉程式控制之流程，以及有些指令為何必須如此安排。

譬如，圖 7-13 所示即為假設乘數為 3，儲存於位址 0200H 之記憶位置，與被乘數為 5，儲存於位址 0202H 之記憶位置，的自我測試情形。如(d)圖所示，當程式執行完第一次迴路後，部份積(DE)已變為十，而最後結果(HL)已累加至 5。希望讀者能自行先作一遍，再將結果與圖 7-13 作一比較。全部過程中，Z 80 各暫存器與旗號之內含變化情形清單，附於圖 7-20。

標 題	指 令	B	C	C	D	E	H	L
MP488	LD BC, (0200)	--	--	-	--	--	--	--

(a) 第一指令執行後

標 題	指 令	B	C	C	D	E	H	L
MP488	LD BC, (0200)	00	03	-	--	--	--	--
	LD B, 08	08	03	-	--	--	--	--

(b) 前兩指令執行後

圖 7-13 例 7-8 程式之自我執行測試

標 題	指 令	B	C	C	D	E	H	L
MP488	LD BC, (0200)	00	03	-	--	--	--	--
	LD B, 08	08	03	-	--	--	--	--
	LD DE, (0202)	08	03	-	00	05	--	--
	LD D, 00	08	03	-	00	05	--	--
	LD HL, 0000	08	03	-	00	05	00	00

(c) 前五個指令執行後

標 題	指 令	B	C	C	D	E	H	L
MP488	LD BC, (0200)	00	03	-	--	--	--	--
	LD B, 08	08	03	-	--	--	--	--
	LD DE, (0202)	08	03	-	00	05	--	--
	LD D, 00	08	03	-	00	05	--	--
	LD HL, 0000	08	03	-	00	05	00	00
MULT	SRL C	08	01	1	00	05	00	00
	JR NC, 0114	08	01	1	00	05	00	00
	ADD HL, DE	08	01	0	00	05	00	05
NOADD	SLA E	08	01	0	00	0A	00	05
	RL D	08	01	0	00	0A	00	05
	DEC B	07	01	0	00	0A	00	05
	JPNZ, 010F	07	01	0	00	0A	00	05

(d) 迴路第一巡迴過後

圖 7-13 例 7-8 程式之自我執行測試

其它方法

八位元乘算程式，除了例 7-8 之作法外，事實上還有許多其它之方法。譬如，於相加之前將被乘數右移而不左移，即為一例。為了節省篇幅，且不散讀者之注意力，我們將此留作習題。

習題 7-12：以同樣之演算法再寫一 8×8 之乘法程式，不過，

此回被乘數在被加至累加積之前，先將之右移而勿左移。

緊接我們所要探討的，是例 7-8 之程式能否寫得再更精簡之問題。

7-4-2 改良之 8×8 乘算程式

例 7-8 之八位元乘算程式，除了迴路結束測試之兩指令可以 DJNZ 指令將之取代簡化外，尚可作進一步之精簡。我們現在就看這個問題。

例 7-8 之程式總共作了三次不同之移位運算。乘數位元先右移，然後被乘數再作兩次左移；E 暫存器先左移，D 暫存器復左旋轉。這種作法非常浪費時間。二進乘算最常用的一種標準程式設計“技巧”，即是基於下列之着眼：**乘數每右移一次，乘數暫存器就空出一位元可用**。譬如，若乘數右移一位，則最左邊之位元即空出可用。同時，此時之部份乘積（或“結果”）最多亦僅佔九位元。因此，若程式開始之初，乘算結果僅以一八位元暫存器加以儲存，則此時我們就可利用乘數移位過程中所空出之位元位置，以儲存同時增長之乘積。

換言之，**乘數每移位一次，部份乘積正巧增長一個位元。是故，程式一開始時，我們可先僅以一八位元暫存器儲存部份乘積，然後，再以乘數移位過程所空出之位元位置，儲存同時累增之乘積**。為此，我們必須將乘數與部份乘積儲存於同一暫存器對，並且將兩者往同一方向（往乘數所在的一邊）移位，使乘數每次移掉一個位元，而部份乘積（結果）每次增長一個位元。最好，我們亦能以單一指令將兩者同時移位。不幸的是，正如其它八位元之微處理器一樣，Z80 並無十六位元之移位指令。

不過，這並沒什麼關係，尚有另一技巧可施。前面已經看過，Z80（8080 亦同）具有一特殊十六位元加法指令。祇要乘數與結果均儲存於 HL 暫存器對，我們即可使用

ADD HL, HL

指令，將 HL 暫存器對之內含加倍。於二進制，將一數值加倍，即等於將之左移一個位元位置。因此，上述指令之效用，即相當於一十六位元之移位指令——將乘數與結果同時左移。遺憾的是，此種移位是向左移，而非如我們所希望的向右移——移出乘數之最低次位元。這也沒什麼大問題。

觀念上，乘數若不右移，亦可左移，由於平常加算均採右移，因此，前面我們用了右移。不過，並非一定要如此不可。將乘數左移同樣見效，因為，加法是適合於交換律的。

為了利用此一十六位元模擬移位，我們必須將乘數左移。如圖 7-14 所示，乘數駐於 H 暫存器，而結果存於 L 暫存器。

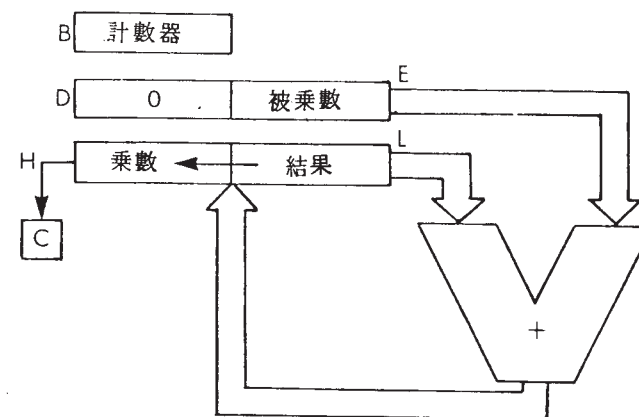


圖 7-14 改良乘算之暫存器配置圖

例 7-9 改良之 8×8 乘算程式

；該程式為例 7-8 之 8×8 乘算程式的改良。主要的改變是，乘數與最後乘積儲存於同一暫存器對 HL。程式一開始時，H 儲存乘數，L 存放結果。此外，新程式已看不見移位或旋轉指令，而以 ADD HL, HL 指令模擬十六位元之移位，將乘數與累加結果同時左移。

```

;
MUL88C LD    HL, (MPRAD-1); 乘數取入 H。
        LD    L, 0          ; 結果清除為零。
        LD    DE, (MPDAD)   ; 被乘數取入 E。
        LD    D, 0          ; D清除為零。
        LD    B, 8          ; 位元計數值取入 B。
MULT    ADD    HL, HL        ; 乘數與結果同向左移。
        JR    NC, NOADD     ; 測試乘數位元。
        ADD    HL, DE        ; 若 1，則被乘數加
                                ; 至結果。
NOADD   DJNZ   MULT
        LD    (RESAD), HL
        RET                  ; 該程式寫成副程式形式。

```

此一程式寫成副程式 (subroutine) 之形式，什麼是副程式將留待後面再詳加討論。目前，讀者只要記住，副程式與一般程式的區別是，其最後一指令必為 RET 指令就夠了。

注意，例 7-8 程式之作法是乘數由最右邊之位元開始乘起，而每次部份積欲加至結果之前，結果先右移一位（若以部份積為基準，即部份積不動）。例如，

1 1	被乘數
× 0 1	乘 數
——	
1 1	累積結果
+ 0 0	新得部份積
——	
0 1 1	

而例 7-9 程式的作法正巧相反。此一程式由乘數最左邊之位元開始乘起（因為 ADD HL, HL 指令每次將乘數最左邊之位元，移入進位

旗號，如圖 7-14 所示），而每次新得之部份積欲加至累積結果之前，累積結果先左移（ADD HL, HL 指令）。其情形就彷彿下述例子：

1 1	被乘數
× 0 1	乘 數
——	
0 0	第一次部份乘積作累積結果
+ 1 1	新得部份積
——	
0 1 1	

若以新得部份乘積為準，相加前，累積結果等於“左”移一位。

除了迴路結束測試之兩個指令為 DJNZ MULT 所取代外，例 7-8 程式之三個移位與旋轉指令，於例 7-9 程式亦為一 ADD HL, HL 指令所取代。因此，兩程式相比，例 7-9 程式之乘算迴路相對簡短多了。這主要得力於資料在暫存器安排得“巧妙”。

至此，我們可獲致一點心得。直截了當之設計方式雖然同樣可產生能運作之程式，但其通常不是最好的 (optimized)。欲設計一最精簡之程式，程式設計者必須能做到將現有之暫存器與指令，作最佳之密切配合。這當然需要經驗啦！「做中學」之真意也就在此。多作題目，多寫程式，與多讀他人之程式，很快您的程式設計技巧就會有所長進！

