

第 12 章

Z80 CPU之界面信號與時序

在看過 Z80 微處理器之內部“資源”，與了解如何以之設計程式，以解決各式各樣的問題後。緊接，我們走出 Z80 微處理器晶片，看看於一微電腦系統中，其與系統其它組件之關係，以及一完整微電腦系統之全貌。這就好比走入深林，沿其道路，看看其內部所含樹木之種類與每一樹木之長相後，再走出深林，搭直昇機，在空中俯瞰樹林之全貌以及整個樹林之出入口與其座向。

首先，第一個步驟，我們先對 Z80 微處理器與外界溝通之孔道作一番了解，並認識此每一孔道之功能。

I、界面信號

Z80 微處理器為一雙列型包裝之 40 支接腳積體電路，其接腳情形依功能分組，示於圖 12 - 1。下面，我們分別介紹此些每一接腳的功能。

12-1 位址與資料巴士

位址巴士由 A0 至 A15 等信號線組成，其中，A15 表最高次位元，而 A0 表最低次位元。A0 至 A15 為高電位動作，且為三態輸出。亦即，位址巴士不動作時，其輸出為高阻狀態。整體位址巴士線代表

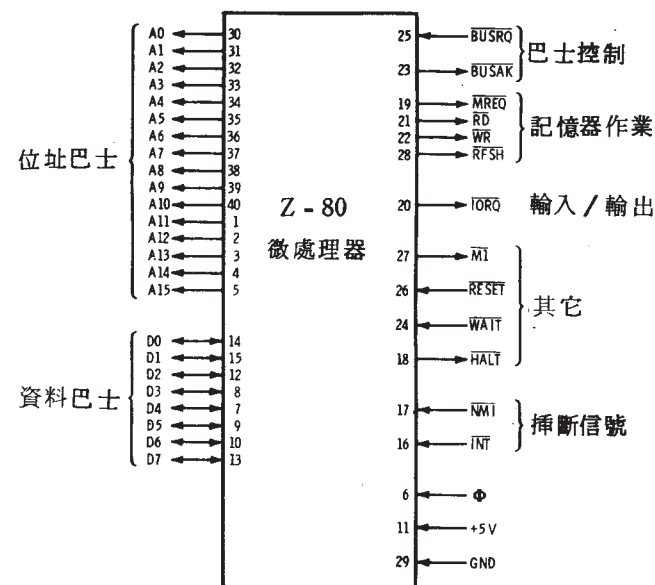


圖 12-1 Z80-CPU 之界面信號

一十六位元之記憶或設備位址。由於十六位元能產生 2^{16} 種不同組合狀態，因之，Z80 微處理器能直接選取 $2^{16} = 65536_{10}$ （或 64 K）個記憶與設備位置。若 Z80 CPU 所選取的是輸入 / 輸出設備，則位址巴士之低次八位元（A0 ~ A7）代表一輸入 / 輸出設備位址，該位址之值為 0 至 255。除記憶或輸入 / 輸出設備位址外，位址巴士之低次七位元所傳送的有時是記憶復新暫存器 R 之內含。此刻即為記憶復新之期間。

D0 ~ D7 之信號線則構成八位元之資料巴士。D0 表最低次位元，D7 表最高次位元。此些信號線為三態，高電位動作，並且為雙向。換言之，透過資料巴士，資料可傳入或傳出 Z80 微處理器。

12-2 巴士控制信號

Z 80 微處理器有兩控制信號與位址巴士及資料巴士有關。它們是輸入信號 $\overline{\text{BUSRQ}}$ ，以及輸出信號 $\overline{\text{BUSAk}}$ 。 $\overline{\text{BUSRQ}}$ （巴士請求）信號為一低電位動作信號，外部設備產生此一信號，要求 Z 80 微處理器放開對系統巴士之控制，以便其能獲致系統巴士之控制權。 $\overline{\text{BUSRQ}}$ 線動作時，Z 80 微處理器只要結束現有之機器週期（machine cycle），即會立刻將系統巴士（位址巴士、資料巴士、與三態輸出控制信號）置於高阻抗狀態。

外部設備在獲致巴士控制權後，很可能進行一直接記憶體存取（Direct Memory Access，簡記為 DMA）作業。DMA 使外部設備能直接伸及記憶體，使資料能直接來回傳遞於記憶體與輸入／輸出設備間，而不需經手微處理器。為了避免發生衝突（CPU 亦同時欲存取記憶體），於 DMA 作業期間，CPU 必須被“鎖住”。

外部設備在對 Z 80 微處理器提出巴士請求（使 $\overline{\text{BUSRQ}}$ 線為邏輯 0）後，Z 80 微處理器以巴士認知（BUS Acknowledge）信號 $\overline{\text{BUSAk}}$ 回答外部設備。 $\overline{\text{BUSAk}}$ 為一低電位動作輸出信號。於接受巴士請求後，Z 80 微處理器使 $\overline{\text{BUSAk}}$ 線輸出低電位，以告知外部設備，系統巴士已處於高阻抗狀態，外部設備可開始用之進行 DMA 作業。

12-3 記憶體控制信號

Z 80 有四個控制信號與記憶體之作業有關： $\overline{\text{MREQ}}$ ， $\overline{\text{RD}}$ ， $\overline{\text{WR}}$ ，以及 $\overline{\text{RFSH}}$ 。

首先，記憶請求信號， $\overline{\text{MREQ}}$ ，為一三態、低電位動作之輸出信號。此信號顯示目前位址巴士上所持之位址，為有效的記憶體讀取或寫入位址。事實上，該信號主要是記憶體之晶片致能（chip enable）信號的一部份。於記憶體讀取時，該信號用以令記憶體輸

出一項資料，而於記憶體寫入時，該信號通知記憶體輸入一項資料。

記憶體讀取信號 $\overline{\text{RD}}$ 與記憶體寫入信號 $\overline{\text{WR}}$ ，兩者均為三態、低電位動作之輸出信號。此兩信號分別用以告知外部記憶體，Z 80 微處理器所欲作的是記憶體讀取或記憶體寫入。在 $\overline{\text{MREQ}}$ 信號變為低電位時，於機器週期內， $\overline{\text{RD}}$ 或 $\overline{\text{WR}}$ 信號亦必有一者會變為低電位。若 $\overline{\text{MREQ}}$ 與 $\overline{\text{RD}}$ 同時為低電位，則表 Z 80 CPU 欲做記憶體讀取。若 $\overline{\text{MREQ}}$ 與 $\overline{\text{WR}}$ 同時為低電位，則表 Z 80 CPU 欲做記憶體寫入。不論記憶體讀取或寫入，位址巴士均傳遞位址，而資料巴士傳送資料。

記憶復新信號 $\overline{\text{RFSH}}$ 與正常之記憶體作業並不相干。其僅用於系統使用動態記憶體之時。為了保持其原有之記憶內含，動態記憶體必須不斷地反復被復新——記憶槽之記憶內含被讀取。典型的動態記憶體均具有一記憶復新信號，以及五至六條位址線之輸入。六條位址輸入線表示，於 2 毫秒之復新週期內，必須做完 64（ 2^6 ）次記憶復新。當 $\overline{\text{RFSH}}$ 與 $\overline{\text{MREQ}}$ 兩輸出信號同時為低電位時，表示 Z 80 微處理器正在進行記憶復新作業，外部動態記憶體會利用位址巴士之低次七位元，進行一次記憶復新。微處理器每作一次指令拿取， $\overline{\text{RFSH}}$ 線即動作一次，同時，由於每次指令拿取後，R 暫存器之內含即自動加一，是以，位址線將不斷地反映次一復新週期所需的新位址。就上述所舉六條位址輸入線之例子而言，復新整個動態記憶體則需費時 64 個指令週期（1 指令週期即為 Z 80 CPU 執行一個指令所需的時間）。若以每個指令週期平均 4 微秒計，則總共需時 256 微秒（0.256 毫秒），即能將整個動態記憶體全部復新一次。

12-4 輸入／輸出信號

輸入／輸出請求 $\overline{\text{IORQ}}$ 信號為一三態、低電位動作之輸出信號。此信號顯示 Z 80 微處理器正在進行輸入／輸出作業。當 $\overline{\text{IORQ}}$ 信號變低電位時，位址巴士之低次八位元（A0～A7）則代表一輸入／輸出設備之位址。由 $\overline{\text{RD}}$ 與 $\overline{\text{WR}}$ 信號，該位址所選定之輸入／輸出設

備即可知 Z 80 究竟欲作讀取或寫入。當 Z 80 微處理器認可 (acknowledge) 一插斷時， $\overline{\text{IORQ}}$ 信號會與 $\overline{\text{M1}}$ 信號 (下面立即討論) 同時產生，以顯示此時插斷向量可置於資料巴士。特別注意，雖然插斷認可作業發生於 $\overline{\text{M1}}$ 期間，但輸入 / 輸出作業却不能。

12-5 其他 CPU 控制信號

第一機器週期 信號， $\overline{\text{M1}}$ ，為一低電位動作之輸出信號。該信號顯示 Z 80 微處理器刻在進行指令運算碼之拿取。不像 8080，Z 80 有許多運算碼為兩位元組之指令，在此種情況下，兩次運算碼之拿取 (Z 80 CPU 每次僅能拿取八位元之運算碼)， $\overline{\text{M1}}$ 信號皆會輸出低電位。(於 Z 80，兩位元組之運算碼皆以 CB，DD，ED，或 FD 開頭) $\overline{\text{M1}}$ 信號亦會與 $\overline{\text{IORQ}}$ 信號同時產生，以顯示插斷認可週期。

重置 信號 RESET 為一低電位動作之輸入信號，該信號用以令 Z 80 微處理器重置——回復至一最原始狀態。每當電源一打開，或系統之重置按鈕被按下時，此一信號線即變成低電位，將 Z 80 微處理器重置。重置產生下列之結果：

- 1 程式計數器內含為零， $\text{PC} = 0000\text{H}$ 。
- 2 插斷致能正反器失效，禁止 $\overline{\text{NMI}}$ 以外之任何插斷。
- 3 插斷向量暫存器內含為零， $\text{I} = 00\text{H}$ 。
- 4 記憶復新暫存器內含為零， $\text{R} = 00\text{H}$ 。
- 5 插斷型態處於第 0 作業型態。
- 6 位址巴士與資料巴士變成高阻抗狀態，輸出控制信號變成不動作狀態。

等待 信號， $\overline{\text{WAIT}}$ ，與緩慢的記憶器或輸入 / 輸出設備有關。此為一低電位之輸入信號。被選定之記憶或輸入 / 輸出設備，以此一信號告知 Z 80 微處理器，其尚未及準備好作資訊傳輸。只要該信號一直保持低電位，Z 80 微處理器即會一直處於“等待”狀態，什麼事都不

不做。 $\overline{\text{WAIT}}$ 信號使得反應速度緩慢之記憶器或輸入 / 輸出設備，能界面至 Z 80 微處理器，並與之取得同步。

暫停 信號 $\overline{\text{HALT}}$ 為一低電位動作之 Z 80 輸出信號。該信號顯示 Z 80 微處理器目前正處於停止狀態。停止狀態乃由 Z 80 CPU 執行一 $\overline{\text{HALT}}$ 指令所引起。一般而言，程式在兩種情況下會使用 $\overline{\text{HALT}}$ 指令。第一，程式已作完所有運算而必須終止。第二，停止狀態已產生，程式正在等待插斷發生。當 CPU 正處於停止狀態時，其執行無運算 (NOP) 指令，以確保記憶復新作業能綿延不斷。

12-6 與插斷有關之信號

其餘之控制信號與插斷處理有關。**不可罩蓋插斷** (nonmaskable interrupt) 信號， $\overline{\text{NMI}}$ ，為一負向緣觸發之輸入信號。當此一輸入線變至低電位時，微處理器一執行完刻在執行之指令，即會立即接受此一插斷，Z 80 微處理器在接受 $\overline{\text{NMI}}$ 插斷時，產生了下列動作：

- 1 現有程式計數器之內含值推入記憶堆疊器存起。
- 2 然後控制轉移至位址 0066H 之記憶位置。換言之，位址 0066H 起之記憶位置，必須存放不可罩蓋插斷之處理常式。

不可罩蓋插斷不可被禁能，並且必須恒被 CPU 所認可。唯一的例外是， $\overline{\text{BUSRQ}}$ 信號比 $\overline{\text{NMI}}$ 信號具有更高優先。此外，連續的等待 ($\overline{\text{WAIT}}$) 狀態將使目前之指令無法結束，致使 Z 80 微處理器無法認可 $\overline{\text{NMI}}$ 插斷。

主要之**插斷請求**信號， $\overline{\text{INT}}$ ，亦為一低電位動作之輸入信號。當需要 Z 80 微處理器之服務時，外部設備以此一信號對 Z 80 提出插斷請求。倘若 Z 80 CPU 內之插斷致能正反器 IFF 被置為 1，並且 $\overline{\text{BUSRQ}}$ 信號不動作，則 CPU 在執行完目前之指令後，會立即認知 $\overline{\text{INT}}$ 信號。若上述條件滿足，則 CPU 會接受此一插斷，並於次一指令之 $\overline{\text{M1}}$ 期間，送出一 $\overline{\text{IORQ}}$ 信號告知外部設備，其已認可此一插斷請求信號。由於輸入 / 輸出作業之 $\overline{\text{IORQ}}$ 信號從不在 $\overline{\text{M1}}$ 期間發生，

因此，外部設備會將此一情況（ $\overline{M1}$ 與 \overline{IORQ} ）視為插斷認知。插斷處理之詳情在後面會有更詳盡之討論。

12－7 Z80 CPU之電特性

圖 12-2 所示為 Z80 微處理器之電特性。為了便利界面，所有輸入與輸出均具有 TTL 吻合性。電源供給僅需一單一正 5 伏特。Z80 微處理器單獨所需之最大電流為 200 毫安。不像 8080，Z80 只需一單相時序輸入，並且為 TTL 準位。最原始 Z80 之時序為 2.5MHZ，不過，目前已有 4 MHZ 之 Z80 微處理器問世。其它動態參數之詳細規格，請見附錄 A。

I、CPU之時序

微處理器執行一個指令所需之時間，稱為一個**指令週期**（instruction cycle）。（因之，指令週期為可變。）如圖 12-3 所示，Z80 微處理器之每一指令週期，可細分成若干基本週期。此些週期有兩種型式。最基本的稱為**T週期**（T cycle）。T週期即為**時序週期**。若 Z80 使用的是頻率 4MHZ 之時序，則每一 T週期固定為 250 ns（時序頻率之倒數）之時間。數個 T週期又組成一較大的**機器週期**（machine cycle），或僅稱**M週期**。一個機器週期即為微處理器完成一記憶器（或輸入／輸出設備）讀取或寫入所需之時間。

基本上，Z80 之每一機器週期包括了 3 至 6 個 T週期，而每一指令週期則包括 1 至 6 個機器週期。不過，每一指令週期之第 1 個機器週期（稱為**M1週期**），至少一定包括 4 個 T週期。圖 12-4 所示即為一具有三個機器週期（M1，M2 及 M3）之指令週期的情形。如圖所示，第一個機器週期時，Z80 微處理器拿取指令之運算碼（該指令之運算碼僅有八位元）。除非被等待信號所加長，否則，M1 週期必含 4，5，或 6 個 T週期。於第 2 與第 3 機器週期時，Z80 微處理器分別作記憶器（或輸入／輸出設備）之讀取與寫入作業。除非為