習題 7-13：計算例 7-9 之八位元乘算程式的執行時間。JR 跳越指令之跳越成功率以 50% 計。（參考指令詳細表，時序頻率以 2 MHz 計）。

習題 7-14：假若被乘數必須存放於 B 與 C 暫存器，則例 7-9 之

程式必須作何改變呢？

習題 7-15：為何被乘數取入 E 暫存器後，D 暫存器欲再清除為零呢？

7-4-3 16×16 乘算

為了測試方所獲致之技巧，緊接，我們試看兩十六位元數目之乘算。不過，為了簡單起見，我們假設兩數之乘積不超過十六位元，並可容納於一暫存器對。如圖 7-15 所示，最後乘積仍存於 HL 暫存器對，而被乘數置於 DE 暫存器對。雖然將乘數置於 BC 暫存器對十分迷人，但為了利用 DJNZ 指令，亦惟有忍痛割愛，將乘數分開儲存於 C（高次八位元）與 A（低次八位元）兩暫存器。

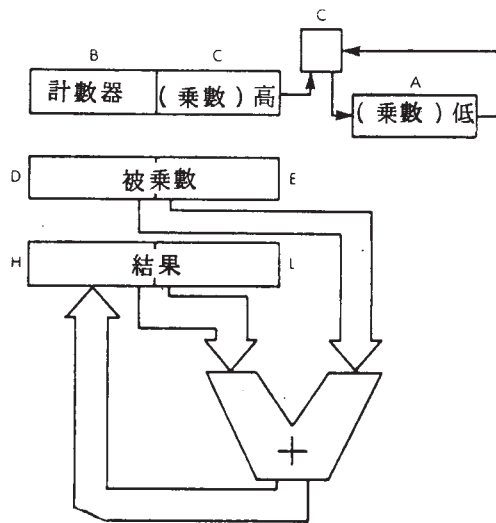


圖 7-15 16×16 乘算：暫存器安排

例 7-10 兩十六位元之乘算程式

；該副程式履行兩十六位元數目之乘算。乘數與被乘數假設原先皆已

；分別存於位址 MPRAD 與 MPDAD，以及其次一緊接位置。於乘算過程，暫存器的安排是：HL：結果，DE：被乘數，C：乘數高次八位元，A：乘數低次八位元，B：位元計數器。

；

```

MUL16  LD      A, (MPRAD+1) ; 乘數高位元組
        LD      C, A        ; 取入 C 暫存器。
        LD      A, (MPRAD)  ; 取入乘數低位元組。
        LD      B, 16       ; 設定位元計數值。
        LD      DE, (MPDAD) ; 取入被乘數。
        LD      HL, 0       ; 結果先令為零。
MULT    SRL     C            ; 乘數高位元組右移。
        RRA          ; 乘數低位元組右旋轉。
        JR      NC, NOADD   ; 測試乘數位元。
        ADD     HL, DE      ; 被乘數加至結果。
NOADD   EX      DE, HL
        ADD     HL, HL      ; 被乘數左移一位。
        EX      DE, HL
        DJNZ    MULT
        RET

```

此一程式與先前者類似。前六指令為暫存器佈建指令，將暫存器設定於適當之起始值。緊接之兩指令對乘數作十六位元之右移。移位後，乘數之最低次位元即進入進位旗號。JR NC, NOADD 指令緊接測試其值。若其值為 1，則 ADD HL, DE 指令將被乘數加至結果；否則，控制略過加法指令。標題 NOADD 以下之前三個指令，主要在將被乘數（即部份乘積）左移一位（顯然，此一作法同於傳統之乘算習慣）。由於僅有 HL 暫存器對能自加，致被乘數必須先由 DE 搬至 HL，運算完後再搬回 DE（兩 EX 指令）。DJNZ MULT 指

令最後將位元計數器 B 之值減一，若其值未達零，控制跳回 MULT 處，繼續乘下一位元。否則，若 B 值為零，程式結束。控制回返 (RET)。

習題 7-16：將乘數置於 BC 暫存器對，以累加器 A 作計數器，重寫十六位元之乘算程式，並測試結果是否正確。

習題 7-17：於例 7-10，您能提出一將含於 DE 之被乘數移位，而不必借用 HL 之方法嗎？

習題 7-18：試寫一兩十六位元數目相乘，產生三十二位元結果之乘算程式。暫存器的分配可如圖 7-16 所示。記得，迴路內第一次加算的結果最長僅為十六位元，而每巡迴一次，乘數即會空出一位元位置。

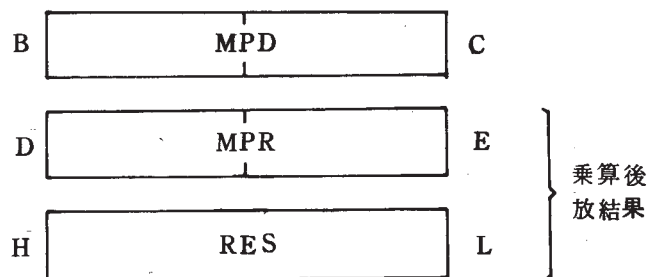


圖 7-16 兩十六位元數目相乘，得三十二位元結果之暫存器安排。

7-5 除 算

雖然以軟體程式作乘算或除算之方法很多，但是，其中大多皆屬不經濟的。前一節說過，乘算可以逐次移位與累加之方式達成。相反的，除算亦可以逐次移位與累減為之。除此之外，乘算與除算尚可以

連加 (乘算) 與連減 (除算)，或甚至是“查表” (look-up table) 之方式達成。不過，此些方法皆具有一共同缺點——執行速度緩慢。

除算最直接的一種方法就是連減法：被除數連續地減去除數，每減一次，商即加一。減算程序一直進行至所得結果為負 (即不夠減) 時停止。例 7-11 所示之程式即是一例。該程式以連減法求兩無號數之商，其中，被除數為十六位元，除數為八位元。

例 7-11 直接連減法之二進除算 (16/8)

該副程式以連減方式，求出一十六位元數目，除以一八位元數目之商。被除數假設事先存於位址 DIVDND 與 DIVDND + 1 兩記憶位；置 (低次位元組在前)，而除數則存於位址 DIVSOR 之記憶位置。運算過程中，暫存器之分配情形為——HL：被除數；BC：除數；之負值；DE：商。

```

;
DIVIDE  LD    HL, (DIVDND)    ; 被除數取入 HL。
        LD    A, (DIVSOR)    ; 除數取入 A，
        NEG                     ; 且變號。
        LD    C, A            ; BC 存除數之
        LD    B, FFH          ; 負值。
        LD    DE, 0           ; 商先令為零。
LOOP    ADD    HL, BC          ; 被除數 - 除數。
        JR    NC, DONE        ; 不夠減就結束。
        INC    DE              ; 否則，商值加一。
        JP    LOOP            ; 繼續。
DONE    RET                     ; 完了，回返。

```

此一程式最快需 25 微秒之執行時間，而最差的情況則需 0.5 秒。平均約 1 毫秒左右。由此可見，以軟體程式作除算之速度實在太慢。

了。

注意，於例 7-11 之程式，被除數減除數是以被除數加上除數之 2 補數達成。由於結果置於 HL，故除數之負數必須取入十六位元之 DE，然後使用十六位元加法指令相加。

以上所舉之方法，由於當被除數不夠減時，除數不須再加回減算所得結果，因此，可稱為**不復原法**（non-restoring method）。除算有另一種方法稱為**復原法**（restoring method）。於復原法，程式進行一系列的試減（被除數減除數），若夠減，則商加 1；否則，若不夠減（即減算所得結果為負），除數再加回減算結果，以恢復原有被減數值，同時，商值亦扣除一。

復原法

圖 7-17 所示即為以復原法作八位元二進除算之演算法流程圖。一開始，商 = 0，移位計數器 = 8。緊接，被除數與商數同時左移一

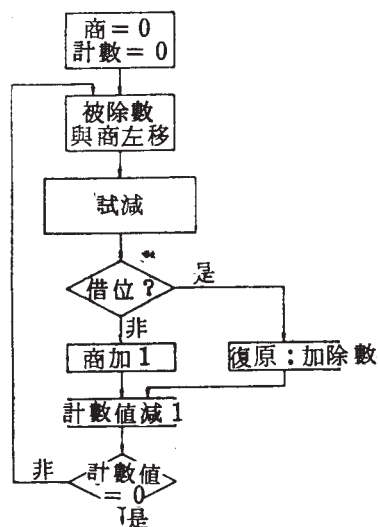


圖 7-17 復原法之八位元二進除算流程圖

位。被除數減除數。若夠減（無借位），則商加一；否則，若不夠減，除數再加回減算餘數。之後，計數值減一，若計數值不為零，控制跳回第二步驟，繼續移位與試減。

茲舉一例說明之。

例 7-12 復原法二進除算（16/8）

；該副程式以復原法，求出一十六位元數目除以一八位元數目之商。
；暫存器之使用安排為——HL：被除數 / 商；B：除數；D：計數
；器。控制進入副程式時，被除數與除數必須已分別儲存於 HL 與 B
；暫存器。

```

DIV 168  LD      C, 0          ; 除數已存於 BC。
          LD      D, 8        ; 反複次數 = 8。
LOOP     ADD     HL, HL       ; 被除數 / 商
          XOR     A           ; 進位旗號清除為零。
          SBC     HL, BC
          INC     HL          ; 商數進 1。
          JP      P, JUMP1    ; 若結果為正，則略過。
          ADD     HL, BC      ; 餘數復原。
          RES     0, L        ; 商數位元清除為 0。
JUMP1    DEC     D           ; 迴路計數值減 1。
          JR      NZ, LOOP    ; 8 次未完時繼續。
DONE     RET                ; 完了回返。
  
```

於上述復原法除算，被除數置於 HL 暫存器對，除數置於 BC 暫存器對（事實上僅 B 暫存器，C 存 0）。標題 LOOP 起之迴路，每次將 HL 之內含左移一位，清除進位，然後將 HL 中之部份被除數或餘數，減去 B 暫存器之除數（SBC HL, BC），且商數位元設定為

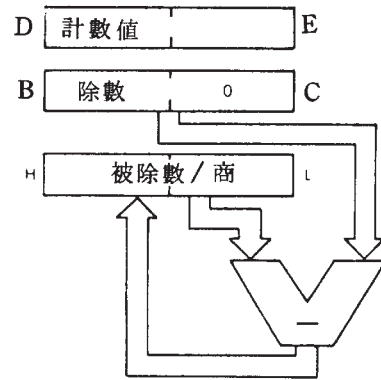


圖 7-18 例 7-12 之十六位元對八位元除算的暫存器使用安排

1 (INC HL)。若剛剛之減算夠減(得到正結果)，則控制直接跳至標題 JUMP1 之指令，將迴路計數值減 1，準備下一次除算。倘若剛剛之減算不夠減，則 JP P, JUMP1 跳越不成，微處理器繼續執行次一緊接指令。ADD HL, BC 將除數加回部份被除數，使其恢復減算前原有值，且將商數位元清除為 0 (RES 0, L)。

注意，HL 暫存器對同時履行了兩種功能，其除了於開始時存放被除數，且於除算過程中存放部份被除數(或稱餘數)外。每次左移後所空出之最低次位元(L 暫存器之第 0 位元)，正巧可儲存商數位元。而且最巧妙之處是，雖然 L 暫存器儲存了商數結果，但 SBC HL, BC 之減算並不破壞商數結果，而只將部份被除數減去除數。因為，C 暫存器一直為 0。

例 7-13 所示則為另一復原法除算之例子。該例題執行一十六位元除以十五位元之除算，CPU 暫存器之安排如圖 7-19 所示。

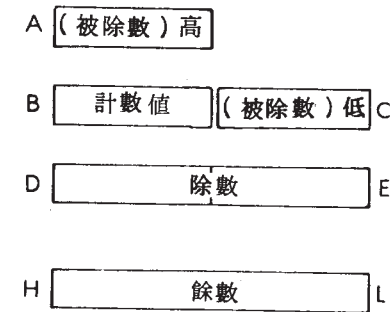


圖 7-19 例 7-13 除算程式之資料安排

例 7-13 復原法二進除算 (16/15)

；該副程式以復原法，求取一十六位元數目除以一十五位元數目之商；。控制進入副程式之前，被除數必須存於 AC，而除數(不可為 0；)存於 DE。控制回返後，AC 含商，HL 含餘數。

```

DIV16  LD    HL, 0           ; 累加器清除為零。
        LD    B, 16          ; 迴路計數初值 = 16。
L00P16  RL     C              ; 累加器—結果左旋轉。
        RLA
        ADC   HL, HL         ; 左移一位。
        SBC   HL, DE         ; 試減除數。
        JR    NC, $ + 3      ; 夠減則跳走。
        ADD   HL, DE         ; 否則，累加器值復原。
        CCF                  ; 求得結果位元。
        DJNZ  L00P16        ; 未達 16 次則繼續。
        RL    C              ; 移入最後結果位元。
        RLA

```

RET ; 回返

注意：“\$”代表現有程式計數器之內含。

有關此一問題之不復原法除算，由於手續過於複雜且不經濟，故此處不加以討論。

7-6 比較運算

比較運算在功能上類似於減算，兩者唯一的區別是：比較的結果並不保留，唯一受影響的只有旗號。因此，比較指令可用以測試某一運算元對另一運算元之關係，並且，緊接之條件跳越指令可根據比較之結果，使控制跳至不同之地方，進行不同之處理。正如加或減算一般，八位元之比較指令能使用多種定址法，包括暫存器間接與索引定址。

比較指令常見之應用有：(1)排序：找出一系列數目之最大或最小值。(2)多途分叉：根據兩數之大小關係，令控制分別跳至不同之位置。以及(3)搜尋：自一系列資料項目中找到所要的一個。下面，我們分別舉一例說明之。

例 7-14 所示為自一系列**正數**中，找出其最小值之程式例題。

例 7-14 找出正數表列之最小值

；該程式利用比較運算，找出一正數表列之最小值。表列儲存於位址；LIST開始之連續緊接記憶位置。表列尾端之資料項目-1，為表；列結束之標記。最小值儲存於B暫存器，索引暫存器IX為表列指；示器。

；

```
GTSMLL  LDX      IX, LIST ; 取入表列起始位址。
          LD       C, -1    ; 取入結束標記。
          LD       B, 127    ; 起始最小數 = 127。
NEXT     LD       A, (IX)   ; 取入次一資料項目。
```

```
CP       C              ; 表列完了嗎？
JR       Z, DONE        ; 若是，則結束。
INC      IX             ; 先指至次一項目。
CP       B              ; 取入值與原最小數比。
JP       P, NEXT        ; 若新數大，則繼續。
LD       B, A           ; 否則，取入值令為
                          ; 新最小值。
JP       NEXT           ; 繼續比。
DONE     HALT           ; 結束。
LIST     DEFB          20
          DEFB          32
          DEFB          1
          DEFB          0
          DEFB          37
          DEFB          112
          DEFB          3
          DEFB          -1 ; 終止標記。
          END
```

於例 7-14，表列資料儲存於位址LIST開始之連續緊接記憶位置。由於表列僅含正數，故表列末端之記憶位置儲存-1，以作為表列終止之標記。

程式一開始，表列起始位址LIST取入索引暫存器IX。終止記號“-1”存入C暫存器，同時，B暫存器儲存起始假想最小值127。起始作業結束。

標題NEXT以下為比較迴路。LD A, (IX)指令取入一新元素，CP C指令測試其值是否為-1，若是，則顯示表列已完，程式可結束。若非，指示器值先加一，然後CP B指令將A中之新取

入值與B中之現有最小值比。若新取入值大，則控制跳回NEXT處，繼續拿取次一元素。否則，LD B, A 指令將新取入值令為新最小值。程式然後繼續迴路。

注意，第一，由於表列均含正數，故以-1作表列終止標記，比起設迴路計數器，將其值遞減，再測試其是否為零之方法容易多了。第二，由於八位元2補數所能表示之最大正數為127，故程式一開始即將最小值令為此值，以確保其不致小於表列之所有元素。第三，若欲求最大值則僅需將B之起始值定為0，且JP P, NEXT 指令改成JP M, NEXT即可。

上述例子為絕對值或無號數比較，有號數比較如何做呢？有號數的比較有四種情況發生：正正，正負，負正，或負負相比。若兩數同號，則在累加器A之內含大於或等於另一運算元時，比較運算一定會產生進位。若兩數異號，則當累加器A內含小於零時，會有進位產生。下面的程式即將累加器A與B暫存器之內含相比。兩數中之任一者均可為-128至+127之任意數。根據兩數比較結果（累加器A內含大於，等於，或小於B暫存器內含）程式控制分別跳至GREAT, EQUAL, 與LESST等記憶位置。

例7-15 兩數之算術比較

；下述指令系列將累加器A與B暫存器所含之兩數目，作算術的相比；。根據累加器內含是否大於、等於、或小於B暫存器內含，程式控制；分別跳至GREAT, EQUAL, 與LESST等位置。

；

```

CMPARE    CP      B          ; A比B
           JP      Z, EQUAL   ; 相等就跳走。
           PUSH    AF         ; A與旗號存起。
           XOR     B          ; 看兩數是否同號。
           JP      P, SAME    ; 若是，跳至SAME。

```

```

POP        AF                ; 異號。取回A值。
TEST      JP      C, LESST   ; A < B。
           JP      GREAT     ; A > B。
POP        AF                ; 取回A與旗號值。
CCF                          ; 進位旗號反相。
JP         TEST

```

上述之程式一開始即測試A與B之值是否相等。若是，則控制跳至EQUAL之位置。然後，A與旗號值存入堆疊器。XOR 指令測試兩數是否同號。若XOR後，累加器之符號位元為1，則表原來兩數必有一正一負（符號位元一為0，另一為1）。否則，若兩數同號，則A與旗號值取回。進位旗號（代表先前比較之結果）被反相，以測試。若同號數相比且無進位，或異號數相比且有進位，則控制轉移至LESST。反之，控制則轉移至GREAT。