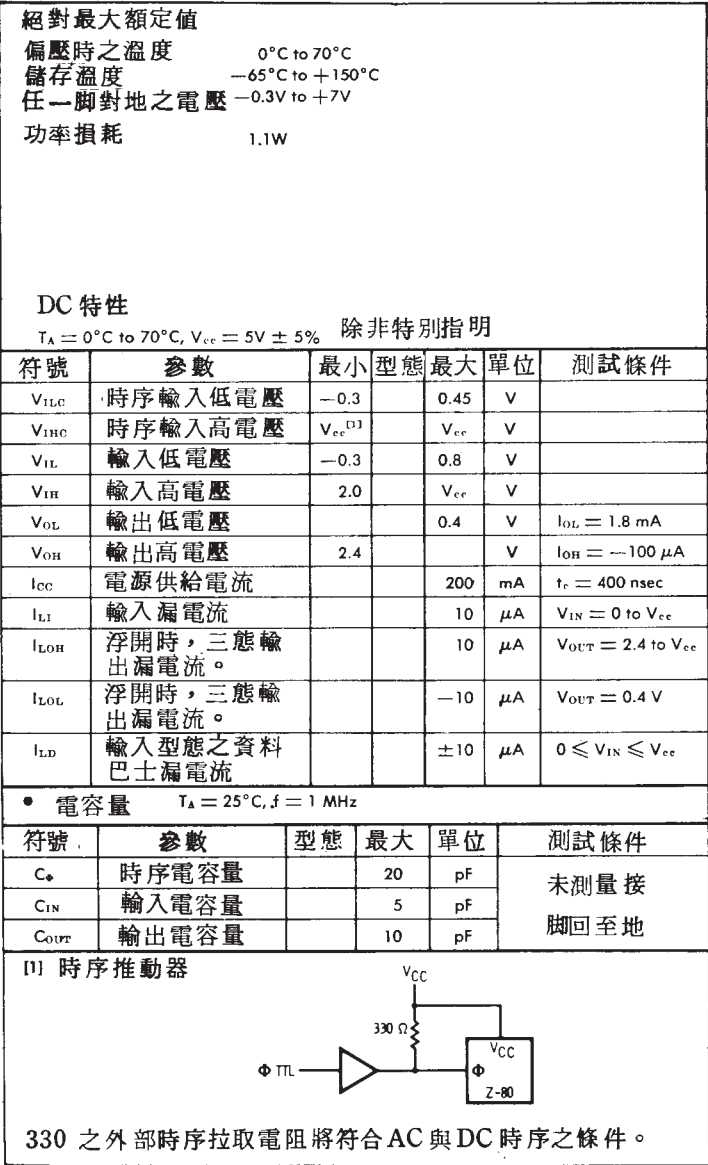


圖 12-2 Z80 微處理器之電特性

等待狀態所加長，否則，此些週期通常包括 3 至 5 個 T 週期。

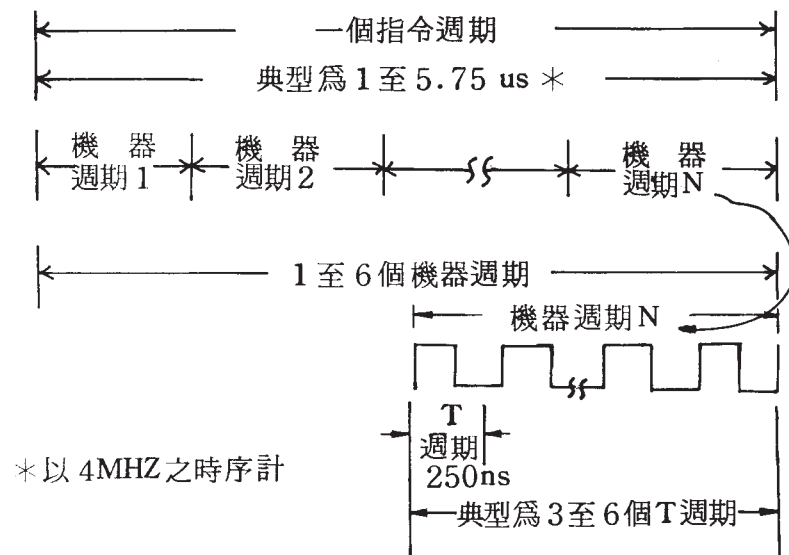


圖 12-3 Z 80 之指令週期

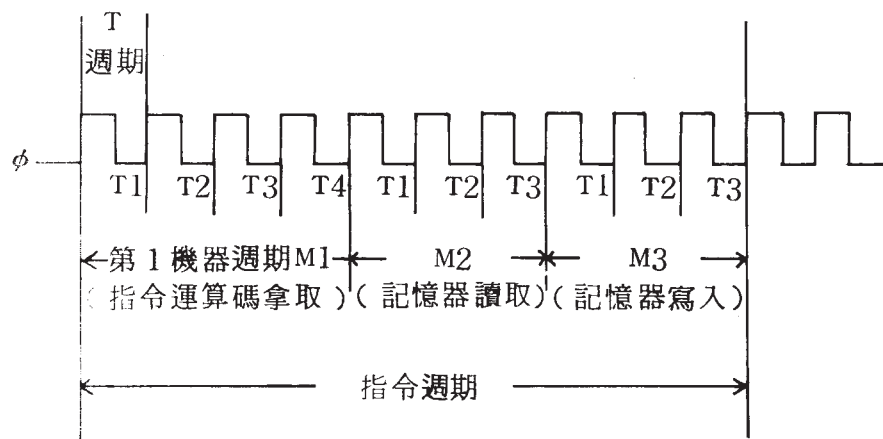


圖 12-4 具有三個機器週期之指令週期

Z 80 微處理器之作業總共包括七種機器週期：

1. 運算碼拿取週期（M1 週期）。
2. 記憶資料讀取或寫入週期。
3. 輸入 / 輸出讀取或寫入週期。
4. 巴士請求 / 認可週期。
5. 插斷請求 / 認可週期。
6. 不可罩蓋插斷請求 / 認可週期。
7. 脫離暫停（HALT）狀態。

茲將此七種週期之時序分別描述於后。

12-8 M1 週期

每一指令之執行均包括一運算碼拿取週期，或 M1 週期。於 Z 80，有少數指令之運算碼為兩個位元組長，因此，此些指令之執行有兩個 M1 週期。於 M1 週期內，Z 80 微處理器自記憶體讀取運算碼位元組，將之解碼，並且履行運算碼所代表之運算的部份或全部作業。圖 12-5 所示即為 INC R 指令之時序圖，該圖可用以舉例說明 M1 週期。INC R 指令只需一個機器週期，即能完全執行，總共需要 4 個 T 週期。

CPU 進入 M1 週期， $\overline{M1}$ 信號即掉至低電位，以顯示此一週期已開始。程式計數器之內含然後置於位址巴士，準備拿取次一指令之運算碼。於 T1 下降緣， \overline{MREQ} 與 \overline{RD} 兩信號一致掉至低電位，以告知外部記憶體，目前位址巴士上為有效之記憶位址。除非記憶體為本章爾後所討論之緩慢記憶體，否則，T3 上升緣之前，記憶體會將被選定之記憶位置的內含，置於資料巴士上。於 T3 上升緣，資料巴士上之運算碼位元組即被取入 Z 80 微處理器。隨即， $\overline{M1}$ ， \overline{RD} ，與 \overline{MREQ} 信號一致恢復至不動作狀態。M1 所剩的兩個 T 週期則用以作外部動態記憶器的復新。 \overline{RFSH} 降低至低電位，並且 \overline{MREQ} 再度動作，以告知外部動態記憶體，記憶復新可開始進行。此時，位址巴士之低

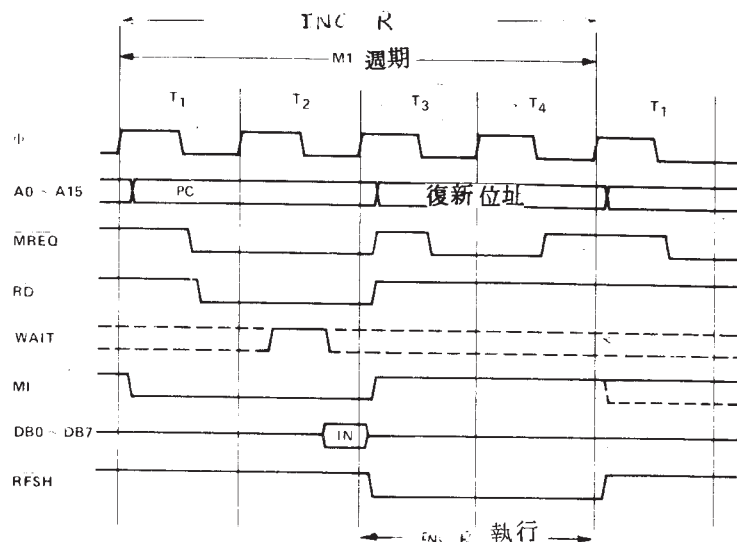


圖 12 - 5 M1 (運算碼拿取) 週期

次七位元含記憶復新暫存器 R 之內含。注意，此時 \overline{RD} 信號並不動作，以防止記憶復新之資料再進入資料巴士。於記憶復新期間， \overline{MREQ} 信號用以作復新讀取。由於復新位址唯有當 \overline{MREQ} 期間方能到達穩定狀態，因此， \overline{RFSH} 信號無法單獨使用。

在 M1 之最後兩 T 週期時間，Z80 微處理器解碼且執行所拿取之指令，亦即 INC R。INC R 指令拿取某一特定一般用途暫存器（A、B、C、D、E、H 或 L、或其補救），將之加一，然後置回原暫存器。由於不需再作任何記憶器存取，並且 CPU 暫存器之存取可輕易在幾百 ns 內完成，因之，該指令之執行不再需要任何額外之機器週期。

在收到 CPU 送來之 \overline{MREQ} 與 \overline{RD} 信號後，記憶器若覺自己無法及時將 CPU 所需之資料送出，則其可令 WAIT 線動作。CPU 然後會偵測出此一狀況，並進入“等待”狀態。圖 12-6 所示即為一具等待狀態之 M1 週期。於 T2 以及每一緊接 T_w 期間，CPU 於 ϕ 下降緣時讀取 WAIT 線之狀態。若此時 WAIT 線動作，則 CPU 於次一週期到來時，自動增加一等待 T 週期（ T_w ），繼續進入等待狀態。透過

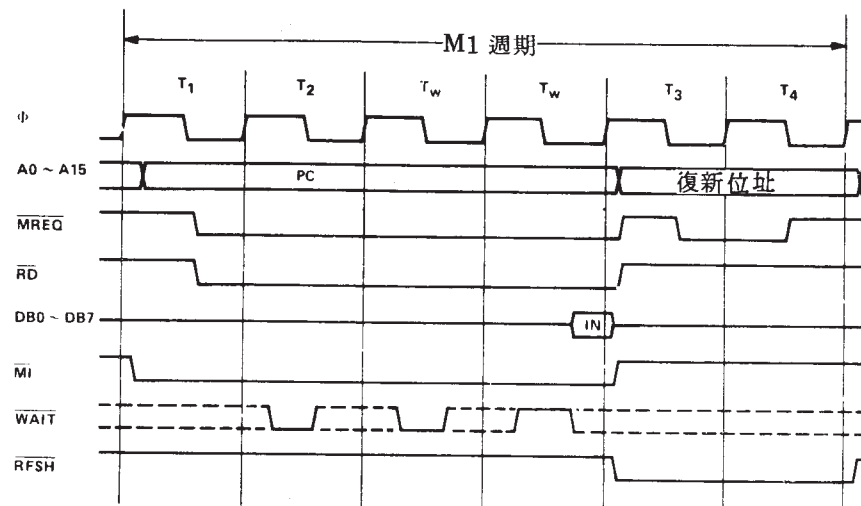


圖 12-6 具有等待狀態之 M1 週期

此一技巧，讀取週期可予增長，以配合各種存取時間之不同記憶器。

12-9 記憶器資料讀取與寫入週期

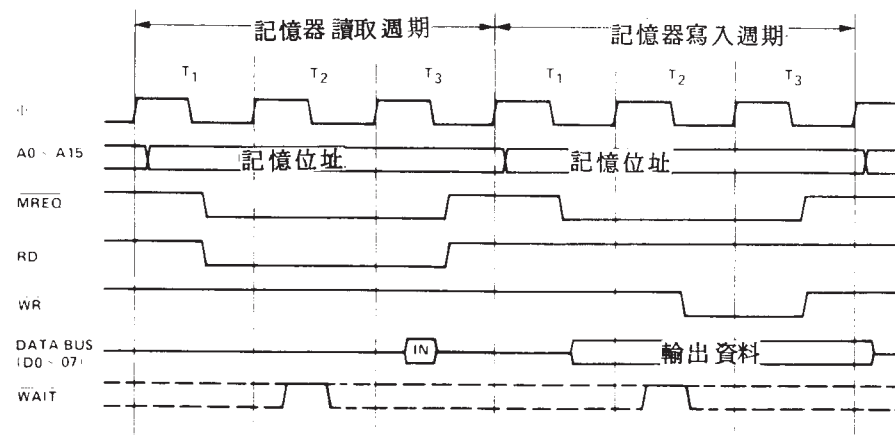


圖 12-7 記憶器讀取或寫入週期

圖 12-7 所示為 Z 80 異於 M1 週期之記憶體讀取或寫入週期的時序情形。除非有等待狀態插入，否則，此些週期均長三個時序週期。於記憶體讀取週期， $\overline{\text{MREQ}}$ 與 $\overline{\text{RD}}$ 信號之用法同於拿取週期。於記憶體寫入週期，當位址巴士上之位址資料達穩定時， $\overline{\text{MREQ}}$ 亦動作，致其能直接作為動態記憶體之晶片選擇信號。同樣地，當資料巴士上之資料達穩定時， $\overline{\text{WR}}$ 線亦動作，致使其能直接作為任何型態半導體記憶體之讀 / 寫脈衝。此外，於位址與資料巴士內含改變前半個 T 週期時間， $\overline{\text{WR}}$ 信號變成不動作，以便能滿足各型半導體記憶體之重疊條件。

下面，我們舉兩個實際之例子，說明記憶體之讀取週期與寫入週期。圖 12-8 所示即為將 HL 暫存器對所指之記憶位置的內含，取入 Z 80 微處理器內任一般用途暫存器（在指令以上 R 代替）的執行情形。M1 週期如同先前所討論過的。於 M1 末了，CPU 已將指令解碼，並開始一記憶體讀取週期，以便由記憶體獲得八位元之運算元。位址巴士， $\overline{\text{MREQ}}$ ，及 $\overline{\text{RD}}$ 信號之動作情形如同 M1 週期。此時，位址巴士所含為 HL 暫存器對之內含。T3 下降緣時，記憶體置於資料巴士上

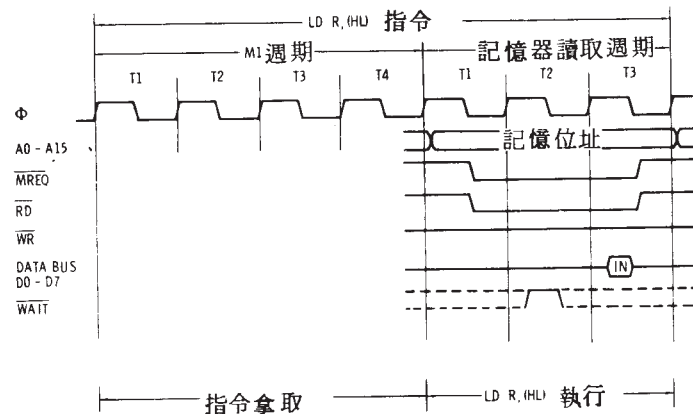


圖 12-8 記憶體讀取週期

之資料被鎖入 CPU，存入指定之暫存器。

圖 12-9 所示則為一記憶體寫入之情形。此時，假設 Z 80 所執行的是，將某一特定 CPU 暫存器之內含，存至 HL 暫存器對所指之記憶

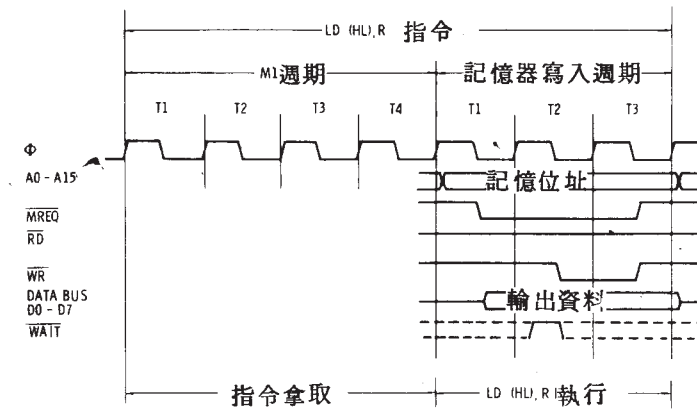


圖 12-9 記憶體寫入週期

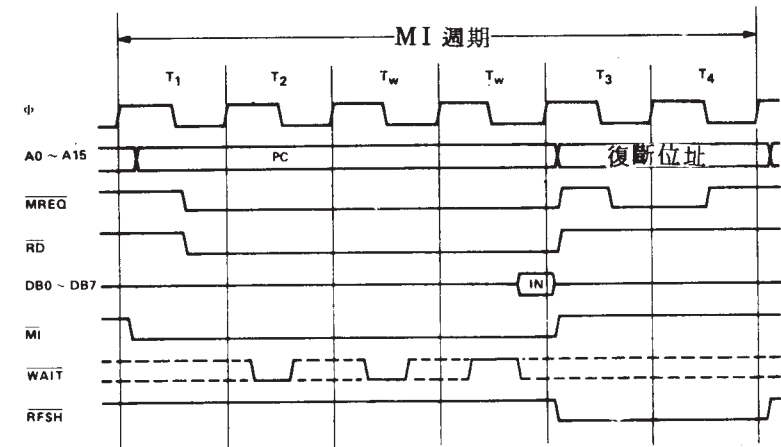


圖 12-10 附有等待狀態之記憶體讀取 / 寫入週期

位置之 LD (HL), R 指令。位址巴士輸出及 $\overline{\text{MREQ}}$ 信號之動作情形同前。T1 下降緣後，CPU 暫存器之內含被置於資料巴士。該資料一直停留於資料巴士，並且，T2 下降緣時， $\overline{\text{WR}}$ 信號線動作。當 $\overline{\text{MREQ}}$ 及 $\overline{\text{WR}}$ 信號同時動作之際，記憶器依位址巴士上之位址，將資料巴士上之資料寫入被選定之記憶位置。

若記憶器提出等待請求，則記憶器讀取或寫入週期將被延長。圖 12-10 所示即為具有等待狀態之記憶器讀取與寫入的時序情形。

12-10 輸入與輸出週期

輸入或輸出週期發生於 Z 80 微處理器執行一輸入或輸出指令時。輸入與輸出指令通常為 3 至 4 個機器週期長，包括 10 至 20 個 T 週期。不過，就最高能傳輸 256 個位元組之輸入 / 輸出區塊搬運指令 (INIR, INDR, OTIR, OTDR) 而言，機器週期必須重複至所有資料搬完為止。因之，此時指令之全部執行時間則需視所傳輸之位元組數，以及輸入 / 輸出設備之速度而定。圖 12-11 與圖 12-12 所示即分別為輸入週期與輸出週期之情形。

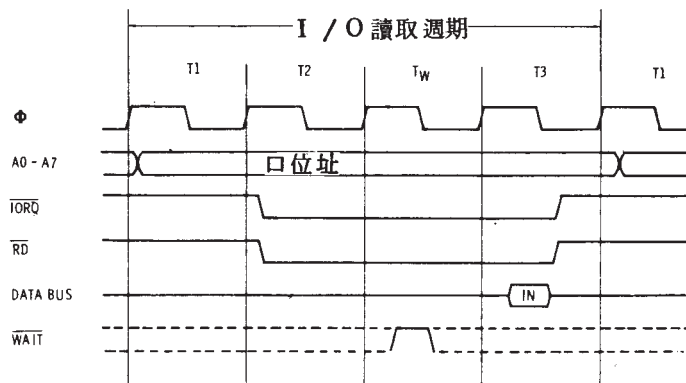


圖 12-11 輸入 / 輸出讀取週期

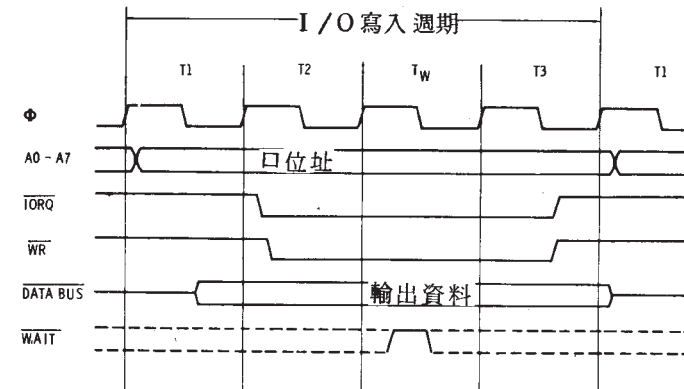


圖 12-12 輸入 / 輸出寫入週期

機器週期開始時，輸入 / 輸出設備之位址被置於位址巴士之低次八位元，A0 至 A7。並且，T2 上升緣後， $\overline{\text{IORQ}}$ 動作。若為輸入 / 輸出讀取 (亦即輸入) 作業，則 $\overline{\text{RD}}$ 信號與 $\overline{\text{IORQ}}$ 同時動作。外部設備之控制電路於認知輸入作業 ($\overline{\text{IORQ}}$ 且 $\overline{\text{RD}}$) 後，會自動將資料置於資料巴士。於 T3 下降時，該項資料被鎖入 CPU。

若為寫入 (即輸出) 作業，則 $\overline{\text{WR}}$ (而非 $\overline{\text{RD}}$) 與 $\overline{\text{IORQ}}$ 同時動作。 $\overline{\text{WR}}$ 線動作之前 (T1 期間)，CPU 之資料已先置於資料巴士上。於週期時間內，外部輸入 / 輸出設備之控制電路會讀取此一資料。

注意到，於輸入與輸出週期內，T2 週期後均自動插入一等待週期 Tw。此乃因為輸入 / 輸出作業時， $\overline{\text{IORQ}}$ 信號動作至 CPU 讀取 $\overline{\text{WAIT}}$ 線之狀態，兩者間之時間過於短暫，致輸入 / 輸出口無足夠之時間以解碼位址，以及在需要等待時令 $\overline{\text{WAIT}}$ 線動作。同時，若無此一等待狀態，則亦很難設計出一能以 CPU 之全速動作的 MOS 輸入 / 輸出設備。於等待狀態期間內，CPU 讀取 $\overline{\text{WAIT}}$ 線之狀態，以決定是否繼續進入次一等待狀態。如記憶器讀取一般，於輸入 / 輸出讀取時，RD 信號用以將被選定口之資料置於資料巴士上。而輸入 / 輸出

寫入時，WR 線則作為輸入／輸出口之時序。

12-11 巴士請求／認可週期

任何時刻，一外部設備可藉著使 Z80CPU 之 $\overline{\text{BUSRQ}}$ 輸入線動作，以獲得位址巴士 A15～A0，資料巴士 D7～D0，與 $\overline{\text{MREQ}}$ ， $\overline{\text{RD}}$ ， $\overline{\text{WR}}$ ， $\overline{\text{IORQ}}$ ，及 $\overline{\text{RFSH}}$ 等控制線之控制權。正常上，這樣做的理由乃在允許外部設備控制器能直接與記憶器溝通，以便資料能直接來回傳遞於高速輸入／輸出設備與記憶器之間，而不必假手 Z80 微處理器（此種作業稱為直接記憶器存取，DMA）。看 12-13 圖。當 $\overline{\text{BUSRQ}}$ 信號致能（動作）時，微處理器於機器週期之最後 T 週期上升緣時，測及此一信號。T 週期完成後，於次一 T 週期期間，微處理器輸出 $\overline{\text{BUSAk}}$ 信號，反應此一請求。此一時刻，位址巴士、資料

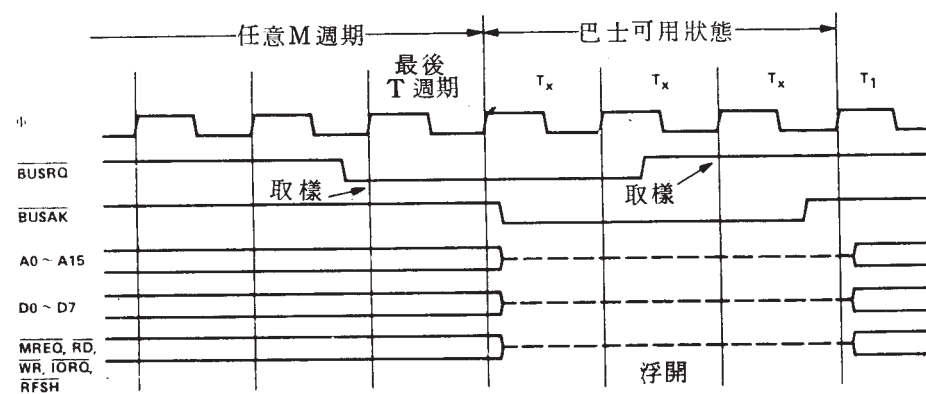


圖 12-13 巴士請求／認可週期

巴士、與其他控制信號一起被置定成三態之高阻抗狀態。爾後，這些線上之任意改變均非由 Z80CPU 所引起的，而 Z80CPU 亦不會影響此些線的狀態。

輸入／輸出設備之控制電路於完成 DMA 傳輸（通常為一個位元組）後，會使 $\overline{\text{BUSRQ}}$ 變成高電位，恢復不動作狀態。次一 T 週期上升緣期間，CPU 會測知此一狀態，並於再次一 T 週期時，使 $\overline{\text{BUsAk}}$ 恢復高電位，將其禁能。CPU 然後自剛剛放開巴士控制權之點繼續往下處理。

由上述說明我們了解，CPU 反應巴士請求之最長時間為一個機器週期。並且，外部設備之控制電路對巴士之控制可維持至任意久。但是，注意到，若 DMA 週期延續太久，並且系統有動態記憶器，則外部設備之控制電路必須負責記憶復新。此外，於巴士請求週期內，CPU 不能為 $\overline{\text{NMI}}$ 或 $\overline{\text{INT}}$ 信號所插斷。

12-12 插斷請求／認可週期

於 Z80 系統，外部輸入／輸出設備可藉著使插斷請求線 $\overline{\text{INT}}$ 動作，對 Z80 微處理器提出插斷請求。Z80 微處理器於每一指令之最

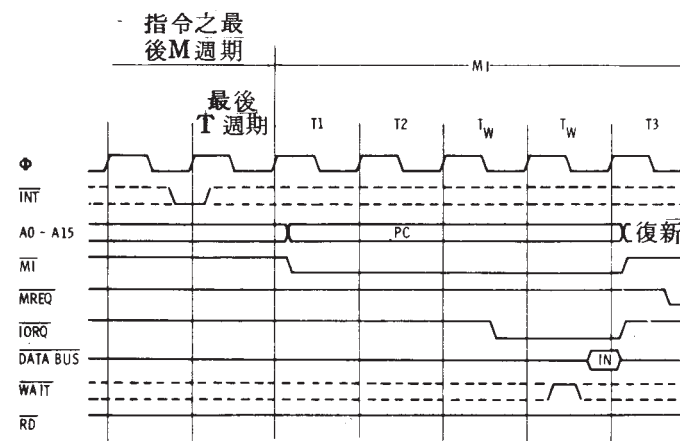


圖 12-14 插斷請求／認可週期

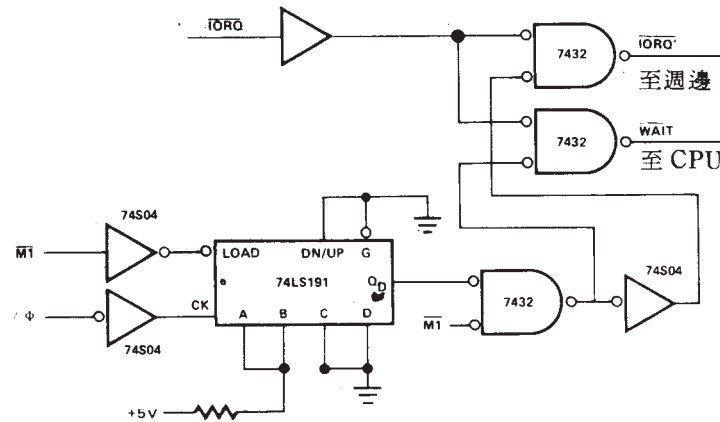


圖 12-15 A 增長插斷認可時間之電路圖

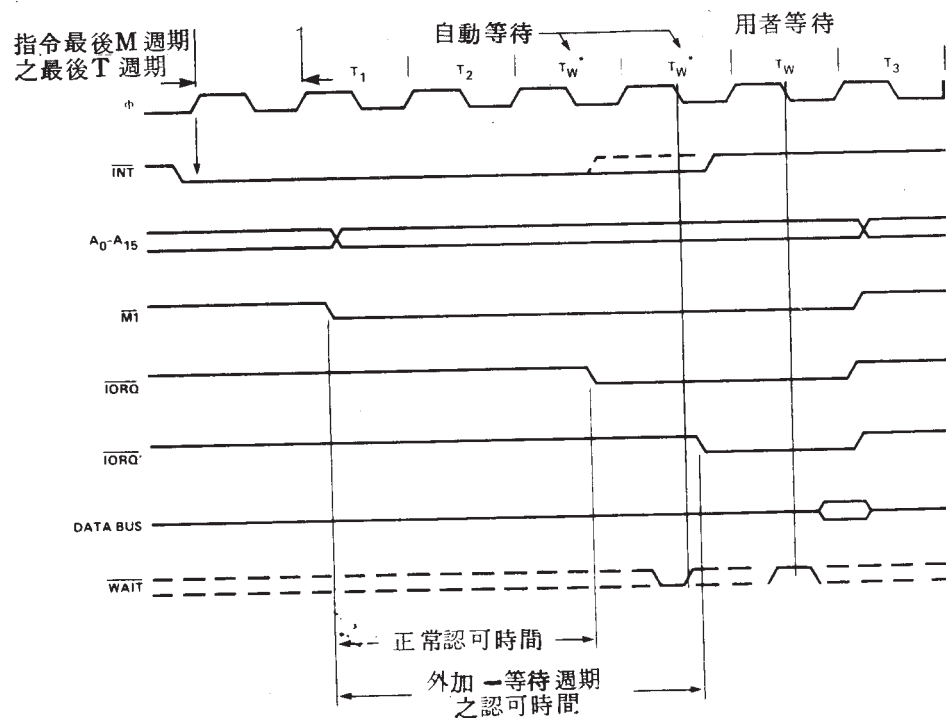


圖 12-15 B 增加等待狀態之插斷請求 / 認可週期

後機器週期的最後 T 週期上升緣時，讀取 $\overline{\text{INT}}$ 線之狀態。若該輸入線為邏輯零（低電位），CPU 插斷致能旗號置為 1，並且巴士請求動作不發生，則 Z80 微處理器於完成最後機器週期後，會立即進入一如圖 12-14 所示之插斷週期，接受並處理此一插斷請求。注意到上述之敘述，Z80 微處理器欲進入一插斷週期，有三個條件。

於接受插斷後，Z80 微處理器會產生一特殊之 M1 週期。此一週期即為插斷週期。T1 時， $\overline{\text{M1}}$ 信號動作。然後，T2 與兩個等待狀態（等待狀態由內部產生），使外部菊花環（daisy-chained）形之插斷電路，有足夠時間對插斷認可信號採取反應，並將八位元之插斷向量（interrupt vector）置於資料巴士。外部輸入／輸出設備之插斷邏輯，將 M1 與 $\overline{\text{IORQ}}$ 之組合視為一插斷認可信號。於測及此兩信號後，外部插斷電路之反應即將插斷向量置於資料巴士。該項位址於 T3 上升緣時被鎖入 Z80 CPU。T3 期間， $\overline{\text{M1}}$ 與 $\overline{\text{IORQ}}$ 信號失效，記憶復新作業開始。插斷週期期間進一步的動作則視插斷型態而定，此將於後面討論。

注意，前面說過，Z80 於插斷週期時自動加入兩個等待狀態。若有必要，插斷認可之時間可再予增長，圖 12-15 所示即為以一可程式化計數器，增長插斷認可時間之情形。（A）圖所示為實際電路圖，而（B）圖為時序圖。

12-13 不可罩蓋插斷週期

圖 12-16 所示即為不可罩蓋插斷之機器週期期間，Z80 微處理器之動作情形。不可罩蓋插斷信號， $\overline{\text{NMI}}$ 無法被插斷致能旗號所禁能。同時，其亦比插斷請求 $\overline{\text{INT}}$ 具有更高之優先權。如同插斷請求之狀況一般，Z80 微處理器於目前指令最後機器週期之最後 T 週期時，認可不可罩蓋插斷信號。圖 12-16 所示乃此一插斷作業之第一部情形。微處理器對不可罩蓋插斷之反應同於記憶體讀取作業。唯一的差別是，微處理器並不讀取資料巴士上之資料，而自動將程式計數器之內含

推入堆疊器並跳至位址 0066H 之記憶位置。該位置起所儲存的，即為不可罩蓋插斷之處理常式。

T3 與 T4 期間， $\overline{\text{RFSH}}$ 與 $\overline{\text{MREQ}}$ 信號動作，R 暫存器之內含被置於位址巴士之低次八位元，A0 至 A7，記憶復新動作開始。

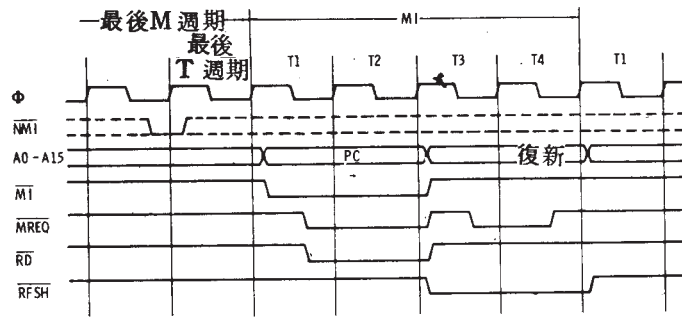


圖 12-16 不可罩蓋插斷週期

取兩插斷線 ($\overline{\text{NMI}}$ 及 $\overline{\text{INT}}$) 之狀態。若此兩插斷線有任一者動作，而且 CPU 之插斷致能旗號為 1，則 Z 80 將於次一 T 週期上升緣時，脫離暫停狀態。自此之後 Z 80 進入前面敘述過之插斷認可週期。注意，若插斷請求 ($\overline{\text{INT}}$) 與不可罩蓋插斷 ($\overline{\text{NMI}}$) 兩信號線同時動作，則微處理器將先處理具有較高優先之不可罩蓋插斷。

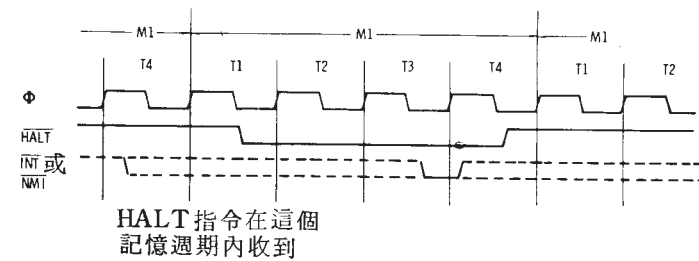


圖 12-17 脫離暫停狀態

12-14 脫離暫停 (EXIT FROM HALT)

於執行 HALT 軟體指令時，Z 80 微處理器自動將 $\overline{\text{HALT}}$ 信號致能，使其輸出低電位，然後，連續反復地產生 M1 週期，執行無運算 (NOP) 指令 (程式計數器內含保持遲滯不前)。記憶器送出之資料被忽略。M1 最後兩個 T 週期時，記憶復新邏輯致能，以便復新作業能繼續進行。

Z 80 微處理器之暫停 (HALT) 狀態，能以 $\overline{\text{RESET}}$ ， $\overline{\text{NMI}}$ ，或 $\overline{\text{INT}}$ 信號將之插斷。換言之，唯有此三種輸入信號，方能使 Z 80 微處理器脫離暫停狀態。Z 80 微處理器於每一 T4 週期上升緣時，讀

第 13 章

輸入／輸出程式設計

至此，我們已經學會了如何將資料來回傳遞於微電腦系統之記憶器與微處理器的各暫存器之間。我們也學會了運用微處理器之暫存器，以及使用各種指令運算數據。緊接，我們所必須探討的是前述之設備如何與外界溝通，此即所謂的**輸入／輸出**（Input / Output，常簡記為**I / O**）。

輸入（input）指自諸如鍵盤、磁碟、與察覺器等外部週邊設備獲取資料；而**輸出**（output）指將微處理器或記憶器之資料，送至諸如印字機、磁碟機、CRT 顯示器、繼電器、或物理察覺器等之外部設備。

此地，我們將分三階段進行討論：第一，先介紹 Z 80 之各種輸入／輸出指令。第二、再介紹如何以輸入／輸出指令與各種外部設備達成溝通。第三，最後，下一章我們再介紹如何同時應付數個輸入／輸出設備。此即所謂的**輸入／輸出排時**（I / O Scheduling）技巧。

13—1 輸入／輸出指令

Z 80 具有一組特別之輸入／輸出指令，這點使其與其它之一般微處理器不同。輸入／輸出指令使 Z 80 系統之用者，能以程式控制輸入／輸出（programmed I/O）之方式，每次自外部設備輸入一位

元組之資料，或將一位元組之資料輸出至外部設備。Z 80 最基本的兩個輸入／輸出指令為：IN A, (n) 與 OUT (n), A。它們達成**累加器**與輸入／輸出口間之資料傳輸。於 Z 80，資料尚可來回傳遞於任一 CPU 暫存器與外部輸入／輸出口之間。此則由 IN r, (C) 與 OUT (C), r 兩指令達成。此外，Z 80 亦具有能半自動或全自動傳輸最多 256 位元組之資料的**整批**輸入／輸出指令。本節分別將之一一介紹如下。