除求最大值與最小值外，比較運算之常見應用尚有搜尋（search）——自表列中找出想要之元素，與判斷某一輸入數字值之範圍等。茲舉一例說明之。

下面的程式利用索引定址以及比較指令，搜尋一表列，看其是否含文數字“*”。

例7-16 以比較指令作表列搜尋

；該副程式利用索引定址及比較指令，搜尋一表列之資料是否含文數字“*”。若找到，則控制回返時，累加器含0，且索引暫存器；IX含元素之記憶位址。若找不到，則程式控制回返時，累加器含；-1。表列之起始位址為BASE。表列長度為COUNT。控制在轉；移至此副程式前。表列資料與表列長度值假設皆已定義好。

；


```

SEARCH  LD      IX, BASE
        LD      A, "*"
        LD      B, COUNT

TEST    CP      (IX)
        JP      Z, FOUND
        INC     IX
        DEC     B
        JP      NZ, TEST

NOTFND  LD      A, -1
        RET

FOUND   LD      A, 0
        RET

```

注意，這個程式有兩個“出口”。若搜尋成功（即表列含已知元素），則控制最後由第二個 RET 處離開副程式。否則，若搜尋不成，控制最後則由第一個 RET 處離開副程式。此種多出口程式是完全合法，且可正常動作的。

```

A=00 BC=0000 DE=0000 HL=0000 S=0300 P=0100 0100' LD  BC,(0200)
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0200')
A=00 BC=0003 DE=0000 HL=0000 S=0300 P=0104 0104' LD  R,0B
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
A=00 BC=0803 DE=0000 HL=0000 S=0300 P=0106 0106' LD  DE,(0202)
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0202')
A=00 BC=0803 DE=0005 HL=0000 S=0300 P=010A 010A' LD  D,00
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
A=00 BC=0803 DE=0005 HL=0000 S=0300 P=010C 010C' LD  HL,0000
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0000')
A=00 BC=0803 DE=0005 HL=0000 S=0300 P=010F 010F' SRL  C
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
C A=00 BC=0801 DE=0005 HL=0000 S=0300 P=0111 0111' JR  NC,0114
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0114')
C A=00 BC=0801 DE=0005 HL=0000 S=0300 P=0113 0113' ADD  HL,DE
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
A=00 BC=0801 DE=0005 HL=0005 S=0300 P=0114 0114' SLA  E
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
V A=00 BC=0801 DE=000A HL=0005 S=0300 P=0116 0116' RL  D
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V A=00 BC=0801 DE=000A HL=0005 S=0300 P=0118 0118' DEC  B
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
N A=00 BC=0701 DE=000A HL=0005 S=0300 P=0119 0119' JP  NZ,010F
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (010F')
N A=00 BC=0701 DE=000A HL=0005 S=0300 P=010F 010F' SRL  C
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V C A=00 BC=0700 DE=000A HL=0005 S=0300 P=0111 0111' JR  NC,0114
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0114')
Z V C A=00 BC=0700 DE=000A HL=0005 S=0300 P=0113 0113' ADD  HL,BF
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V A=00 BC=0700 DE=000A HL=000F S=0300 P=0114 0114' SLA  E
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
V A=00 BC=0700 DE=0014 HL=000F S=0300 P=0116 0116' RL  D
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V A=00 BC=0700 DE=0014 HL=000F S=0300 P=0118 0118' DEC  B
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
N A=00 BC=0600 DE=0014 HL=000F S=0300 P=0119 0119' JP  NZ,010F
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (010F')
N A=00 BC=0600 DE=0014 HL=000F S=0300 P=010F 010F' SRL  C
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V A=00 BC=0600 DE=0014 HL=000F S=0300 P=0111 0111' JR  NC,0114
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0114')
Z V A=00 BC=0600 DE=0014 HL=000F S=0300 P=0114 0114' SLA  E
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
V A=00 BC=0600 DE=002B HL=000F S=0300 P=0116 0116' RL  D
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V A=00 BC=0600 DE=002B HL=000F S=0300 P=0118 0118' DEC  B
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
N A=00 BC=0500 DE=002B HL=000F S=0300 P=0119 0119' JP  NZ,010F
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (010F')
N A=00 BC=0500 DE=002B HL=000F S=0300 P=010F 010F' SRL  C
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V A=00 BC=0500 DE=002B HL=000F S=0300 P=0111 0111' JR  NC,0114
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0114')
Z V A=00 BC=0500 DE=002B HL=000F S=0300 P=0114 0114' SLA  E
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
V A=00 BC=0500 DE=0050 HL=000F S=0300 P=0116 0116' RL  D
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V A=00 BC=0500 DE=0050 HL=000F S=0300 P=0118 0118' DEC  B
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
N A=00 BC=0400 DE=0050 HL=000F S=0300 P=0119 0119' JP  NZ,010F
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (010F')
N A=00 BC=0400 DE=0050 HL=000F S=0300 P=010F 010F' SRL  C
A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00

```

Multiplication: A Complete Trace

第 8 章

```

Z V  A=00 BC=0400 DE=0050 HL=000F S=0300 P=0111 0111' JR  NC,0114
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0114')
Z V  A=00 BC=0400 DE=0050 HL=000F S=0300 P=0114 0114' SLA  E
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
S V  A=00 BC=0400 DE=00A0 HL=000F S=0300 P=0116 0116' RL  D
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V  A=00 BC=0400 DE=00A0 HL=000F S=0300 P=0118 0118' DEC  B
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
N    A=00 BC=0300 DE=00A0 HL=000F S=0300 P=0119 0119' JF   NZ,010F
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (010F')
N    A=00 BC=0300 DE=00A0 HL=000F S=0300 P=010F 010F' SRL  C
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V  A=00 BC=0300 DE=00A0 HL=000F S=0300 P=0111 0111' JR  NC,0114
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0114')
Z V  A=00 BC=0300 DE=00A0 HL=000F S=0300 P=0114 0114' SLA  E
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
C    A=00 BC=0300 DE=0040 HL=000F S=0300 P=0116 0116' RL  D
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
      A=00 BC=0300 DE=0140 HL=000F S=0300 P=0118 0118' DEC  B
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
N    A=00 BC=0200 DE=0140 HL=000F S=0300 P=0119 0119' JF   NZ,010F
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (010F')
N    A=00 BC=0200 DE=0140 HL=000F S=0300 P=010F 010F' SRL  C
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V  A=00 BC=0200 DE=0140 HL=000F S=0300 P=0111 0111' JR  NC,0114
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0114')
Z V  A=00 BC=0200 DE=0140 HL=000F S=0300 P=0114 0114' SLA  E
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
S    A=00 BC=0200 DE=0180 HL=000F S=0300 P=0116 0116' RL  D
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
      A=00 BC=0200 DE=0280 HL=000F S=0300 P=0118 0118' DEC  B
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
N    A=00 BC=0100 DE=0280 HL=000F S=0300 P=0119 0119' JF   NZ,010F
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (010F')
N    A=00 BC=0100 DE=0280 HL=000F S=0300 P=010F 010F' SRL  C
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V  A=00 BC=0100 DE=0280 HL=000F S=0300 P=0111 0111' JR  NC,0114
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0114')
Z V  A=00 BC=0100 DE=0280 HL=000F S=0300 P=0114 0114' SLA  E
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z V C A=00 BC=0100 DE=0200 HL=000F S=0300 P=0116 0116' RL  D
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
V    A=00 BC=0100 DE=0500 HL=000F S=0300 P=0118 0118' DEC  B
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00
Z N  A=00 BC=0000 DE=0500 HL=000F S=0300 P=0119 0119' JF   NZ,010F
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (010F')
Z N  A=00 BC=0000 DE=0500 HL=000F S=0300 P=011C 011C' LD   (0204),HL
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00 (0204')
Z N  A=00 BC=0000 DE=0500 HL=000F S=0300 P=011F 011F' NOP
      A'=00 B'=0000 D'=0000 H'=0000 X=0000 Y=0000 I=00

```

Multiplication: A Complete Trace (continued)

移位、旋轉、與位元運作

本章，我們介紹移位、旋轉、與位元運作指令之常見應用。

8-1 邏輯移位

邏輯移位可能是最容易了解之移位。八位元之邏輯移位不顧慮正負號，資料字組每次被左移或右移一位。如前所提過的，資料位元並不循環，被移出之位元恒進入進位旗號，而另一端空出之位元則補0。Z80有兩個邏輯移位運算指令：SRL（邏輯右移）與SLA（算術左移）。後者雖然名為“算術”左移，但實際上却為邏輯左移。留意，Z80之所有移位指令均對八位元資料作業。

邏輯移位指令之主要功用有二：

- 1 將運算元乘2（邏輯左移）、除2（邏輯右移）、或乘除其它倍數。
- 2 調整資料字組某些位元之位置。

圖8~1所示即為邏輯移位指令最簡單之應用：將運算元值乘2與除2。如圖所示，運算元53H（=83₁₀）經邏輯右移一次後，變成了29H（=41₁₀），同時，進位旗號值（代表餘數）為1。若將53H邏輯左移一次，則所得結果即為原來之兩倍——A6H（=166₁₀）。此時，進位旗號值為0，代表所得結果尚未越過256₁₀。