13—1—1 累加器輸入／輸出指令

IN A, (n) 與 OUT (n), A 是兩個承襲 8080 之輸入／輸出指令。前者自選取之輸入口讀取一位元組之資料，並將之存入累加器 A，後者將一位元組之資料，自累加器 A 傳送至被選取之輸出口。兩個指令均長兩位元組，第一位元組為運算碼，而第二位元組為八位元之輸入／輸出口位址，其值由 0 至 255。Z 80 微處理器執行這兩指令時，指令提供之輸入／輸出口位址 n，出現於位址巴士之低次八位元—A₀ 至 A₇，而累加器 A 之內含則被置於位址巴士之高次八位元—A₈ 至 A₁₅。

在欲選取之輸入／輸出設備僅有 256 個之情況下，若輸入／輸出設備亦解碼任一 A₈ 至 A₁₅ 之位址線，則累加器之內含必須特別清除為零（譬如在使用 IN A, (n) 指令之前）。為了簡單起見，於爾後之例子，我們均假設輸入／輸出設備少於 256 個，並且位址線 A₈ 至 A₁₅ 並未連接至輸入／輸出設備，以致累加器內含不必先特別清除為零。

IN A, (n)

欲自諸如鍵盤之輸入設備輸入某項資料，微處理器與輸入設備間必須有一套彼此溝通的辦法。譬如，為了免於讀取錯誤（非預期）之資料，在自輸入設備讀入資料之前，微處理器必須先詢問輸入設備，欲輸入之資料是否已經準備好。若未，則微處理器就不能立即讀取資料

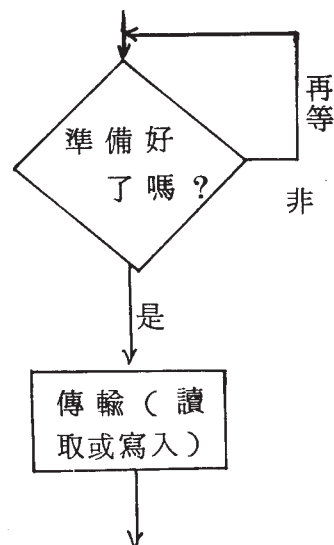


圖 13-1 詢問式輸入／輸出

，而必須再等。因此，輸入作業之程序通常為

- 1 資料字組是否已備好？
- 2 若尚未，再執行步驟 1。
- 3 否則，讀取資料字組。

於此一作業過程，輸入設備之控制電路必須知道，資料字組何時準備好，並且能讓微處理器知道此一狀態。為達此目的，一般之作法是於輸入設備設置**兩個暫存器**，一個儲存資料是否已備好之**設備狀態**，而另一個則儲存**資料字組**。此兩個暫存器各指定有一獨特之位址。欲知要輸入之資料是否已備妥，微處理器可先讀取輸入設備之狀態暫存器。而若欲輸入資料，則微處理器可讀取資料暫存器。

舉個例子而言，例 13-1 之程式即為以上述方式自鍵盤讀取一資料字組之副程式。該副程式之流程圖如圖 13-2 所示。

例 13-1 鍵盤輸入副程式

；該副程式自電傳打字機之鍵盤輸入一資料字組。鍵盤之狀態暫存器；之位址為 5，資料暫存器之位址為 6。當資料備好時，狀態暫存器；之最低次位元為 1。

```

;
READC      EQU      $
WAIT       IN        A, (5)      ; 讀取設備狀態。
          BIT        0, A        ; 資料備妥了嗎？
          JP         Z, WAIT     ; 若未，則再等。
          IN         A, (6)      ; 若是，則讀取。
          RET
  
```

副程式一開始即讀取鍵盤之狀態，並加以測試。若資料字組尚未備妥，狀態暫存器為 0，控制跳回 WAIT 處，重新再詢問輸入設備之狀態。於前面三個指令，微處理器一直在**等待**輸入設備將資料準備好，而不作其它任何事，因此稱之為**等待迴路**。一旦資料備妥，輸入設備狀態暫存器之最低次位元即為 1（不再為 0），故 JP Z, WAIT 跳越不成，微處理器緊接執行 IN A, (6) 指令，自輸入設備之資料暫存器，讀取輸入之資料字組。然後回返。假若操作員在鍵盤上按了 A 鍵，則此時累加器所含的即為英文字母 A 之 ASCII 碼—41H。鍵盤在被讀取後，狀態暫存器又自動恢復為 0，並一直等至下一輸入位元組備妥（亦即，操作員打入次一鍵）時，再為 1。

如以上所言，於例 13-1，微處理器與輸入／輸出設備之溝通方式是，由微處理器主動詢問（藉著讀取狀態訊息）輸入／輸出設備是否已準備好，若是，則進行交易（輸入或輸出），若非，則微處理器繼續等。如圖 13-1 所示，此種溝通之方式稱為**詢問法**或**取樣法**（polling）。取樣法僅是微處理器與輸入／輸出設備間溝通方式之一種，此種方式之效率並不高。有關溝通之方式，本章第三節有更詳細

之討論。

OUT (n), A

OUT (n), A 指令之動作十分類似於 IN A, (n) 指令。輸出作業之程序爲：微處理器先讀取設備控制器之狀態，並加以測試。若設備控制器已準備好（已處理完前一項資料，可再接受次一字組），則程式執行 OUT (n), A 指令，將先前存入累加器 A 之內含輸出至資料巴士。此一輸出同時將設備控制器置定於“忙碌”（busy）狀態。在輸出文數字自設備控制器之暫存器傳遞至輸出設備時，控制器之忙碌狀態就自動清除。例 13-2 所示即爲一典型之輸出副程式。譬如，該副程式可用以將累加器 A 所含之一文數字 ASCII 碼，輸出至連接至系統之印字機，令其印出。圖 13-3 所示即爲此一輸出副程式之流程圖。

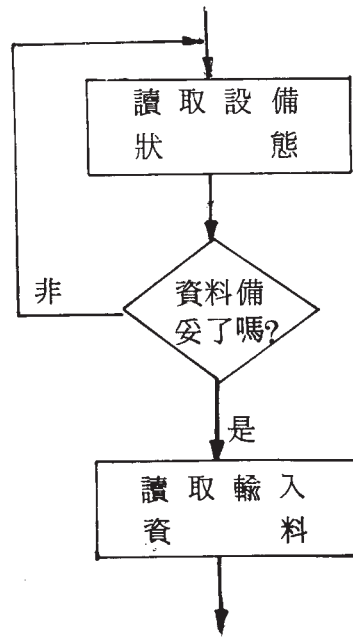


圖 13-2 以取樣法自鍵盤輸入資料

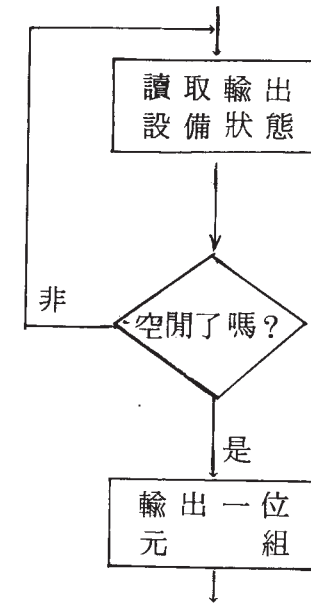


圖 13-3 取樣法輸出副程式之流程圖

此時，狀態訊息假設與輸入副程式相吻合。狀態暫存器之第 1 位元顯示設備控制器之輸出部份，是否已寫出了前一位元組。若是，則該位元值爲 0（“不再忙碌”），累加器 A 之內含輸出至資料巴士。

例 13-2 輸出副程式

；該副程式將一位組之資料，自累加器輸出至某一輸出設備。輸出設備之狀態暫存器與資料暫存器的位址分別爲 3 與 4。設備忙碌時，狀態暫存器之第一位元爲 1。反之，若設備已能接受次一位元組資料，則狀態暫存器之第 1 位元爲 0。

WRITEC EQU \$; 副程式名稱。
 LOOP IN A, (3) ; 讀取設備狀態。

BIT	1, A	; 測試設備狀態。
JP	NZ, LOOP	; 若忙碌，則再等。
OUT	(4), A	; 否則，輸出。
RET		; 回返。

以上兩個副程式是在極為簡化之情況下，用以輸入或輸出一個文數字資料。雖然實際應用之狀況可能還要複雜些，但就簡單之輸入／輸出設備而言，上述之副程式仍十分好用的。在上述之輸入／輸出作業前，或許還必須先對輸入／輸出設備發出一清除（clear）命令。此一清除動作通常於每一整批輸入／輸出傳輸前進行，以便清除設備之狀態（免於其一直停留於錯誤之狀態），並起始傳輸。視設備控制器之不同，此一清除亦有所差異。最簡單的狀況就是將一清除命令寫入狀態暫存器。

13—1—2 使用暫存器C之輸入／輸出指令

除上述兩者外，Z 80 之其它非整批傳輸的輸入／輸出指令，均以C暫存器儲存一0至255之輸入／輸出位址。IN r, (C)指令可將資料自輸入口傳送至A, B, C, D, E, H或L等任一CPU暫存器。反之，OUT (C), r指令能將任一此些暫存器之內含，傳送至C暫存器內含所選取之輸出口。微處理器在執行此些指令時，C暫存器之內含被置於位址巴士之低次八位元（A₀～A₇），而B暫存器之內含則被輸出至位址巴士之高次八位元（A₈～A₁₅）。正如下面欲討論之整批輸入／輸出指令一般，B暫存器之內含可用以溝通狀態訊息或輸出目前之位元組計數。當然，在C暫存器持輸入／輸出設備之真正位址，而資料巴士用以傳輸出入設備控制器之資料時，B暫存器亦可不用。指令使用前，暫存器之佈置的工作仍類似於使用累加器之輸入／輸出指令。

例 13—3 另一輸出副程式

; 該副程式將一位元組之資料，輸出至位址 30 H 之輸出設備。輸出；資料來自 HL 所指之記憶位置。設備之狀態暫存器假設與資料暫存器共用同一位址。設備於備好狀態時，狀態指示器值為 1；反之，其值為 0。

WRITEC	LD	D, (HL)	; 取入次一字組。
	LD	C, 30H	; 置定設備位址。
LOOP	IN	E, (C)	; 讀取設備狀態。
	JP	Z, LOOP	; 未備好，則再等。
	OUT	D, (C)	; 輸出資料字組。
	RET		

於例 13-3 之副程式，欲輸出之資料字組自記憶器取入於D暫存器，且C暫存器存設備位址30H。此時，輸出設備假設為一“僅能寫入”之設備，故狀態暫存器與資料暫存器共用同一位址。IN E, (C)指令將設備之狀態訊息讀入E暫存器。若設備已處於“空閒”狀態（狀態指示器值為1），則輸出指令將D暫存器之內含輸出。

特別注意，使用C暫存器之輸入指令與先前之累加器輸入指令有點不同。此時輸入之作業影響了旗號。正負數、零值，與極性等旗號之值會受輸入資料之影響。因此，在IN r, (C)指令之後，不必再特別加一BIT指令。

13—1—3 整批輸入／輸出指令

Z 80 之整批輸入／輸出指令，每次能輸入／輸出1至256個位元組之資料。如同其它之整批（區段）作業指令一般，此些指令能順向或逆向地處理過整個區段。並且，資料之傳遞能半自動（每指令一個位元組）或全自動（整批資料以一個指令完成）地進行。

INI

INI 指令將一個位元組之資料，自輸入／輸出設備之控制器傳至記憶器。如同 IN r, (C) 指令之情況一般，開始前，C 暫存器先儲存一輸入／輸出位址。當 INI 被執行時，輸入之資料直接存入 HL 暫存器對所指之記憶位置。資料存妥後，HL 暫存器對之內含加一，B 暫存器之內含減一。若指令執行後 B 暫存器之內含值為零，則零值旗號 (Z) 置定為 1。例 13-4 所示即為 INI 使用前暫存器之佈建，以及 INI 之輸入迴路的情形。

例 13-4 整批輸入副程式 (INI)

；該副程式以 INI 指令，自位址為 23H 之設備，輸入 100₁₀ 個位元組之資料，存於位址 BUFFER 起之連續記憶位置。

；

```
LD HL, BUFFER      ; 記憶緩衝區位址。
LD BC, 25600+23H   ; 100 位元組及設備位址。
STATUS IN D, (C)    ; 讀取設備狀態。
BIT 0, D            ; 並加測試。
JP Z, STATUS        ; 未備好則再等。
INI                 ; 輸入一資料位元組。
JP NZ, STATUS       ; 未完則繼續。
RET
```

LD BC 指令將 B 暫存器之內含 (位元組計數) 置定為 100₁₀，且將 C 暫存器之內含 (設備位址) 置定為 23H。由於 INI 與 JP 兩指令計費時 6.5 μ S，故此方法所能達成之最大資料傳輸率，比前面之輸入／輸出迴路高百分之五十。

INI 指令所對等之指令系列為

STATUS

```

:
:
IN      A, (C)
LD      (HL), A
INC     HL
DEC     B

```

雖然 INI 指令十分方便，輸入、儲存，與指示器維護等一氣呵成，但此種輸入／輸出作業仍是**程式化控制** (programmed) 之輸入／輸出。於輸入／輸出傳輸完成前，微處理器無法執行任何其它之作業；換言之，微處理器之作業速度受限於輸入／輸出設備之反應速度。就 Teletype 公司之 ASR-33 印字機而言，即使使用整批傳輸輸入／輸出指令，最大資訊傳輸率仍僅限於每秒十個位元組；每兩位元組傳輸之間有 100 微秒的時間浪費於狀態測試 (即等待) 上。

INIR

除了 B 暫存器所指明之位元組數全數由自己一個指令傳輸完畢外，INIR 指令之動作情形類似。輸入之資料以順向 (由位址小至位址大) 方式儲存於 HL 所指之記憶位置。除了最後一次 (B=0) 外，INIR 指令重複一次所需的時間為 5.25 微秒。因此，最大資料傳輸率趨近於每秒 190000 個位元組。但 INIR 指令並無測試設備控制器狀態之能力；沒有事先做好之“**握手連絡**”電路。現在的問題為，CPU 如何知道次一資料位元組已備妥。答案是，CPU 並不接受通知。輸入／輸出設備之控制器必須快達每秒鐘能傳輸 200000 個位元組，否則，其就必須自動於時序中引入等待週期 (見時序信號一章)，以使 INIR 之指令時間能與設備之傳輸率相等。若此，則 INIR 所需之執行時間則隨輸入／輸出設備之不同而異，同時，INIR 之傳輸唯有等 N 個位元組皆傳輸完畢後，才算完成。

若硬體電路產生之等待週期取代 INI 指令之軟體狀態測試，則 INIR 之典型用法如下：

```

READC    LD      HL, BUFFER    ; 緩衝區位址。
          LD      B, 200        ; 位元組計數。
          LD      C, 30H        ; 設備位址。
          INIR                    ; 輸入200個位元組。

```

除了進行之方向相反外（由高位址位置至低位址位置），IND 與 INDR 指令之作業，完全與 INI 與 INIR 相同。例 13-5 所示即為一使用 IND 指令，作整批輸入之例子。此一程式之功能完全類似於例 13-4 之程式。

例 13—5 輸入副程式 (IND)

；該副程式以 IND 指令，自位址 23H 之輸入設備，輸入 100 個位元
；組之資料，儲存於位址 BUFEND 以降之記憶位置。
；

```

READC    LD      HL, BUFEND    ; 緩衝區記憶位址。
          LD      B, 100        ; 共100個位元組。
          LD      C, 23H        ; 輸入設備位址取入C。
STATUS    IN      D, (C)        ; 讀取狀態字組。
          BIT     0, D          ; 資料備好了嗎？
          JP      Z, STATUS      ; 若未，則再等。
          IND                    ; 若是，則輸入
          JP      NZ, STATUS      ; 未完時繼續。
          RET

```

OUTI 指令將一位元組之資料，自 HL 暫存器對內含所指之記憶

位置，傳送至 C 暫存器內含所選取之輸入／輸出設備。傳輸後，HL 暫存器對內含加一，位元組計數器 B 之內含值減一。若 B 已遞減至零，則零值旗號置定為 1。除了資料傳遞之方向不同外，OUTI 指令極類似於 INI。例 13-6 之程式說明了 OUTI 指令如何使用。

例 13—6 輸出副程式 (OUTI)

；該副程式將位址 BUFFER 起連續 100 個記憶位置之內含，輸出至
；位址 23H 之輸出設備。
；

```

          LD      HL, BUFFER    ; 指至資料記憶區。
          LD      B, 100        ; 計一百個位元組。
          LD      C, 23H        ; 設備位址取入C。
STATUS    IN      D, (C)        ; 讀取設備狀態。
          BIT     0, D          ; b0 = 0 表忙碌。
          JP      Z, STATUS      ; 若忙碌，則再問。
          OUTI                    ; 否則，輸出一位元組。
          JP      NZ, STATUS      ; 未完時繼續。
          RET                    ; 完了，回返。

```

OUIR 為類似於 INIR 之自動整批輸出指令。視位元組計數器 B 之內含而定，該指令可輸出 1 至 256 個位元組。同樣地，若輸出設備之最大資訊傳輸率小於 190000 位元組 / 秒，則輸入／輸出週期必須自動加入等待週期，以期輸入／輸出設備之速度能與微處理器相吻合。底下即為一使用 OUIR 之輸出迴路的例子。

```

WRITE    LD      HL, BUFFER    ; 資料起始位址。
          LD      B, 200        ; 共二百個位元組。
          LD      C, 30H        ; 設備位址 30 H。

```

OTIR

；輸出200位元組。

13-2 產生脈衝信號與延遲

以上我們已將 Z 80 微處理器之所有輸入／輸出指令介紹完畢，緊接，讓我們看看此些指令的最常應用。

就最簡單的情況而言，計算機的輸出作業就是關掉或起動某一輸出設備。為此，程式設計者必須使計算機之輸出，自某一邏輯準位變成另一邏輯準位（譬如，自邏輯 0 變為邏輯 1，或反之。），以改變輸出設備之狀態。假設有一外部繼電器連接至一稱為“OUT 1”之暫存器的第 0 位元上。為了將之起動，程式就必須在其所對應之位元位置上寫入一邏輯 1。若以 OUT 1 代表系統中此一輸出暫存器之位址，則能起動此一繼電器之程式為：

```
TURNON    LD      A, 00000001B    ; 輸出內含取入 A。
           OUT     (OUT 1), A      ; 輸出至設備。
```

顯然，這兒所用的是累加器輸出指令：OUT (n), A。

於此一例子，我們假設暫存器之其它七位元均不相干。不過，經常不是這種情況。此些位元可能仍連接有其它繼電器。因此，讓我們將上述程式稍加改進。假設我們仍欲起動前述繼電器，但却不影響輸出暫存器之其它位元的狀態。若輸出暫存器之內含同時可讀寫，則合乎我們要求的程式即為

```
TURNON    IN      A, (OUT 1)
           OR      00000001B
           OUT     (OUT 1), A
```

此一程式先讀取輸出暫存器之原有內含，存入累加器 A。然後將第 0

位元值改成 1。最後結果再寫回輸出暫存器。因此，於整個作業後，除了接至輸出暫存器第 0 位元之繼電器被起動外，其餘者皆不受影響。

圖 13-4 所示即為起動一繼電器的情形。

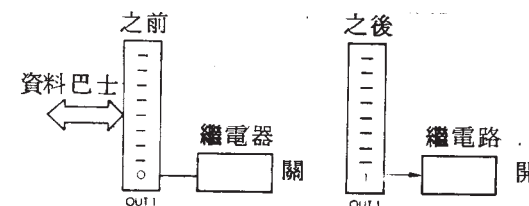


圖 13-4 起動一繼電器

產生脈衝

產生脈衝 (pulse) 的情形與上述之準位 (level) 輸出完全相同。如圖 13-5 所示，微處理器先將輸出位元起動，然後關閉，這就產生了一個脈衝。不過，這次還有一個問題：脈衝的長度為何？下面，我們就探討此一問題——計數延遲之產生。

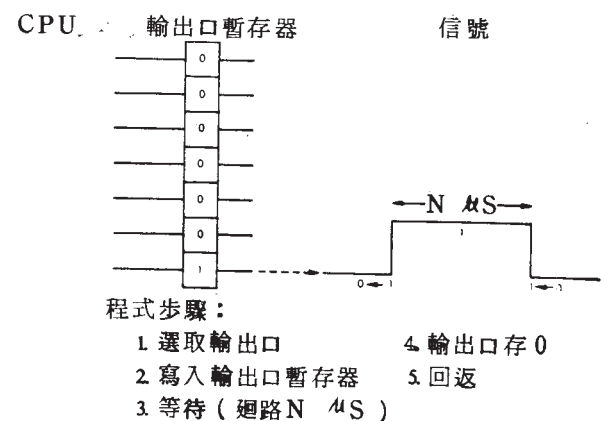


圖 13-5 以程式產生之脈衝

延遲之產生與度量

時間延遲可以硬體的方式產生，亦可以軟體的方式達成。此時，我們先探討以軟體達成之方式，隨後，再看以一硬體之計數器，稱為**可規劃期間計時器**（Programmable Interval Timer，簡記為**PIT**），完成的情形。

軟體延遲是藉著**計數**達成。程式設置一暫存器作為計數器，開始之時先將某一計數值存入其內，然後於迴路內每次將之減一，直至其值為零，計數完畢，延遲達成。圖 13-6 所示即為以軟體方式達成時間延遲之方法。

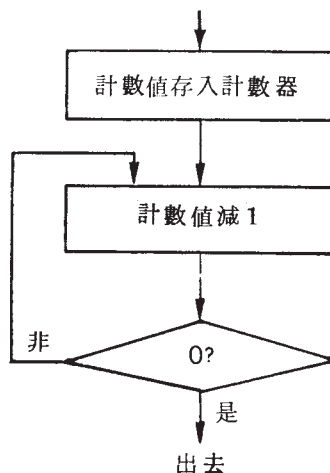


圖 13-6 軟體延遲之基本流程

舉個例子而言，下面之指令系列即產生 82 個時序週期之延遲：

```

DELAY    LD    A , 5      ; 計數值 5 取入計數器 A。
NEXT     DEC    A         ; 計數值減一。
  
```

J P NZ, NEXT ; 迴路至計數值為零。

此一程式以累加器 A 為計數器。程式一開始，最初計數值 5 取入累加器。緊接指令將累加器所含之計數值減一，若累加器內含尚未遞減至零，則緊接之跳越指令使控制再度跳回標題 NEXT 處，繼續遞減。當累加器內含最後遞減至零，程式控制跑出迴路，繼續往下執行緊接之指令。

上述程式為何達成 82 個時序週期之延遲呢？第四章曾經介紹過，執行立即定址之 LDA 指令需 7 個時序週期，DEC 指令需 4 個週期，而 J P NZ, NEXT 指令，於跳越成功時需 12 個週期，跳越不成時需 7 個週期。因此，程式所消耗之時序週期總數為第一指令 7 週期，第二指令 4 週期被執行 5 次，而第三指令 12 週期被執行 4 次，7 週期被執行一次（最後一次跳越不成）：

$$\begin{aligned} \text{延遲之時序週期數} &= 7 + (4 + 12) \times 5 - 5 \\ &= 82 \text{ 週期} \end{aligned}$$

倘若每一時序週期時間為 0.5 微秒（時序頻率 2 MHz），則上述程式所產生之時間延遲為 41 微秒。

剛剛所介紹之延時迴路經常用於各種輸入／輸出程式中，讀者務必了解。請再試下列習題：

習題 13-1： 剛剛所介紹之三指令延時迴路，所能產生之最大與最小延遲各為若干？

習題 13-2： 請將剛剛程式修改成產生 100 微秒之延遲。

更長之延遲

假若欲產生更長之時間延遲，則一種簡單的方式就是於剛剛程式之 DEC 與 J P 兩指令間，再加上其它之指令。而最單純的方式就是

加 NOP 指令（該指令產生 4 個時序週期之延遲，但任何事都不做。）

除了此種方式外，另一種辦法就是使用長度更長之計數器。程式可利用一長達十六位元之暫存器對作為計數器，亦可使用兩個長八位元之暫存器而製作一雙層迴路之程式。於後者，第一暫存器之內含於內層迴路中每次被減一，當此一暫存器之內含遞減至零時，控制執行外層迴路，將第二暫存器所含之計數值減一，然後繼續再執行內層迴路。顯然，除了控制第一次進入內層迴路外，第二暫存器（即外層迴路之迴路計數器）所含之計數值再被減一之前，內層迴路之指令將被反復執行 256 次。最後，當第二暫存器之計數值亦遞減至零時，程式結束。

例 13-7 之程式亦即為一結合雙層迴路與十六位元迴路計數器兩種特色，以一長 24 位元之計數值作計數的延時副程式。於內層迴路（LOOPB），迴路計數器為長十六位元之 HL 暫存器對。而於外層迴路（LOOPA），迴路之計數器為八位元之 B 暫存器。換言之，B 暫存器為 24 位元計數器之高位元組，其起始值為 COUNTH；而 HL 暫存器對為計數器之“低位元組”（其實為十六位元），其起始值為 COUNTL。

例 13—7 24 位元計數器之延時副程式

；該副程式以一 24 位元之計數器產生時間延遲。B 暫存器為計數器；之高位元組，其起始值為 COUNTH；HL 暫存器對為計數器之低；“位元組”，其起始值為 COUNTL。副程式所能產生之最低延遲；為 53 個時序週期（就 4 MHz 而言，為 13.25 微秒），此時 COUNTH 與 COUNTL 兩者皆為 1。HL 暫存器對之值每增加 1，延；遲時間即增加 23 個時序週期。而 B 暫存器之內含每增加 1，延遲；時間即增加 $23 \times \text{COUNTL}$ 個時序週期時間。

```
DEL24  LD      B,COUNTH      ; 計數值高位元組。
DEL16  LD      DE,-1
LOOPA  LD      HL,COUNTL     ; 計數值低位十六位元。
LOOPB  ADD     HL,DE          ; 內層迴路計數值減一。
        JR      C,LOOPB      ; 未零時繼續。
        DJNZ    LOOPA        ; 外層迴路計數值減一；
                                ; 且未零時繼續。

RTS
```

自然，更長之時間延遲能使用三個位元組以上之計數值達成。此種方式就好比汽車上之里程計的原理一樣。當最低次（右邊）之齒輪由 0 增至 9 以後，次一齒輪值即加一，如此類推。

然而，此種方法有一主要缺點：於延時計數時，微處理器於幾百毫秒或甚至幾秒內，什麼事都沒做。假若計算機正好無所事事，那倒無妨。但一般而言，微電腦不會如此閒著無事的。因此，較長之時間延遲通常不以軟體方式製作。事實上，若目的僅在為某一已知狀況提供一確定反應時間，則即使再短之延遲亦不用軟體之方法。此時，唯有採用硬體延遲。此外，若系統使用插斷，則計數迴路在被插斷後，計時之精確性也就喪失了。

習題 13-3：試寫一延遲時間為 100 毫秒（為電傳打字機之典型延遲時間）之延時程式。

硬體延遲

硬體延遲是以可規劃期間計時器（Programmable Interval Timer，簡寫為 PIT），或簡稱計時器，製作而成。一開始，計時器之暫存器亦存入一數值。所不同的是，計時器會週期性地自動遞減計數器。此一週期通常能由程式設計者加以調整或選定。每當計時器

之計數值遞減至零。其通常會送出一插斷信號給微處理器。同時，其亦可將計算機能週期性讀取之狀態位元置定為 1。插斷的使用將在本章稍後討論。

計時器之其它作業型態尚包括由 0 開始，計數信號之週期，或計數所接收到之脈衝個數。作為期間計時器時，我們稱計時器動作於**單觸**（one-shot）型態。而於計數脈衝時，稱之為**脈衝計數**（pulse counting）型態。有些計時器可能還會包括多個暫存器，以及數個程式設計者可選用之設備。

Z80-CTC 界面晶片能用以產生硬體計時。此一電路能加以規劃，使其在某一特定時間後，即對外輸出一“到時”信號。

察覺脈衝

察覺脈衝的問題正巧與產生脈衝相反，但却增加了一項困難：雖然輸出脈衝是在程式的控制下產生的，但計算機所接收到之輸入脈衝並不與程式**同步**。為了測知脈衝輸入，有兩種方法可用：**取樣**（polling）與**插斷**（interrupt）。插斷在本章稍後討論，此刻，我們先看取樣技巧。

於取樣技巧，程式連續不斷地讀取某一已知暫存器，然後測試某一位元位置之值。我們可假設代表脈衝輸入之位元（亦即程式欲測試之位元）平常為 0。而當有脈衝輸入時，該位元值為 1。如此，只要程式發現該位元值為 1，即代表脈衝已經輸入。此一脈衝察覺程式為

```
POLL    IN      A, ( INPUT )    ; 讀取輸入暫存器。
ON       BIT     0, A           ; 測試其第 0 位元值。
        JR      Z, POLL        ; 若 0 則再繼續詢問。
```

其中，INPUT 代表輸入暫存器之八位元位址。同時，假設其第 0 位元為脈衝輸入位元。

反之，若假設脈衝輸入線平常為“1”狀態，而程式所欲測試的是其何時變為“0”，則程式變為

```
POLL    IN      A, ( INPUT )    ; 讀取輸入暫存器。
        BIT     0, A           ; 測試輸入位元值。
        JR      NZ, POLL        ; 若仍為 1，則再問。
START    :                    ; 脈衝開始輸入。
```

注意，若此一輸入線所連的是一部電傳打字機，則此一指令系列的測試正巧為一起始位元（starting bit）之測試。換言之，若測試成功，控制開始執行 START 處之指令時，代表電傳打字機又輸入了一文數字。

測知輸入脈衝長度

測知輸入脈衝之長度可以與計算輸出脈衝長度相同之方式達成。同樣有硬體與軟體兩種技巧可用。在以軟體測知脈衝期限時，計數器值規律性的加一，然後程式查核脈衝是否仍然存在。若是，則程式一直迴路至脈衝消失為止。最後，計數器所含之計數值再用以計算脈衝之期限。此一程式如下所示。

例 13—8 測知輸入脈衝長度之副程式

```
; 該副程式測知自位址 INPUT 之輸入暫存器的第 0 位元輸入之脈
; 衝的長度。副程式回返後，B 暫存器之內含乘以 35 個時序週
; 期時間，方為真正之脈衝長度。
;
DURTN    LD      B, 0           ; 計數器清除為零。
AGAIN    IN      A, ( INPUT )   ; 讀取輸入暫存器。
        BIT     0, A           ; 脈衝輸入了嗎？
```

```

        JR      Z, AGAIN      ; 若尚未，則再問。
COUNT INC      B            ; 計數值加一。
        IN      A, ( INPUT)   ; 檢查脈衝是否還在。
        BIT     0, A
        JR      NZ, COUNT     ; 若是，則繼續計數。
        RTS                      ; 否則，回返。

```

程式一開始，計數器 B 之值先清除為零。緊接三個指令藉著測試輸入暫存器第 0 位元之值是否已由“0”變“1”，而測知輸入脈衝是否已抵達。若是，則控制進入標題 COUNT 以下之計數迴路。每次將計數器 B 之值加一，並且測試脈衝是否已經消失（第 0 位元由 1 變成 0）。若是，則程式回返；否則，計數繼續。當然，於此一例中，脈衝之長度不可使計數值大於八位元之 B 暫存器所能容納者。

13-3 資訊傳遞之方式

知道如何產生與察覺脈衝信號之後，我們現在開始討論較大量資料之傳輸。資料傳輸的方式有兩種：並行傳輸與串行傳輸。讓我們分別加以探討。

13-3-1 並行資訊傳輸

並行資訊傳輸指的是，整個資料字組之所有位元一起**同時傳遞**，因此又稱為並行字組傳輸。於此種傳輸，資料之來源地與目的地之間，必須有數條線並行搭起，以讓資料字組之位元能同時傳遞，每一位元循一條線傳遞。

緊接，我們舉一並行傳輸之例子。如圖 13-7 所示，假設位址 INPUT 處有一八位元之資料欲傳輸，當狀態字組顯示資料字組有效時，微處理器必須讀取該字組。狀態資訊含於位址 STATUS 處之第 7 位元。試寫一自動讀取並且儲存每一輸入之資料字組的程式。

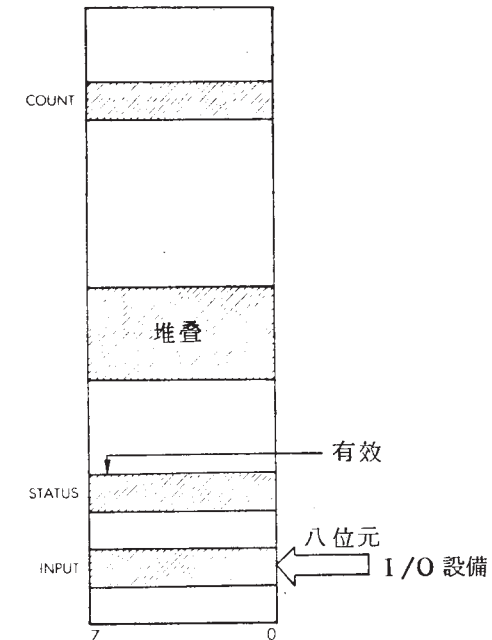


圖 13-7 並行字組傳輸一位址分佈圖

為了簡單起見，假設欲讀取之字組數事先曉得，並且存於位址 COUNT 處。若非如此，則程式必須測試所謂的**結尾文數字**（break character），以知曉輸入資料何時完了，程式必須結束。此一結尾文數字或許是一“*”或，其它文數字。

圖 13-8 所示即為一事先知道欲傳輸字組數之並行資訊傳輸的流程圖。此一流程甚為直截了當。程式首先讀取欲傳輸之字組數，然後緊接測試狀態訊息，看是否字組（或外部設備）已準備好。若未，則再等；否則，傳輸字組，字組計數值減一，未完時繼續。此一流程圖可同時適用於輸入與輸出。例 13-9 所示即為根據此一流程圖之演算法設計而成的並行資訊傳輸副程式。

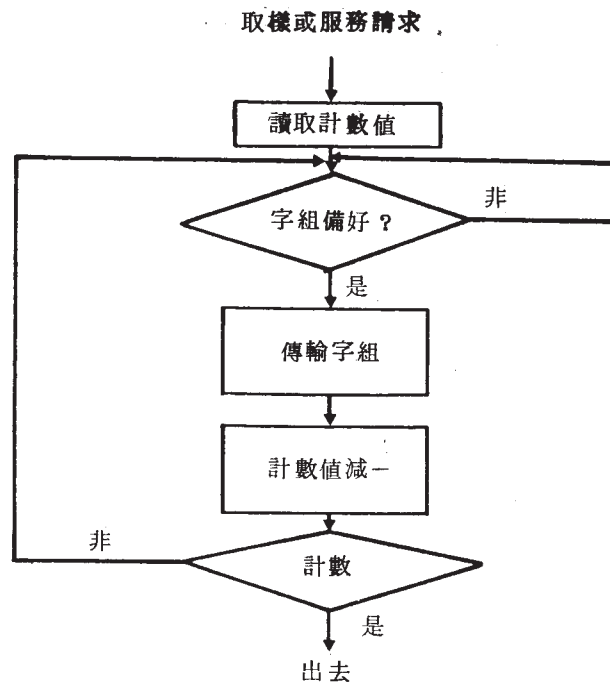


圖 13-8 並行字組傳輸——流程圖

例 13-9 並行字組傳輸之輸入副程式

；此副程式自位址 INPUT 處連續輸入數個資料字組。資料字組之個數已知存於位址 COUNT 之記憶位置。若下一字組已備妥讀取，則位址 STATUS 處之第 7 位元值為 1；否則，其值為 0。該位置之狀態資訊被讀取後，其值自動清除為零。所有輸入之資料字組均被存入堆疊器。

```

;
PARAL  LD    A,(COUNT)    ; 字組個數取入 B。
        LD    B,A
WATCH  IN    A,(STATUS)    ; 輸入字組備妥了嗎?
        BIT   7,A
  