Z80 微電腦軟體硬體

```

MUL 10  LD    A,(NUMBER)  ; 取入運算元。
        SLA    A           ; 乘 2。
        LD     B, A        ; 兩倍結果存起。
        SLA    A           ; 再乘 2 (四倍)。
        SLA    A           ; 再乘 2 (八倍了)。
        ADD    A, B        ; 加上兩倍結果。
        RET

```

雖然旋轉與罩蓋運算有時亦可用以重整資料位元之位置，但大多數情況下還是使用邏輯移位運算。例 8～3 之程式即為藉用邏輯移位運算，將兩十六進數字轉換成其對等 ASCII 值之一種方式。

例 8～3 求兩十六進數字之 ASCII 碼

；該副程式分別以兩種不同之方式，求出兩十六進數字之 ASCII ；值。第一數字使用邏輯移位之方式，第 2 數字則採用罩蓋法。兩十 ；六進數字之濃縮二進值來自於位址 VALUE 之記憶位置。其 AS ；CII 值則分別儲存於位址 BUF 與 BUF+1 之記憶位置。

```

CVERT  LD     A,(VALUE)    ; 取得兩十六進數字。
        LD     B, A        ; 並複製一份存起。
        SRL    A           ; 使第一數字之二進
        SRL    A           ; 值靠右。
        SRL    A
        SRL    A
        ADD    A, 30H      ; 轉換成 ASCII 碼。
        CP     A, 3AH      ; 數字介乎 A 至 F 嗎？
        JP     M, OK1      ; 若非，則正確。
        ADD    A, 7        ; 若是，則再加 7。

```

```

OK1    LD     (BUF), A     ; 存起第一數字之結果。
        LD     A, B        ; 取入第 2 數字。
        AND    FH          ; 遮掉第 1 數字。
        ADD    A, 30H      ; 轉換成 ASCII。
        CP     A, 3AH      ; 數字介乎 A 至 F 嗎？
        JP     M, OK2      ; 若非，則結果正確。
        ADD    A, 7        ; 若是，則必須加 7。
OK2    LD     (BUF+1), A   ; 存起第 2 數字之結果。
        RET              ; 完了。

```

注意，若數字為 0 至 9 中任一者，則其二進值只要加 30H，即為 ASCII 值（例如，數字“9”之七位元 ASCII 值即為 39H）。但若數字為 A 至 F 任一者，則就必須加 37H。

有時，利用加算可達成邏輯左移之效果。於 Z80，累加器 A 或 HL 暫存器對之內含皆可自加，此一運算之結果即等於將運算元值乘 2。由於 ADD A, A 指令僅長一位元組，且需時 1 微秒之執行時間，而 SLA 指令長兩位元組，且需 2 微秒之執行時間。因此，不用說，誰也知道應該採用加法指令。例 8～4 之程式，即使用 ADD HL, HL 指令，將一十六位元數值乘以 10 倍之例子。

例 8～4 十六位元數值乘以十

；該副程式使用十六位元之自加指令，將位址 NUMBER（低次） ；與 NUMBER+1 兩記憶位置所存之十六位元數值，乘以十倍。 ；結果留於 HL 暫存器對。

```

SMUL10 LD     HL,(NUMBER) ; 取入運算元。
        ADD    HL, HL      ; 乘 2。
        PUSH   HL         ; 兩倍結果存至 DE。

```

POP	DE	
ADD	HL, HL	; 乘 2 (四倍了)
ADD	HL, HL	; 乘 2 (已八倍)
ADD	HL, DE	; 再加兩倍結果。
RET		; 剛好十倍。

8-2 旋轉型移位

Z80 有六個旋轉型移位 (亦即旋轉) 指令, 其中有些是重複的。前面曾提過, Z80 之旋轉指令有八位元旋轉與九位元旋轉兩種。前者僅將運算元本身之八位元旋轉, 後者則將進位當成暫存器或記憶位置之延伸, 併入旋轉。於八位元旋轉, 被移出之位元同時進入進位旗號。圖4~9與4~10所示即為八位元旋轉之情形。

旋轉移位主要用以調整資料字組之位元位置以便存取及儲存, 或使多暫存器之移位成為可能。CPU暫存器或記憶位置之資料可以旋轉移位加以測試, 而不需破壞原來之資料。

儘管以 AND A 或 OR A 能很輕易地獲得一八位元運算元之極性, 但下面我們看如何以旋轉運算計算一來自記憶位置之運算元的極性。

例 8~5 計算某一記憶運算元之極性

; 該副程式計算位址 MEMOP 記憶位置所含之八位元資料的極性。
; 若資料為偶極性, 則累加器 A 之內含為 0。否則, 若奇極性, 則累
; 加器內含為 1。

;

PARITY	XOR	A	; 清除極性與進位。
	LD	B, 8	; 設定計數值。
	LD	HL, MEMOP	; 取入記憶運算元位址。
LOOP	RLC	(HL)	; 移出一位元至進位。

	JR	NC, JUMP1	; 若非 1, 則略過。
	XOR	1	; 若 1, 則極性反態一次。
JUMP1	DEC	B	; 計數值減 1。
	JR	NZ, LOOP	; 非零則繼續。
	RET		

首先, XOR A 將 A 暫存器與進位同時清除為零。然後, RLC (HL) 指令被執行八次。每執行一次, 記憶運算元就有一個位元被移入進位, 若該位元為 1, 累加器 A 之最低次位元則反態一次。因此, 旋轉八次後, 若運算元所含之 "1" 位元的總個數為偶數 (即偶極性), 則累加器之內含必為零 (因其原先為零); 否則, 若運算元為奇極性, 則累加器內含必為 1。旋轉八次後, 記憶運算元回復原來之狀態, 保持不變。

下面為旋轉應用之另一例子。例 8~6 使用旋轉指令, 將一八位元二進數轉換成八個二進 ASCII 數字。每次運算元右移一位, 若被移出之位元為 0, 程式就產生一數字 "0" 之 ASCII 碼 (30H); 若被移出之位元為 1, 程式則產生一數字 "1" 之 ASCII 碼 (31H)。八個產生之 ASCII 碼則儲存於位址 BUF 起之連續記憶位置。由於運算元置於累加器 A, 故右旋轉指令可採用 RRC

A 或 RRCA。兩者之功用相同 (RRCA 主要是與 8080 相吻合的)。不過, 由於 RRC A 佔兩位元組, 且需 2 微秒之執行時間, 而 RRCA 僅長一位元組, 且需 1 微秒之執行時間。因此, 當然採用 RRCA。

例 8~6 二進運算元轉換成二進 ASCII 數字

; 此副程式根據位址 BYTE 之記憶位置的內含, 產生八個二進 ASCII 數字碼。BYTE 位置之內含每次被右旋轉一位, 若被移出
; 位元為 0, 則程式產生數字 "0" 之 ASCII 碼——30H; 若

；被移出位元為 1，則程式產生數字“1”之 ASCII 碼——31
 ；H。八個產生之 ASCII 碼分別儲存於位址 BUF+7 至 BUF
 ；之記憶位置。
 ；

```

BXASB EQU $
        LD  A, (BYTE)    ; 取入欲轉換字組。
        LD  C, 8          ; 設定位元計數器。
        LD  IX, BUF+7     ; 緩衝區最後位元組。
LOOP    LD  B, 30H        ; ASCII 0。
        RRCA              ; 移出一位元。
        JR  NC, JUMP1     ; 若位元為 0，跳越。
        INC B             ; 否則，改成 ASCII 1。
JUMP1   LD  (IX), B       ; 存起 ASCII 數字。
        DEC IX
        DEC C             ; 計數值減 1。
        JR  NZ, LOOP
        RET
  
```

於例 8~6，若 BYTE 之記憶內含為 10111001₂，則程式所產生之 ASCII 文數字碼將分別為 31, 30, 31, 31, 31, 30, 30, 31（皆為十六進制），儲存於位址 BUF, BUF+1, ..., BUF+7 等之記憶位置。

旋轉運算可與邏輯移位並用，以達成多段（倍長）移位。假設我們欲將位址 UDGE 之記憶位置起的三位元組運算元，以邏輯移位左移兩個位元位置。例 8~7 所示即是以旋轉運算，使進位連續傳遞過三個位元組之情形。

例 8~7 三位元組資料邏輯左移兩位元位置

；此副程式將位於位址 UDGE 處之三位元組資料，邏輯左移兩個
 ；位元位置。資料最高次與最低次位元組分別儲存於位址 UDGE 與
 ；UDGE+2 之記憶位置。
 ；

```

SLUDGE EQU $
        LD  IX, UDGE+2
        SLA (IX)          ; 0 → b0, b7 → c
        RL  (IX-1)        ; c → b0, b7 → c
        RL  (IX-2)        ; c → b0, b7 → c
        SLA (IX)          ; 0 → b0, b7 → c
        RL  (IX-1)        ; c → b0, b7 → c
        RL  (IX-2)        ; c → b0, b7 → c
        RET
  