```

```

JP     Z,WATCH    ; 若不，則再等。
IN     A,(INPUT)  ; 若是，則讀取。
PUSH   AF         ; 並存入堆疊器。
DEC    B          ; 字組計數值減一。
JP     NZ,WATCH   ; 未完則繼續。
RTS                      ; 完了，回返。
  
```

於例 13-9 之副程式，程式一開始之兩個指令將欲輸入之資料字組數，自位址 COUNT 處取入 B 暫存器。由於擴展定址之八位元取入指令僅能取入累加器 A，所以，資料必須先取入 A 後，再轉至 B。若不如此，則必須以十六位元之指令，同時取入 B 與 C 兩暫存器。

緊接的三個指令自位址 STATUS 處讀取狀態資訊，並測試該狀態字組之第 7 位元是否為 1。若非，則表欲輸入字組尚未備妥，程式再等。若是，則表資料字組已備妥。緊接的兩個指令將資料字組讀取，並推入堆疊器存起。然後，迴路（字組）計數器之值減一，未完則繼續。最後當 B 暫存器之值遞減至零時，所有位元組傳輸完畢，副程式結束，控制回返。

有一點必須注意的是，程式假設“資料備好”旗號（STATUS 位置之第 7 位元）在被讀取後，自動清除為零。事實上，一般輸入設備之控制器的實際情形也是這樣的。因為，唯有如此，方能達到確保該旗號之值於資料被讀取後呈現 0，而於次一資料字組抵達（備妥）時再被置定為 1 之要求。

例 13-9 之九個指令的程式可視為一標竿（benchmark）。一標竿程式即為一經過仔細最佳化的程式，其專門用以測試中央處理器在某一已知狀況下的能力，以供比較參考之用。並行字組傳輸就是其中一種典型狀況。由於此一程式已設計成具有最快之速度與最高之效率。因此，讓我們計算一下其最大之資訊傳輸率。

由第四章之指令個別介紹表中可查出，例 13-9 副程式之每一指

令所需的執行時間依序分別是 13, 4, 11, 8, 7/12, 11, 11, 4, 7/12 個時序週期。最短的執行時間發生於每次程式一讀取狀態訊息，資料就已經備好之情況。換言之，JP 指令每次測試皆失敗，跳越均不成。因此，程式所耗費之最短時間為

$$13 + 4 + (11 + 8 + 7 + 11 + 11 + 4 + 7) \times \text{COUNT}$$

。假若忽略最初起始計數器值所需之 17 週期，則傳輸一個字組所需之時間為 59 個時序週期。就 2 MHz 之時序而言，需時 29.5 微秒。因此，最大的資訊傳輸率為

$$\frac{1}{29.5(10^6)} = 33 \text{ K 位元組 / 秒}$$

習題 13-4： 假設欲傳輸之字組數大於 256，試修正例 13-9 之程式，並評估其對最大資訊傳輸率所造成之影響。

習題 13-5： 設法修正例 13-9 之程式，以期能改進其速度：

- 1 以 JR 指令代替 JP。
- 2 使用 DJNZ 指令。
- 3 使用 INIR 或 INDR。

上述之程式真的最佳化了嗎？

13-3-2 串行資訊傳輸

串行 (serial) 資訊傳輸指的是，資料字組之每一位元依序先後地（而非並列地），循一共同線傳輸至目的地。因此，又稱為串行位元傳輸。換言之，於串行資訊傳輸，資料之來源地與目的地之間僅有一條電線連起，且任一時刻此兩位置間僅有一位元之資訊在傳送，資料字組之位元，先後一一地循同一條線發送出去（或接收進來）。若兩位元間之時間間隔相同，則稱為**同步 (synchronous)** 傳輸。否則

，若位元間之時間間隔彼此不同，則稱為**非同步 (asynchronous)** 傳輸。

下面，我們設計一能同時用於同步傳輸與非同步傳輸兩種情況之輸入副程式。捕捉串行輸入資料之原理非常簡單：只要我們監視著輸入線，然後遇有位元進來時就將之讀取，並且移位使之進入一儲存暫存器。然後，等字組（八位元）組成完畢後，再將之存入記憶器，並開始組合次一字組即可。為了簡化起見，我們同樣假設欲接收之字組數事先曉得，並存於位址 COUNT 之記憶位置。因為，若非如此，程式必須增加數個指令，以隨時檢驗終止文數字是否已來到。圖 13-9 所示即為此一串行輸入作業之流程圖。其程式如下：

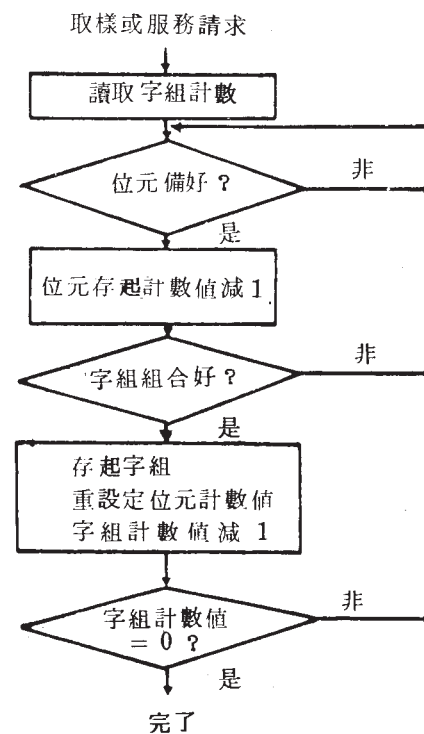


圖 13-9 串行位元傳輸作業的流程圖

例 13-10 串行位元傳輸之輸入副程式

；該副程式自位址 INPUT（八位元）之輸入口，輸入多個八位元之資料字組。資料以位元串行之方式輸入。欲輸入之資料位元組數事先已知，並存於位址 COUNT 之記憶位置。位址 INPUT 之輸入暫存器的第 0 位元含輸入資料位元；而第 7 位元含一狀態旗號，該旗號值為 1 時代表資料位元有效（已經備妥）。

```

;
SERIAL  LD      C, 0          ; 輸入字組先清除為零。
        LD      A, (COUNT)  ; 位元組計數取入 B。
        LD      B, A          ;
LOOP    IN      A, (INPUT)    ; 讀取輸入口。
        BIT     7, A          ; 狀態位元之值為何？
        JR      Z, LOOP      ; 若 0，則再等。
        SRL     A             ; 否則，資料位元（第 0
                                ; 位元）移入進位旗號。
        RL      C             ; 然後組合於 C 暫存器。
        JR      NC, LOOP     ; 繼續至輸入八位元為止。
; 八位元之字組已組合完畢。
        PUSH    BC           ; 輸入字組存入堆疊器。
        LD      C, 01H       ; 重新置定記號位元。
        DEC     B            ; 位元組計數值減一。
        JR      NZ, LOOP     ; 再組合下一字組。
        RTS                    ; 完了，回返。

```

例 13-10 之副程式以 C 暫存器組合輸入字組。因此，一開始，C 暫存器先被清除為零。然後，位元組計數由位址 COUNT 之記憶位置取入 B 暫存器。

標題 LOOP 以下至 JR NC, LOOP 指令為止，構成一輸入字組

第十三章 輸入／輸出程式設計

組合迴路。首先，程式先讀取位址 INPUT 處之輸入暫存器，該暫存器之第 0 位元含輸入資料位元，且第 7 位元為一顯示輸入資料位元是否有效之狀態位元。若輸入資料位元有效，則該旗號值為 1；否則，其值為 0。緊接之兩個指令（BIT 與 JR）測試狀態位元之值，若其值為 0（表資料位元尚未備妥），則程式繼續等到其值變成 1 為止。一旦狀態位元值變為 1，即表示剛剛 IN A, (INPUT) 所讀取之資料中，第 0 位元即為欲輸入之資料位元。因此，SRL A 指令立刻將該位元移入進位旗號，然後，RL C 指令將之組合於 C 暫存器。

JR NC, LOOP 指令然後測試剛剛自 C 暫存器之最高次位元移出的位元值是否為 1，若是，則表八位元已組合完畢。若非，則控制跳回標題 LOOP 處，繼續讀取下一位元。當然，由於程式一開始 C 暫存器即已被清除為零，因此，前述之測試是基於一種假設：資料開始輸入之前必須先有一“1”的標記位元，該位元代表資料位元“自此”開始。由於第一字組組合完成後，緊接即有一 LD C, 01H 指令，在每次字組組合之前，先將 C 暫存器之內含令為 00000001₂，以作次一字組八位元是否已組合完成之測試的依據，因此，標記位元只要加於所有資料之最前端一個即可。

當輸入字組之八位元均已讀取且組合完畢後，PUSH BC 指令緊接將輸入字組推入堆疊器（記憶器）中存起。然後，LD C, 01H 指令設定次一字組組合迴路結束測試的標記位元。DEC B 與 JR NZ, LOOP 兩個指令將位元組計數器 B 之值減一，若其值未達零，則控制跳回 LOOP 處，繼續輸入次一位元組。否則，若 B 之值已為零，則副程式回返。圖 13-10 所示即為此一程式例題之作業情形。

有一點再強調的是，於例 13-10，雖然我們假設資料位元與狀態位元共用一暫存器。但實際上它們亦可分別佔用不同位址之暫存器。還有，所有輸入之字組均被存入堆疊器。一旦程式結束後，此些輸入資料可以前面介紹過之區段搬運副程式，將之移至記憶器之任何其它區域。

時，亦須先確定輸出設備已能接收資料了，方能將資料送出。這有兩種方法可以運用：第一、握手連絡（或稱交互連絡）（handshaking）。第二、插斷（interrupt）。

13-4-1 握手連絡

於本章第一節介紹輸入／輸出指令時，我們曾舉過如何自鍵盤輸入一項資料，以及如何將一項資料由印字機印出之例子。那種微處理器與輸入／輸出設備溝通的方式，就是一種握手連絡。還記得，微處理器在自鍵盤輸入一項資料之前，必須先讀取鍵盤之狀態暫存器的訊息，看看次一項資料是否已經備妥可立即讀取了。同樣地，微處理器在送出一項資料給印字機印出前，亦須先詢問印字機，前一項資料是否已印出，而可再接收下一項資料了！輸入與輸出作業之握手連絡，分別如圖 13-9 (a) 與 (b) 所示。

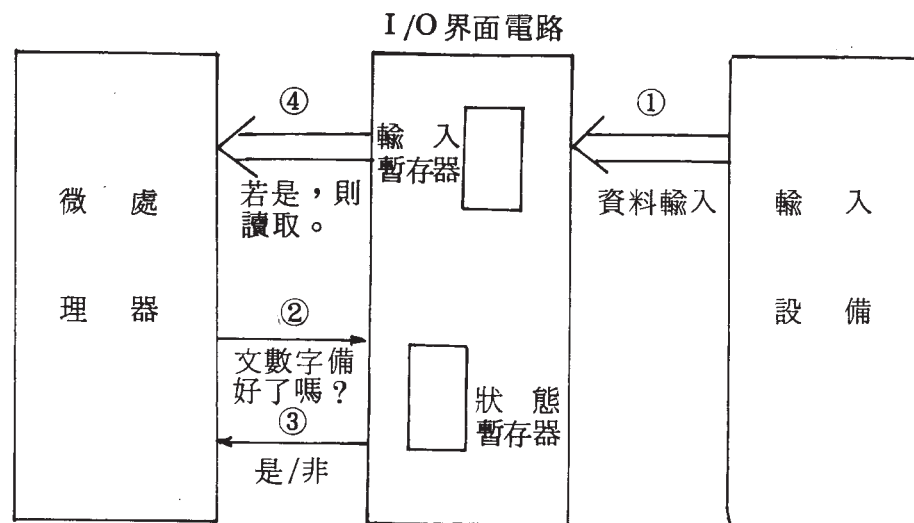
如圖 13-11 (a) 所示，輸入設備（如鍵盤）經一輸入／輸出界面電路連接至微處理器界面電路主要含兩暫存器：一儲存自輸入設備輸入之資料字組的輸入暫存器，以及一顯示輸入資料是否已備妥之狀態暫存器。微處理器在讀取輸入暫存器輸入之前，會先讀取狀態暫存器之內含，並加以測試，以看新的位元組資料是否已自輸入設備輸入至輸入暫存器。若是，則再將之讀取。此種一問一答之方式，即為所謂的握手連絡。

同理，輸出亦然。如圖 13-11 (b) 所示，微處理器於輸出一新的位元組資料前，會先詢問 I/O 界面電路（實際上為讀取狀態暫存器，並測試狀態位元值），前一輸出之位元組資料是否已送給輸出設備，若是，（此時狀態暫存器會有所記載）則再將新的位元組輸出。

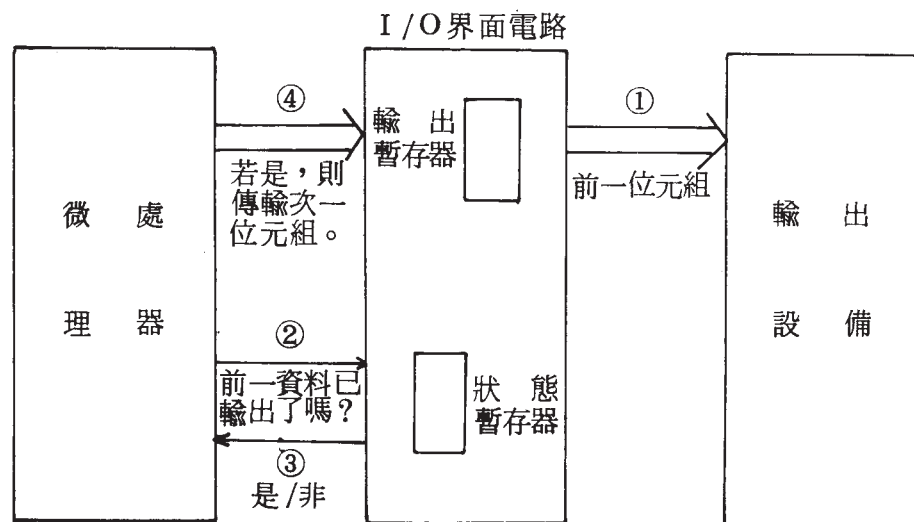
13-4-2 七段 LED 顯示

顯示器與顯示碼

於一般廉價或較簡單之微電腦系統，最常見之輸出設備要算是七



(a) 輸入



(b) 輸出

圖 13-11 握手連絡

段發光二極體 (LED) 顯示器了。如圖 13-12 所示，每一七段LED顯示器由七段個別之顯示段組成。為了容易區別起見，七個顯示段分別稱作A段，B段，…，G段等。如圖 13-13 所示，藉著點亮某幾根適當的顯示段，每一七段顯示器能顯示出“0”至“9”等十個十進數字，或甚至“0”至“F”等十六個十六進數字。譬如，假若使A，B，C，D，E，F等顯示段發亮，就能顯示出“0”字。

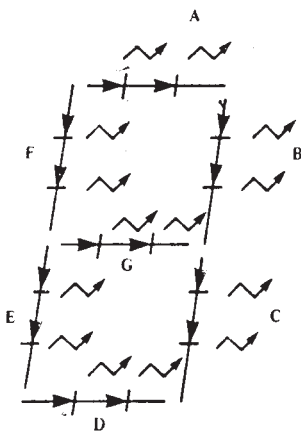


圖 13-12 七段LED顯示器

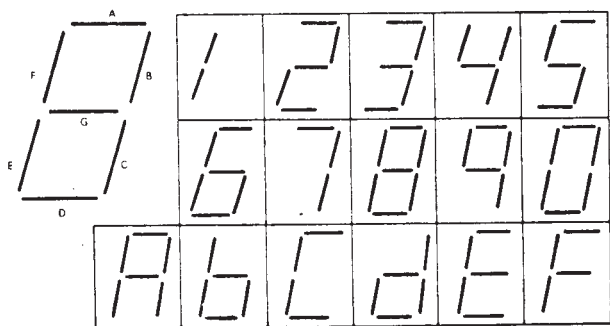


圖 13-13 七段LED顯示器所顯示之十六進數字

假設界面電路之輸出暫存器的第0至第6位元，分別連接至七段LED顯示器之A至G顯示段，並且假設位元“1”使顯示段發亮。如此，欲顯示出十進數字“0”，微處理器就必須對輸出暫存器寫入（輸出）“01111111”，寫成十六進制為“3F”，此即所謂的七段LED顯示碼。試完成圖 13-14 所示之七段LED顯示碼表格。

十六進數字	七段顯示碼	十六進數字	七段顯示碼
0	3F	8	
1		9	
2		A	
3		B	
4		C	
5		D	
6		E	
7		F	

圖 13-14 七段LED顯示碼表格

LED並無記憶性。顯示段唯有在加電壓的時間，才會發亮。祇要輸出電壓一消失，顯示段立刻跟着熄滅。因此，一種點亮多個LED顯示之低成本辦法，就是僅設一輸出推動程式，而使每一LED顯示器輪流發亮，而非每一LED顯示器專設一點亮裝置，使之永遠“不停地”發亮。不過，為了避免視者看出閃爍之現象，輪換點亮每一LED顯示器之速度必須夠快。由於人的視覺暫留時間是1/16秒，因此，巡迴點亮一次的週期間必須小於此一數值。

例 13-11 所示即為一將累加器所含之十六進值，輸出至C暫存器所選取之LED顯示器的副程式。

例 13-11 七段 LED 顯示副程式

；該副程式以查表法，將累加器 A 所含之十六進數值，輸出至 C 暫存器內含所選取之 LED 顯示器。七段 LED 顯示碼依十六進數字之順序，儲存於位址 SEG TBL 起之連續十六個記憶位置。

```

LEDPLY  LD  E,A           ; A 含十六進數字值。
        LD  D,0           ; DE 含表格之位移。
        LD  HL,SEG TBL    ; HL 作索引暫存器。
        ADD HL,DE         ; 求取顯示碼之表格位址。
        LD  A,(HL)        ; 取入七段顯示碼
        LD  B,50 H        ; 設定延遲時間。
        OUT (C),A         ; 數字顯示出。
DELAY   DEC B             ; 這兩指令為延遲迴路。
        JR  NZ,DELAY      ; 產生延遲。
        DEC C             ; 輸出口位址減一。
        LD  A,C
        CP  MINLED        ; 最後一個 LED 了嗎？
        JR  NZ,OUT        ; 若非，則回返。
        LD  BC,(MAXLED)   ; 若是，則令 C 再指回第一
                           ; 個 LED。

```

```
OUT     RET
```

於例 13-11 之程式，累加器 A 含欲顯示之十六進數字值。C 暫存器含下一個欲點亮之 LED 的位址（輸入／輸出位址僅有八位元）。HL 為索引指示器，指至七段顯示碼表格之起點。而 DE 暫存器對儲存顯示碼距表格起點之位移。譬如，若欲顯示之十六進數字為 7，則此一數字之七段 LED 顯示碼即儲存在位址 SEG TBL + 7 之記憶位置。

程式一開始，表格位移先取入 DE 暫存器對（A 取入 E，D 存 0）。同時，HL 暫存器對儲存顯示碼表格之起始位址 SEG TBL。然後，ADD 指令將表格起始位址與位移相加，求得欲顯示數字之七段顯示碼的位址，結果存於 HL。緊接指令將該顯示碼自記憶位置取入累加器 A。然後，OUT (C)，A 指將之輸出至 C 暫存器內含所選取之 LED 顯示器，在此之前，B 暫存器先存 50H，以備延遲計數之用。

顯示碼輸出後，程式緊接進入一延遲迴路，產生 16×50 個時序週期之延遲。之後，C 暫存器所含之輸出口位址減一，指至下一個 LED 顯示器。CP MINLED 指令然後檢查剛剛點亮的是否為最後一個顯示器，若非，則程式控制跳至 OUT 處，進行回返。若是，則 LD BC,(MAXLED) 指令使 C 暫存器再指至第一個 LED 顯示器，以備下一次的顯示。在此，MAXLED 代表第一個七段顯示器之位址，而 MINLED + 1 表最後一個七段顯示器之位址。

例 13-11 之程式寫成副程式形式，此副程式每被叫用一次，就有一緊接之 LED 顯示器被點亮；亦即有一十六進數字被顯示出。藉着不斷反覆地叫用此一副程式，我們即可推動多個 LED 顯示器。譬如，下面之指令系列即為在含有三個七段顯示器之輸出設備上，不斷的顯示出十六進數 704 的情形。

```

CONT    LD      A,7
        CALL    LEDPLY
        LD      A,0
        CALL    LEDPLY
        LD      A,4
        CALL    LEDPLY
        JP      CONT

```

當然，在實際應用上，欲顯示之數字可能來自某些特定記憶位置，譬如，某一輸入鍵盤之輸入緩衝記憶區。

習題 13-13：通常在顯示數字之前必須先熄滅每一顯示段。試在例 13-11 之程式加上此一作業所必須之指令（於輸出數字顯示碼之前先輸出“00”。）

習題 13-14：若標題 DELAY 往上提升一行，試問顯示結果有何不同？時序有何影響？

習題 13-15：您可能已經發覺，程式之最初四個指令事實上即在作十六位元之索引記憶器存取，但因未使用索引定址法致作得很别扭。試問有否其它較佳之方法呢？

習題 13-16：例 13-11 之副程式使用了 B, D, E, H, 與 L 等暫存器，並改變了其原有內含。若副程式可自由使用位址 T1, T2 ..., T5 等位置。試在副程式之開頭與結尾，加上必要之指令，使副程式回返時，此些暫存器的內含完好如初。

習題 13-17：目標同前一習題，但副程式無法使用 T1 等記憶位置。試問如何達成同一目的？（暗示：使用每一計算機皆具有之一特殊設施。）

13-4-3 電傳打字機輸入／輸出

較完整或較昂貴之微電腦系統，通常都具有一既為輸入且為輸出之設備：電傳打字機（teletype）。電傳打字機事實上為一鍵盤輸入與一印字機輸出之組合，其屬於一種串行之設備。資訊字組皆以串行之方式輸入或輸出，一次一個位元。

圖 13-15 所示即為輸入或輸出電傳打字機之資訊格式。每一文數字皆寫碼成八位元之 ASCII 形式。此外，每一文數字之前均有一起始位元，且之後有兩個停止位元，總共為 11 位元。平常，資料線均保持於“1”狀態，因此，該線上之“1”至“0”轉換，即表示資訊傳輸自此開始，接收設備應知緊接於後的即為資料位元。

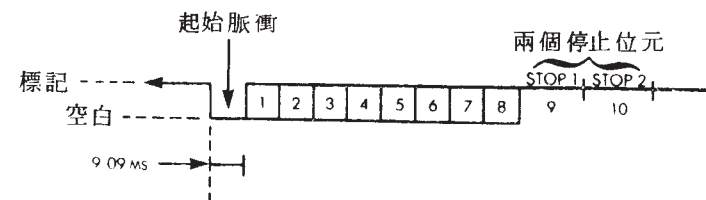


圖 13-15 電傳打字機之資料字組格式

標準之電傳打字機每秒能傳輸 10 個文數字，此意即電傳打字機每秒能傳輸 110 個位元。這速度隱含每一位元的週期時間為 9.09 微秒。換言之，不論是輸入或輸出，每兩連續位元之間必須有 9.09 微秒之延遲。下面，我們就設計一電傳打字機輸入（亦即微處理器讀取電傳打字機之輸出）之程式。此一程式之流程圖如圖 13-16 所示。

例 13-12 電傳打字機輸入副程式

；此副程式自電傳打字機接收一位元組之資料，控制回返時，資料字組存於 C 暫存器內。位址 STATUS 之第 7 位元為電傳打字機之狀態位元，而位址 TTYBIT 之第 0 位元儲存輸入之資料位元。每一資料位元在被讀取後，皆立即被反存回原處。

；

```
TTYIN  IN      A, (STATUS)    ; 讀取狀態資訊。
      BIT      7, A           ; 資料備妥了嗎？
```

	JR	Z, TTYIN	; 若未，則再等。
	CALL	DELAY 1	; 對準位元之中央。
	IN	A, (TTYBIT)	; 讀取起始位元。
	OUT	(TTYBIT), A	; 反存回去。
	CALL	DELAY 9	; 延遲一位元時間。
	LD	B, 08H	; 設定位元計數。
NEXT	IN	A, (TTYBIT)	; 讀取一資料位元。
	OUT	(TTYBIT), A	; 反存回去。
	SRL	A	; 並存入進位旗號。
	RR	C	; 組合於C暫存器。
	CALL	DELAY 9	; 延遲一位元時間。
	DEC	B	; 位元計數值減一。
	JR	NZ, NEXT	; 迴路至八位元讀完。
	IN	A, (TTYBIT)	; 讀取停止位元。
	OUT	(TTYBIT), A	; 反存回去。
	CALL	DELAY 9	; 略過第2停止位元。
	RET		

開始時，程式先讀取並且測試電傳打字機之狀態位元，若資料尚未備妥（狀態暫存器之第7位元為0），則程式一直等至其備妥。當輸入資料位元備妥時，程式先叫用DELAY.1副程式，產生1/2位元時間（4.5ms）之延遲，以便緊接之讀取（IN）指令能讀得起始位元之中間點。在將讀取位元反存回原處後。副程式緊接叫用DELAY 9副程式，產生一位元時間（9ms）之延遲。此時，時序正巧位於第1個資料位元之中點。

於將位元計數器B之值設定成8後，程式開始進入一讀取八個資料位元，並將之組合成字組存於C暫存器之迴路。於此一迴路，程式先讀取輸入資料位元，將之移入進位旗號，再移入C暫存器之最高

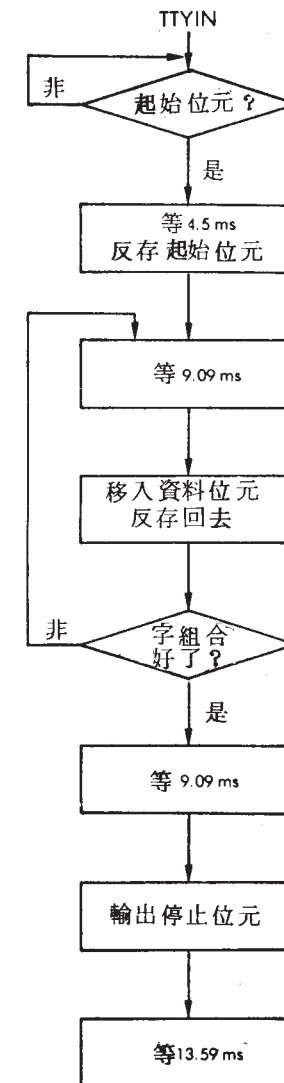


圖 13-16 電傳打字機輸入副程式之流程圖

次位元 (RR C)，完後，再叫用 DELAY 9 副程式，延遲一位元時間，先指至次一資料位元之中點。

當八個資料位元均讀取且組合完成後，控制自 JR NZ, NEXT 指令下來。讀取第一停止位元，略過第二停止位元，然後回返。圖 13-17 所示即為此一程式作業之圖解說明。

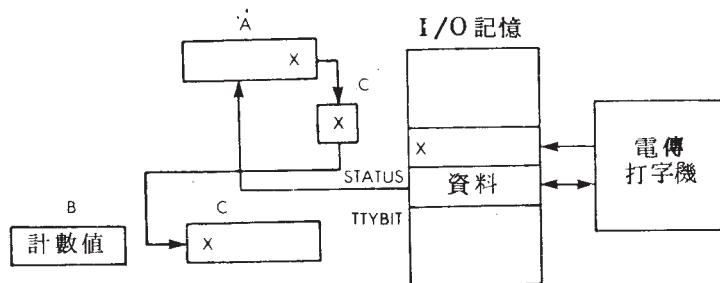


圖 13-17 電傳打字機輸入

例 13-12 之程式邏輯非常簡單，但必須小心檢視。唯一的新觀念是，每一位元被讀取後，皆又立即被存回原處。此乃電傳打字機之標準特色。只要用者按下一鍵，該鍵即被送至微處理器，然後又送回電傳打字機之印字機部份。這證明了傳輸線目前正在動作，並且，當印字機正將文數字正確地印在紙上時，微處理器亦在動作。

習題 13-18：寫一延遲 9.09 毫秒時間之副程式 DELAY 9。

根據例 13-12 之程式，我們可寫一將位址 CHAR 之記憶位置內含由電傳打字機印出之副程式。此一副程式之流程圖如圖 13-18 所示。

例 13-13 電傳打字機輸出副程式

```

;此副程式將位址 CHAR 之記憶位置的內含，自位址為TTYBIT
;之電傳打字機印出。
;

```

TTYOUT	LD	B, 11	; 位元計數值 = 11。
	LD	A, (CHAR)	; 輸出文數字取入 A。
	OR	A	; 清除進位作起始位元。
	RLA		; 進位移入 A。
NEXT	OUT	(TTYBIT), A	; 輸出一位元。
	CALL	DELAY9	; 延遲一個位元時間。
	RRA		; 準備下一位元。
	SCF		; 進位 = 1，作停止位元。
	DEC	B	; 位元計數值減一。
	JR	NZ, NEXT	; 迴路至 11 位元作完。
	RET		

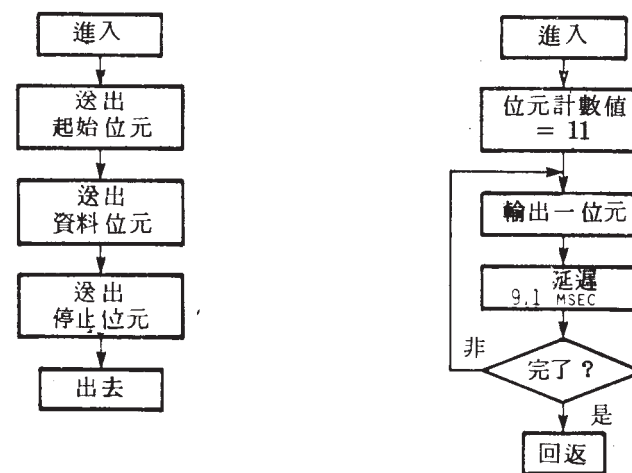


圖 13-18 電傳打字機輸出副程式之流程圖

程式一開始，位元計數器 B 之起始值先置定為 11，欲輸出文數字之 ASCII 碼自位址 CHAR 之記憶位置取入累加器 A。OR A 先將進位旗號清除為零，作為起始位元。然後，RLA 將起始位元移入累加器之第 0 位元，以便輸出。標題 NEXT 以下為位元輸出迴路。每次，累加器之第 0 位被輸出至位址 TTYBIT 之輸出暫存器，然後，DELAY 9 副程式被叫用一次，產生一位元時間之時間延遲，以便能輸出次一位元。在輸出次一位元之前，RRA 指令先將次一欲輸出之位元移入累加器之第 0 位元，然後，SCF 指令將進位旗號置定為 1，該旗號值在爾後之右移過程中，被移入累加器，並貼於資料位元之後（左），故可作為停止位元。此步驟完成後，程式將位元計數器之值減一，然後控制跳回標題 NEXT 處，繼續輸出次一位元，直至一起始位元，八個資料位元，以及兩停止位元均輸出完畢為止。

13-4-4 印出一串文數字

本章第一節我們曾經舉過如何自印字機印出一文數字之程式例題，現在，我們藉著叫用此一副程式，以印出位址 START 至 START+N 等記憶位置之內含。

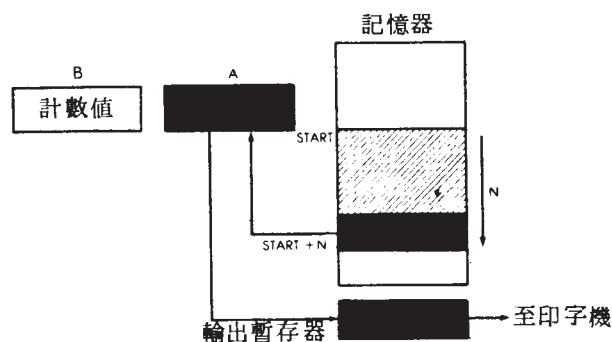


圖 13-19 印出某一記憶區段之資料

例 13-14 印出一串文數字

；該副程式叫用印出一文數字之副程式 WRITEC（例 13-2），以印出位址 START 至 START+N 等記憶位置之內含。

```

;
PSTRING  LD      B,LENT      ; 文字串長度取入 B。
          LD      HL,START    ; 起始位址取入 HL。
NEXT      LD      A,(HL)      ; 取入一文數字。
          CALL    WRITEC      ; 印出。
          INC     HL          ; 指至次一文數字。
          DEC     B           ; 迴路計數值減一。
          JR      NZ,NEXT      ; 未完時繼續。
          RET

```

本章至此已經介紹過了用以溝通幾個典型輸入／輸出設備之基本程式設計技巧。除了資料傳遞外，輸入／輸出作業通常仍需存取每一設備之一個或多個控制暫存器，以界定傳輸速率，插斷型態，以及各種可選擇之參數等。

第 14 章

輸入／輸出技巧

截至目前為止，我們已經學會了如何駕馭一單獨之輸入／輸出設備。然而，在一實際系統裡，系統可能同時擁有數個輸入／輸出設備，這些設備皆同時連接至系統巴士上，因此，它們很可能在同一時間同時對微處理器提出服務請求。為能同時應付此些設備，微處理器必須對自己的時間，有所分配與安排。何時服務某一設備？何時服務另一設備？……等等。此即所謂的輸入／輸出排時（scheduling）技巧。

處理器通常有三種方法可以應付諸多輸入／輸出設備：**取樣法**、**插斷**、與**直接記憶體存取**（Direct Memory Access, 簡寫為 DMA）。**取樣法**指的是由微處理器（亦即程式）親自一一趨前詢問每一設備：“您是否需要服務？”。若是，則立即加以服務；若非，則轉而詢問次一設備。如此類推。此種方法在前面討論單一設備之輸入／輸出時曾經提過。**插斷法**指的是，微處理器平常就執行其自己的程式去了，不管輸入／輸出設備。當輸入／輸出設備需要服務時，由其個別主動通知微處理器。此種方法的優點是節省時間。微處理器平常不必浪費時間在詢問各週邊設備之狀態以及等待其變成備好狀態上。以上的兩種方法，不管取樣或插斷，真正的傳輸作業仍在微處理器的控制下進行的，微處理器涉足其中。直接記憶體存取則不然。於此一作業

進行時，微處理器暫停動作，資訊傳輸於一稱為**DMA 控制器**（DMA Controller）之特殊電路的控制下，直接發生於記憶體與輸入／輸出設備之間。圖 14-1 所示即為以上所述之三種輸入／輸出控制方式。下面我們分別介紹此每一種方式之作業情形。