```

特別注意，**多段移位**的方法是於**最低次位元組**使用**邏輯移位**，而其它**較高次位元組**使用**旋轉**指令，合併達成。例 8~8 即為例 8~7 之另一變樣，該副程式使用 HL 暫存器對。

例 8~8 例 8~7 之另一變樣

；該副程式以 HL 與 IY 兩十六位元暫存器，重作例 8~7。
 ；

```

SLUDGE LD  HL, (UDGE+1)    ; 取入兩低次位元組。
        LD  IY, UDGE       ; 取入最高次位元組。
        ADD HL, HL         ; 左移一位。
        RL  (IY)           ; 進位移入高次位元組。
        ADD HL, HL         ; 再左移一位。
        RL  (IY)
  
```

LD (UDGE+1), HL ; 存回低次兩位元組。
RET

8-3 算術移位

算術左移於本章第一節已經介紹過了。於所有製造廠商間，算術左移通常有兩種做法。多數令算術左移等於邏輯左移——符號位元於第一次左移時即被移出運算元。另一種作法則在左移的過程中保留符號位元。每次移位時，第 6 位元被移出，進入進位旗號，而第 7 位元（即符號位元）保持不動。

無論如何，**算術右移**就無所模擬兩可了！每次右移時，符號位元不僅保持不變，同時亦被複製入第 6 位元。此種移位即可用於**有號數之算術運算**，因為，經移位後，正數仍為正數，負數仍為負數。圖 8-2 所示即為以算術右移，將某一負數每次除 2 的情形。正如圖中所

1 0 0 0 1 0 0 1	起始值 = -119_{10}
1 1 0 0 0 1 0 0	第一次 S R A 後 = -60_{10}
1 1 1 0 0 0 1 0	第二次 S R A 後 = -30_{10}
1 1 1 1 0 0 0 1	第三次 S R A 後 = -15_{10}
1 1 1 1 1 0 0 0	第四次 S R A 後 = -8_{10}

圖 8-2 算術右移運算

示的，每次算術右移時，運算元值同時被四捨五入。事實上，右移後

進位之值即代表除 2 運算之餘數。若算術右移除 2 運算後小數部份欲留置，則被移出之位元必須儲存於另一暫存器。

例 8-9 之副程式即對某一數目連作算術右移運算，並將小數部份存於 B 暫存器的情形。若二進小數點假設在兩暫存器之間，則小數部份之位元位置分別代表 $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, 等等。

例 8-9 連續右移並存起小數部份

；此副程式對某一數目連作算術右移三次，並將小數部份儲存於 B 暫存器。數目來自位址 NUMBER 之記憶位置。

；

```
ARSSVF LD A, (NUMBER) ; 取入數目。
        LD B, 0 ; 小數清除為零。
        SRA A ; 除 2。
        RR B ; 存起  $\frac{1}{8}$ 。
        SRA A ; 除 4。
        RR B ; 存起  $\frac{1}{4}$ 。
        SRA A ; 除 8。
        RR B ; 存起  $\frac{1}{2}$ 。
        PET
```

若忽略餘數，則**正數**經算術右移的結果，即相當於**商之整數部份留住，而小數部份被截掉**。譬如， $00001011_2 (=11_{10})$ 經右移兩次（除 4）後，數目值變為 $00000010_2 (=2_{10})$ ，餘數 $\frac{3}{4}$ 遺失。若忽視小數部份，則**負數**經算術右移的結果，**商被四捨五入了**。譬如， $11110101_2 (= -11_{10})$ 經除 4 後，數目變為 $11111101_2 (= -3_{10})$ 。小數部份若存起，則為 01000000_2 。商被四捨五入了。

由於算術右移運算展延了符號位元，因此，通常其無法用於重排資料位元。

8-4 BCD 數字移位

以上之所有移位運算每次均只移位一個位元位置，這節要介紹之 RLD 與 RRD 指令，則每次能將累加器 A 及 HL 暫存器對選取之記憶位置的內含，分別向左或向右移動**四個位元位置**。雖然此種移位能方便地用於任意四位元資料欄之處理，但其主要用途仍在 BCD 資料之處理。每一四位元移位可移出一 BCD 數字，並引入一新 BCD 數字。下面我們看一如何以此種移位處理 BCD 資料之例子。

例 8~10 之程式轉換 ASCII 數字（假設 0 至 9）為 BCD 數字。位址 INBUF 至 INBUF+9 之記憶位置儲存了十個剛由鍵盤輸入之 ASCII 數字，此些數字代表一由 0000000000₁₀ 至 9999999999₁₀ 間之十位 BCD 數。程式將十個數字之 ASCII 碼分別轉換成其對應之 BCD 數字，並以濃縮之形式儲存於位址 INBUF 至 INBUF+4 等五個連續記憶位置，每一位元組儲存兩個 BCD 數字，如圖 8~3 所示。此種轉換於微電腦應用上是極為普遍的。因為，由微電腦之輸入設備輸入之資料，均為 ASCII 碼形式（譬如，若程式設計者在鍵盤上打一“1”鍵，則微電腦將收到“1”數字鍵之 ASCII 碼：00110001），此些資料在能運算之前，必須先轉換成其它形式（如直接二進數或 BCD 數等）。

例 8~10 ASCII 數轉換成 BCD 數

；此副程式將位址 INBUF 至 INBUF+9 等記憶位置所儲存的
；十位 ASCII 數目（假設每一數字均介乎 0 至 9 之間），轉換成
；其對等之 BCD 數，並以濃縮 BCD 之形式，儲存於地址 INBUF
；至 INBUF+4 之記憶位置。
；

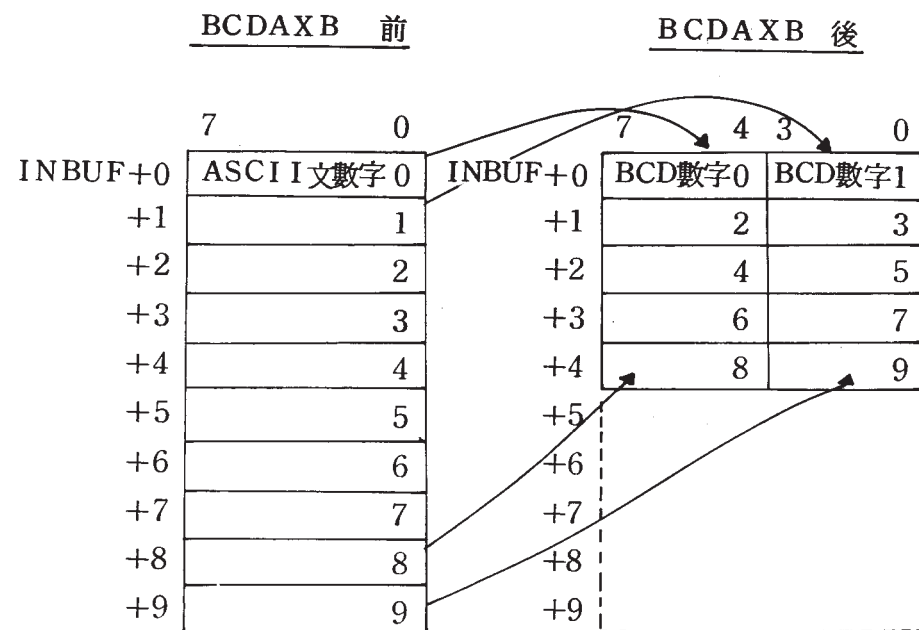


圖 8~3 ASCII 轉換成 BCD

BCDAXB	EQU	\$	；ASCII 換成 BCD。
	LD	IX, INBUF	；指至緩衝區。
	LD	HL, INBUF	；指至緩衝區。
	LD	B, 5	；設定計數器。
LOOP	LD	A, (IX)	；取入一位數字。
	SUB	A, 30H	；轉換成 BCD 0~9。
	RLD		；旋轉至 (HL)。
	LD	A, (IX+1)	；取入次一位數字。
	SUB	A, 30H	；換成 BCD 0~9。
	RLD		；移入 (HL) 位置。
			；一濃縮 BCD 已存妥。
	INC	IX	；指示器往前跳二位置。

```

INC  IX
INC  HL                      ; 結果指示器進一位置。
DEC  B
JR   NZ, LOOP
RET

```

於例 8~10 之例子，程式每經歷迴路一次，即拿取（並處理）兩個 ASCII 數字。每一數字經拿取後，即被轉換成 BCD（SUB A, 30H），然後，所得 BCD 數字即被移入 HL 所指之記憶位置。由於結果以濃縮 BCD 形式儲存（每一記憶位置儲存兩 BCD 數字），因此，迴路末了，HL 內含僅加 1，指至次一記憶位置。而 ASCII 數字之指示器 IX 的內含則加 2，以跳過兩個已處理過之數字。處理十個數字，迴路總共重複五次。

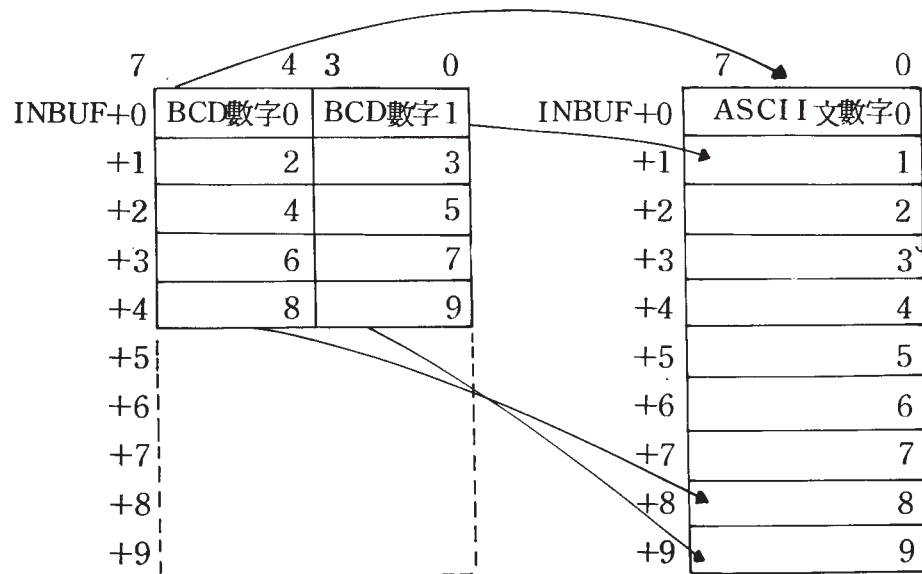


圖 8~4 BCD 轉換成 ASCII

作為四位元旋轉之另一應用例子，我們再看看相反之轉換——將位址 INBUF 至 INBUF+4 等記憶位置所儲存之濃縮 BCD 數，轉換成十個 0 至 9 之 ASCII 數字。如圖 8~4 所示。特別注意，此回必須倒過頭做，因為，每一位置之 BCD 數字轉換的結果，必須以兩個記憶位置加以儲存。此時，IX 將指至緩衝區次一欲儲存之位置，而 HL 指至目前之兩 BCD 數字。最初兩 BCD 數字取自位址 INBUF+4 之記憶位置，結果儲存於位址 INBUF+9 與 INBUF+8 兩記憶位置。迴路同樣執行五次。

例 8~11 濃縮 BCD 轉換成 ASCII

此副程式將位址 INBUF 至 INBUF+4 等記憶位置所儲存之濃縮 BCD 數，轉換成一十位數字之 ASCII 數目，儲存於位址 INBUF 至 INBUF+9 等記憶位置。轉換由位址最高之位置開始。

```

;
BXBCDA EQU $
LD  IX, INBUF+9 ; 指至緩衝區最後一位置。
LD  HL, INBUF+4 ; 指至最後兩數字。
LD  B, 5 ; 迴路計數為 5。
LOOP XOR A ; 清除累加器 A。
RRD ; 高位數字取入 A。
ADD A, 30H ; 轉換成 ASCII 碼。
LD  (IX), A ; 存至緩衝區。
XOR A ; 為低位數字重複上述步驟
RRD ; 。
ADD A, 30H
LD  (IX-1), A
DEC  IX ; 緩衝區指示器值減 2。

```

```

DEC    IX
DEC    HL          ; 數字指示器減 2。
DEC    B           ; 迴路計數減 1。
JR     NZ, LOOP    ; 未零時繼續。
RET

```

注意，程式迴路之最後兩個指令 DEC B 與 JR NZ, LOOP，於 Z 80 可以一特殊之 DJNZ LOOP 指令將之取代。例 8 ~ 11 程式中唯一要注意的是，在每一 BCD 數字取入前，累加器 A 必須先清除為零。因為，旋轉移入的動作並不影響 A 暫存器之高次四位元。

8-5 位元運作

第四章曾說過，Z 80 之位元運作指令，能測試、置定、或清除任一 CPU 暫存器或記憶位置之內含的任一位元值。並且對記憶器中之運算元能使用暫存器間接與索引兩種定址法。此些指令由於能達成快速且有效之位元運作，故甚為好用。固然併用取入與罩蓋運算亦能達成位元運作，但如以下所示的，此種方法却至少需三個以上之指令。

例 8 ~ 12 測試某一位元值

；下面指令系列以取入及罩蓋運算，測試位址 BYTE 之記憶內含的；某一位元值。該指令系列之功能等於 Z 80 之 BIT 指令。

```

;
BITEST EQU $          ; 位元測試。
LD      HL, BYTE       ; 指至欲測試位元組。
LD      A, (HL)         ; 位元組取入 A。
AND     A, VALUE        ; 遮掉其它位元，看測
                        ; 試位元值為 0 或 1。

```

```

VALUE EQU 1            ( 或 2, 4, 8, 16, 32, 64, 128,
                        ; 視欲測試的是第 0, 1, ..... 7 位
                        ; 元而定 )。

```

例 8 ~ 13 置定某一位元值

；下面指令系列以取入及邏輯 OR 運算，將位址 BYTE 位置之某位；元值置定為 1。該指令系列之功能即等於 Z 80 之 SET 指令。

```

;
BITSET EQU $           ; 位元置定。
LD      HL, BYTE       ; 指至位元組。
LD      A, (HL)         ; 取入位元組。
OR      A, VALUE        ; 置定某一位元。
LD      (HL), A         ; 結果存回原處。

```

```

VALUE EQU              ; VALUE 值視欲置定是那位元而定。
                        ; 其情形如測試之狀況。

```

例 8 ~ 14 清除某一記憶位元

；下面指令系列以取入與罩蓋運算，將位址 BYTE 位置之某一位元；值，清除為零。該指令系列之功能等於 Z 80 之 RESET 指令。

```

;
BITRST EQU $           ; 位元清除。
LD      HL, BYTE       ; 指至位元組。
LD      A, (HL)         ; 取入位元組。
AND     A, NOTVAL       ; 預定位元清除為零。
LD      (HL), A         ; 結果存回原處。

```

```

NOTVAL EQU FEH         ( 或 FD, FB, F7, EF, DF, BF,

```


7F，視欲清除的是第0，1，
...，7位元而定。)

以上之三個程式皆可分別以一等效Z80指令將之取代。譬如，若欲測試某一記憶位元組之第0位元值，則我們可寫

```
LD    HL, BYTE    ; 指至欲測試位元組。
BIT   0, (HL)      ; 測試第0位元值。
JR    Z, ZERO      ; 若零，則跳至ZERO。
ONE   ;             ; 若1，則繼續。
```

若記憶位元組之第5位元值欲設定為1，則可用

```
LD    IX, BYTE
SET   5, (IX)
```

若記憶位元組之第1位元值欲清除為零，則可用

```
LD    IY, BYTE
RES   1, (IY)
```

當然，BIT，SET，或RES任一指令亦皆可對任一CPU暫存器之任一位元運算。例如，

```
BIT   7, B ; 測試B暫存器之第7位元值。
```

為了舉例說明位元處理指令如何應用，下面我們看一例子。圖8～5所示為某一Z80微電腦系統所使用之256列×256行電視顯示幕。螢幕上之每一光點以一記憶位元值代表，並且可為亮（白）或暗（黑）。第一列之光點由位址VDTTB1至VDTTB1+31等記憶位置之內含代表。由於每一記憶位置可表示8個光點，故總共剛好256個光點。第二列之光點則由位址VDTTB1+32至VDTTB1+63，……，第256列由位址VDTTB1+8160至VDTTB1+8191之記憶位置內含代表等。全部8K位元組（64K位元）之

資料，則由電視顯示幕（VDT）之控制電路，以直接記憶器存取之方式，輸出至顯示幕。

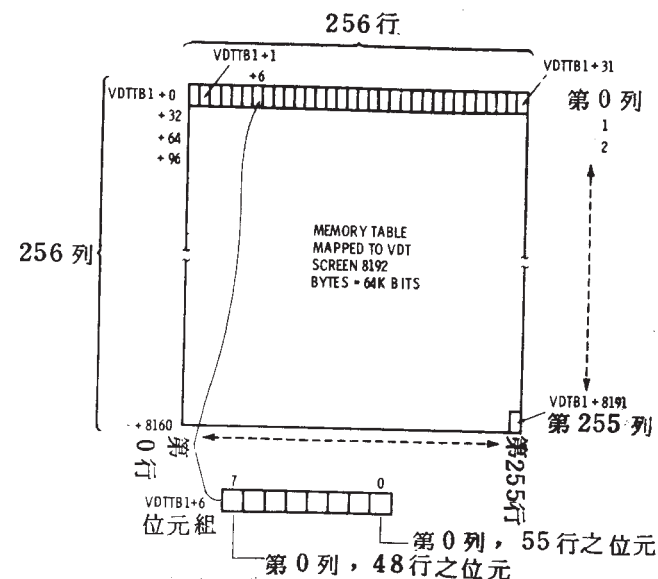


圖8～5 64K個光點之VDT位元分佈圖

顯示程式將隨時不斷地更新顯示緩衝區VDTTB1之記憶內含，以改變顯示圖案。為了方便起見，根據其所在之相對位置，我們賦予每一光點一個數對（列數，行數）。譬如，最左上角之光點即為（0，0）——第0列第0行，最右上角之光點為（0，255），最右下角之光點為（255，255）等。因此，現在的問題就是如何將光點之行列位址，轉換成其所對應之記憶位元的位址，並且自VDTTB1緩衝區拿取該位元，測試其值，置定其值，或清除其值，以改變顯示結果。例8～15之程式包括了此一程式之三部份。於程式中，資料以A與B兩暫存器傳至程式，A表列數，B表行數。置定（SETPX）與清除（RESTPX）程式主要將所需光點置定為1或清除為0。測

試程式 (TESTPX) 則以 Z 旗號送回一代表光點之現有值的真值
; Z = 0 代表亮 (白) , 而 Z = 1 代表暗 (黑) 。

例 8 ~ 15 電視螢幕顯示程式

```

; 此一常式測試光點之值。
;
TESTPX  LD      C, 46H
JUMP    CALL    GTADD
        OR      A, C          ; 合成位元數。
        LD      (MODIFY), A   ; 欲測試之位元數存入
INSTRU  BIT      0, (HL)      ; BIT 指令第 2 位元組。
        RTN                ; 回返。
MODIFY  EQU      INSTR+1
;
; 此一常式將光點之值置定為 1。
;
SETPX   LD      C, C6H
        JP      JUMP
;
; 此一常式將光點之值清除為 0。
;
RESTPX  LD      C, 86H
        JP      JUMP
;
; 此一常式將含光點之位元組資料的記憶位址置於 HL 暫存器對，位
; 元數置於累加器 A 之第 3 至 5 位元。
;

```

```

GTADD   PUSH    BC          ; BC 內含存入堆疊器。
        SRL     A           ; 去掉 B 之低次三位元。
        RR      B           ; 調整三個位址位元。
        SRL     A
        RR      B
        SRL     A
        RR      B
        LD      L, B        ; 位移存入 HL。
        LD      H, A
        ADD     HL, VDTTB1
        POP     BC          ; 取回 BC。
        LD      A, B        ; 取入低次位元。
        CPL
        AND     7           ; 獲得位元位址。
        SLA     A           ; 為測試指令，調整位元
        SLA     A           ; 位址之位置。
        SLA     A
        RET                ; 回返。

```

上述之常式 (副程式) 或許比至目前為止我們所看過的任一程式困難許多。首先，它們皆是真正可以 CALL 指令叫用之副程式。回返以 RTN 指令達成。整個副程式總共有三個入口處 (entry point)，測試光點值之 TESTPX，將光點值置定為 1 之 SETPX，以及將光點值清除為 0 之 RESTPX。此些可分別以類似於

```

LD      A, ROW
LD      B, COL
CALL    SETPX
:

```

之指令系列加以叫用。

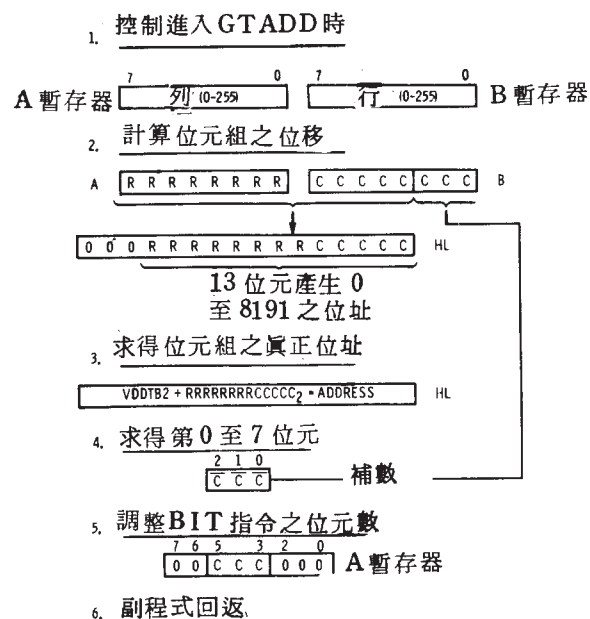


圖 8~6 GTADD 副程式之動作

TESTPX 副程式又叫用另一副程式 GTADD。GTADD 副程式將光點之行位址，轉換成一位元組位址存於 HL，以及一位元位址（於位元組內之第幾位元）存於 A。此地，我們先詳細探討 GTADD 副程式。

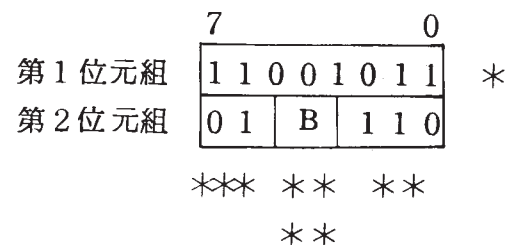
輸入至 GTADD 副程式之參數如圖 8~6 所示。列數存於累加器 A，行數存於暫存器 B。含光點位元之位元組距 VDDTB1 表格起點之位移為（列數×32）+（行數÷8）。位元位址則恰巧等於（行數÷8）之餘數。GTADD 計算位元組位移之方法是將 A 與 B 移位，然後將位移加上表格之起始位址，以獲得位元組之實際位址。該位址緊接儲存於 HL 暫存器對。光點之位元位址則經由遮掉 B 暫存器之

最低次三位元求得。由於結果恰為 Z80 位元位置數之反號，因之，結果又被反相（CPL）一次。位元位置數求得後，被調整至 A 暫存器之第 3 至第 5 位元（理由在下面即可分曉）。

副程式 TESTPX 之功用在於測試光點位元值。當被叫用時，A 與 B 暫存器分別含光點之列數與行數。副程式之第一個動作是將一對應於 BIT 指令第二元組之數值取入 C 暫存器。46H 代表 BIT 0，（HL）指令之第二元組。副程式然後又叫用 GATADD 副程式，算出光點在 VDDTB1 表格內之位元組位址與位元位址。回返時，A 暫存器內之位元位址與 C 暫存器所存之 BIT 0，HL 指令的第 2 位元組互相 OR。以決定 BIT 指令該測試的是位元組內之第幾位元。此一結果最後被存入 BIT 指令之第二元組（MODIFY 之位置）。BIT 指令然後被執行，結果置於零值旗號 Z。當控制由 TESTPX 副程式回返至叫用程式時，Z 旗號值仍然有效。

SETPX 與 RESETPX 兩副程式之功用則類似於 TESTPX，唯一的差別是取入 C 暫存器的乃 SET 0，（HL）與 RES 0，（HL）之第二元組。一自 GTADD 副程式回返後，代表欲測試位元之位元碼（B 欄）即以 OR 運算鑄入指令。圖 8~7 所示即為由三個不同入口點進入副程式時，在標題 INSTRU 所執行之指令的情形。

BIT B，（HL）

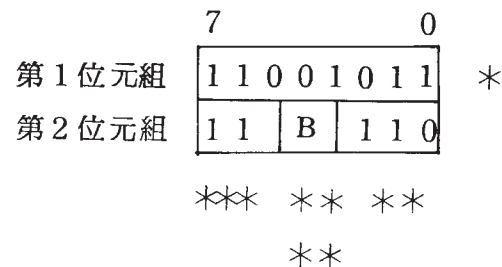


註解：

*，** 表該欄在三個指令皆相同。

SET B, (HL)

*** 表該欄隨各指令而異。



** 表該欄之值於
** GTADD 副程式
中求得。

RES B, (HL)

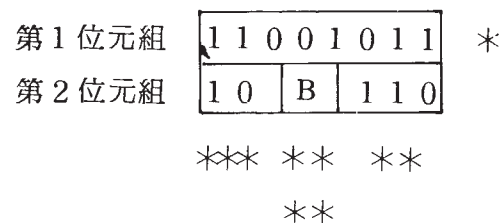


圖 8 ~ 7 VDT 位元常式中之指令修飾

例 8 ~ 15 之程式廣泛使用了移位、位元運作、索引、與邏輯等各種功能。同時，其亦介紹了許多新的觀念。其中最重要的是說明了，指令亦是一種“資料”，其能如修飾其它資料般地被修改。標題 INSTRU 處之指令於程式執行時被動態修飾，而非於組譯時刻。

至此，我們已介紹了 Z80-CPU 之所有移位、旋轉、與位元運作指令，以及其應用。事實上，移位與旋轉指令尚有一極為重要之應用。那就是用以構成軟體之乘算與除算程式。由於這在前一章已提過，故此處不再重複。

文件名稱： Z80 微電腦軟體硬體第 5、6、7、8 章(掃描版)

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	2007-02-09	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。
-----------------	---

Documents Digitize Service Center 製作
1998-2007

文件分類	I
文件編號	00028
文件批號	04

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