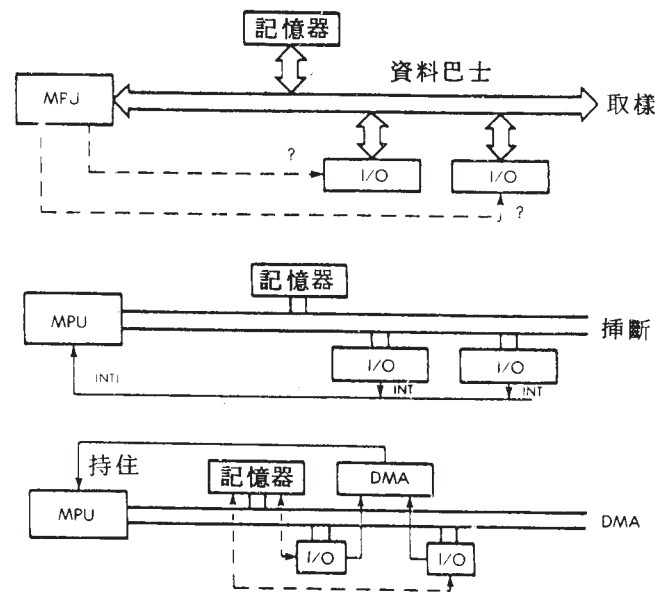


圖 14-1 三種輸入／輸出之控制方式

14-1 取樣

觀念上，取樣是駕馭多個週邊設備之最簡易方式。於此一策略，處理器輪流詢問連接至巴士之每一設備。若有某一設備需要服務，則處理器就服務那一設備。若無，則處理器繼續詢問次一設備。取樣技巧不僅用於設備，而且用於**任一設備服務常式**（device service routine）。

例如，若系統備有一電傳打字機，一磁帶錄音機，以及一 CRT 顯示器，則取樣程式就會這麼問電傳打字機：“你有文數字欲傳輸嗎

？”它會問電傳打字機之輸出常式 (output routine)， “你有一文數字要送出嗎？”。若答案為否定，則處理器繼續詢問磁帶錄音機之常式，最後 CRT 顯示。此一取樣的過程即如圖 14-2 所示。假如系統僅接有一個設備，則取樣法同樣可用以決定其是否需要服務。舉例而言，圖 14-3 與圖 14-4 所示則分別為以取樣法自一讀紙帶機輸入一項資料，以及自一印字機或打卡機輸出一項資料的流程圖。

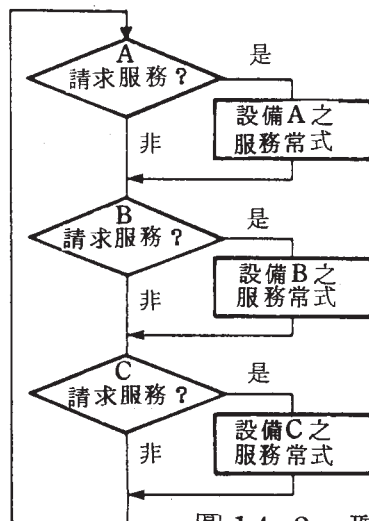


圖 14-2 取樣迴路之流程圖

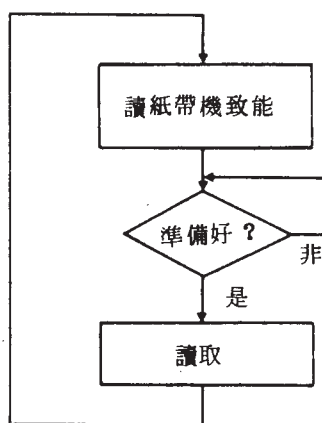


圖 14-3 自讀紙帶機輸入資料

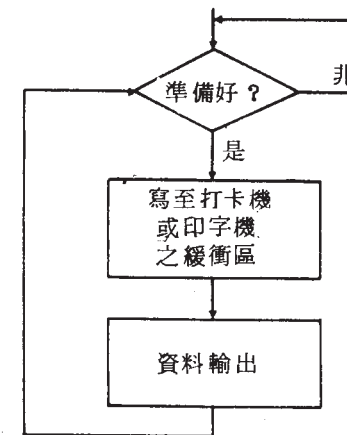


圖 14-4 自印字機或打卡機輸出資料

緊接我們舉一以取樣方式同時服務四個不同設備之例子，此一程式如例 14-1 所示，其流程圖類似圖 14-2 所示。

例 14-1 以取樣法服務四個設備之副程式

；此一常式以取樣法詢問四個設備是否需要服務，若是，則分別叫用；其所對應之設備服務常式：ONE, TWO, THREE, 與 FOUR。注；意常式使用條件副程式叫用指令。

```

;
POLL4   IN      A, (STATUS1) ; 讀取設備 1 之狀態。
        BIT      7, A          ; b7 = 1 表需要服務。
        CALL     NZ, ONE       ; 若是，則叫用服務常式。
        IN      A, (STATUS2) ; 回返後或設備 1 不需服務時
        BIT      7, A          ; 繼續詢問設備 2。
        CALL     NZ, TWO       ;
        IN      A, (STATUS3) ; 設備 3。
        BIT      7, A
  
```

```

CALL  NZ, THREE
IN     A, (STATUS4) ; 設備 4。
BIT    7, A
CALL  NZ, FOUR
JR     POLL 4       ; 繼續再試。

```

於此一程式，當每一設備需求服務時，其狀態暫存器之第 7 位元值為 1。每當程式發現設備有服務請求時，控制即跳至該設備之服務常式，設備 1 者位於位址 ONE，設備 2 者位於位址 TWO 等等。

有一點必須注意的是，每一指令影響旗號的情形非常重要。輸入指令並不改變任何旗號值。因此，倘若讀取狀態訊息時所使用的是記憶體取入指令，則輸入資料之第 7 位元將自動影響正負值旗號（S），此時，位元測試指令“BIT 7, A”即不必要。

於硬體製作時，在位址的選取上，輸入／輸出設備可視成記憶設備。此即所謂的記憶映像輸入／輸出。於此種情況，輸入作業必須以 LD 指令擔任，而非 IN 指令。同時，如以上所述地，BIT 指令亦可省略。

顯然，取樣法之優點為：簡單、毋需任何輔助硬體、以及輸入／輸出與程式作業同步。而缺點為：微處理器的時間大多數浪費在詢問一些不需求服務的設備上。此外，在浪費這麼多時間後，微處理器再對設備提供服務的時間或許已經太遲了。

因此，為了確保處理器之大部份時間能用於有用之計算，而非浪費於無價值之詢問過程中，我們就必須採用其它方式。插斷法是目前最普遍的作法了！

14-2 插 斷

14-2-1 何謂插斷

於插斷法，中央處理器與週邊設備間之傳輸作業，由週邊設備主動對中央處理器提出請求或通知後開始。當週邊設備已到達需中央處理器服務之狀態（例如，某一讀卡輸入設備已讀取一項資料，盼處理器將之取走，或某一印字輸出設備已將前一項資料印出，盼處理器再送出次一項資料）時，週邊設備主動以一信號告知中央處理器（此一信號即稱為插斷或插斷請求信號）；而非如取樣法，由中央處理器主動採取設備之狀態。

根據以上所述，在外部週邊設備“提出插斷”（送來插斷信號）之前，平常中央處理器可執行其它之程式。而當接收到插斷信號後，中央處理器才暫停原來程式之執行，並跳至一設備所屬之插斷服務常式，執行那一常式，完成週邊設備所要求之服務。完後，中央處理器再回至原來中斷之程式，自中斷點繼續再往下執行。如圖 14-5 所示，此即為插斷之溝通方式。

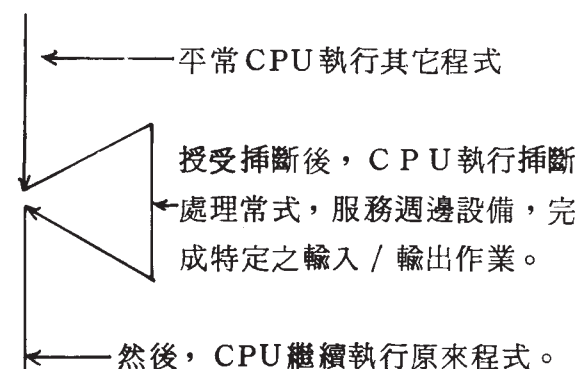


圖 14-5 插斷溝通技巧

14-2-2 插斷之用途

插斷之角色於任何微處理器或微電腦均大體相同——其通知微處理器，某一需要注意之外部事件已發生。一般而言，此一外部事件通常與輸入／輸出資料之傳遞、時序功能、或非尋常之系統狀況有關。

當插斷與輸入／輸出資料之傳遞有關時，其即為一種將 CPU 處理時間與輸入／輸出作業重疊之技巧。舉個此種插斷之例子來說。假設某一微電腦系統連接一“高速”之讀紙帶機。此部讀紙帶機每秒能讀取 500 位元組之資料。亦即，每 $1/500$ 秒或 2 毫秒就有一新位元組可讀取。若我們捨插斷法而採取樣作法輸入，則輸入作業之處理程式每 2 毫秒僅能有一 IN 指令被執行，而如圖 14-6 所示地，整個讀取動作却僅需 2.5 微秒。其它的時間，程式皆浪費於不斷地詢問（藉著執行一讀取狀態訊息之 IN 指令）紙帶機，次一位元組資料是否已備好。倘若有 500 位元組之資料欲讀取，CPU 之平均指令時間以

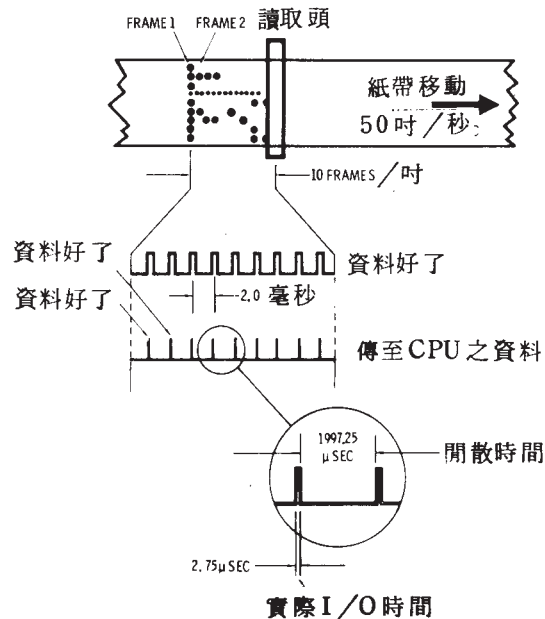


圖 14-6 輸入／輸出開散時間舉例

2.5 微秒計，處理器總共必須浪費 $(1/2.5 \times 10^{-6} - 500) = 399500$ 個指令時間，閒著不作事而只等待次一位元組資料來臨。

插斷法使得微處理器能利用與輸入／輸出活動有關之空閒時間。於插斷法，微處理器平常可執行其它之程式。當有次一位元組欲傳輸時，由週邊設備主動告知微處理器。微處理器於容許之情況下，暫停原先之作業，進行處理插斷，完成新位元組之傳輸後，再回至原來程式之執行。顯然，於插斷法，微處理器將其自己時間有效地分配運用於計算與輸入／輸出作業上，而絲毫無閒著沒事幹浪費時間之情形發生。有資料來時就讀取，無資料時就進行其它計算。若系統同時具有數個週邊設備，必須進行多種輸入／輸出活動，則**向量插斷**（vector interrupt）可用以解決此一問題。（Z 80 將 8080 之八個向量插斷的能力，擴充成 128 個個別之向量插斷。）

插斷之第二種用途是對 CPU 提供計時功能。為使 CPU 能維護一**即時時鐘**（real-time clock），以作計時（time-out）或顯示日期時間（time-of-day）之用。系統必須能對 CPU 提供某一固定長度之時限。典型上，此一時限皆經由插斷，以一大約每十分之一秒插斷 CPU 一次之**可程式化計數／計時器**（programmable counter-timer）提供。CPU 會將此一插斷視為計時器插斷，並且進入適當之插斷處理常式，調整系統時鐘且／或履行其它計時功能，然後再回至插斷前執行之程式，繼續往下執行。

插斷之第三種之用途是顯示**非尋常或災難性的系統狀況**。舉個例子而言。在停電時，電壓尚未降至某一準位以前，CPU 必須接受通知。再如，若某一即時系統之某一部份故障，CPU 亦須獲得通知。由於此等插斷必須為 CPU 所立即認知並加以處理，而無法延遲。因此，此些功能通常以**不可罩蓋**（nonmaskable）插斷實現。Z 80 即擁有此種插斷，其具備一**NMI**（Non-Maskable Interrupt）輸入接腳，以專供處理系統之非常緊急狀況之用。凡必須引起 CPU 之立即注意的外部情況，皆可經由此一特殊輸入線告知 Z 80 CPU。

14-2-3 插斷處理

若能接受外部週邊設備所提出之插斷請求，則微處理器於執行完刻在執行之指令後，即會自動跳至一相對於此一設備之**插斷服務常式**（Interrupt Service Routine，簡記為**ISR**）。微處理器然後執行此一副程式，提供設備所需之特定服務。

插斷之處理與副程式類似。舉個例子而言，若微處理器正在執行位址 0071 處之指令時，一插斷服務常式為 ISR1 之週邊設備提出插斷請求，若 ISR1 儲存於位址 EC00 起之記憶位置，則圖 14-7 所示即為微處理器接受插斷後，程式計數器及堆疊器所含之內含的情形。注意，由於 LD B, 05H 指令長兩個位元組，故次一緊接指令之起始位址 0073 被推入堆疊器存起。

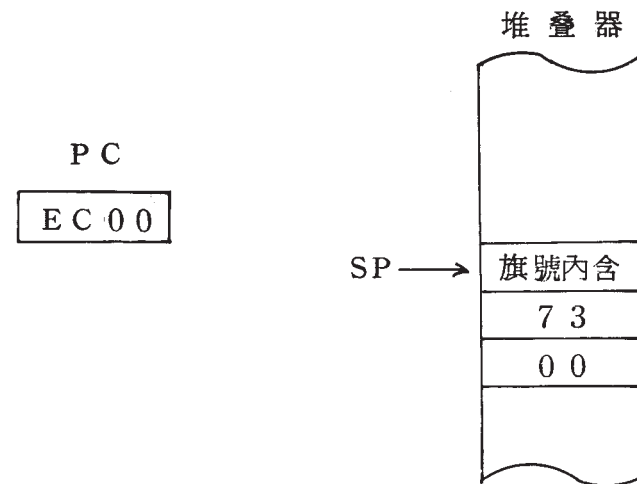
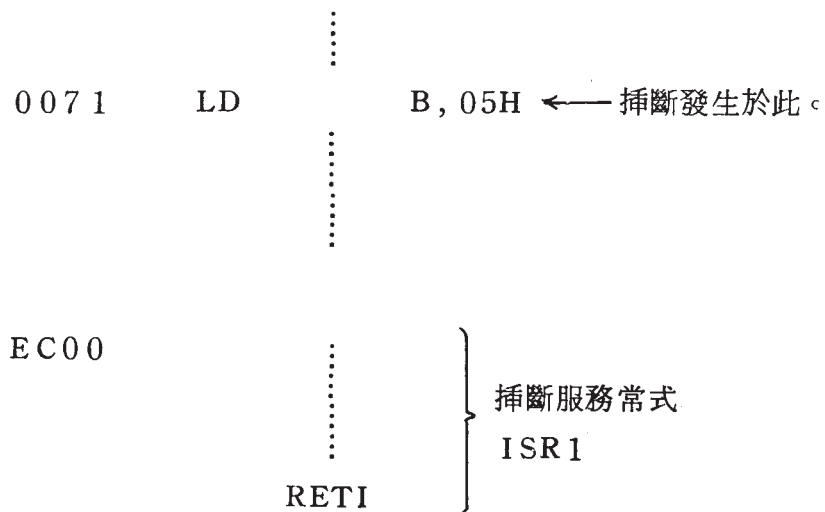


圖 14-7 微處理器接受插斷請求後

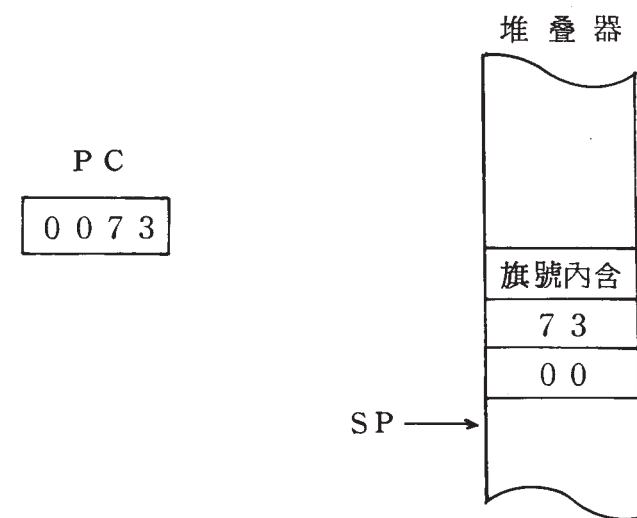


圖 14-8 微處理器自插斷回返後

插斷回返由 RETI 指令達成。正如副程式回返指令之效用一般，該指令復原程式計數器之內含，使控制回返至原來程式之插斷點。圖

14-8 所示即為前述舉例之插斷回返情形。有一點必須提醒的是，程式設計者經常需特別將服務過輸入／輸出設備之插斷清除，以及恒需將 Z 80 內部之插斷禁能旗號重新存回。不過，週邊設備之控制器亦可以 INTA 信號將插斷請求清除，以免程式設計者再來做。

暫存器內含之儲存與復原

其次，有一點必須注意的是，萬一插斷處理常式改變了任一 CPU 內部暫存器之內含，則插斷處理常式必須於常式一開始時，負責將此些暫存器之內含推入堆疊器中存起，並於回返之前又將之自堆疊器中取回，以免插斷處理常式破壞了此些暫存器之內含，並使控制回返後，原來程式能繼續正常作業。假設插斷處理常式用及 A，B，C，D，E，H，與 L 等暫存器，則插斷處理常式（設稱為 ISRO）必須如例 14-2 所示。特別注意，暫存器被推入與拉取之次序正好相反。

例 14-2 插斷處理常式中存起與恢復暫存器內含之指令；此為插斷處理常式 ISRO。常式一開始之四個指令，將常式所用及之暫存器的內含皆推入堆疊器中存起。插斷回返前，同樣有四個拉；取指令，將此些暫存器之內含恢復原狀。

```
ISRO  PUSH    AF
      PUSH    BC
      PUSH    DE
      PUSH    HL
```

```
      ⋮
      ⋮
      ⋮
```

```
POP    HL
POP    DE
POP    BC
POP    AF
RETI
```

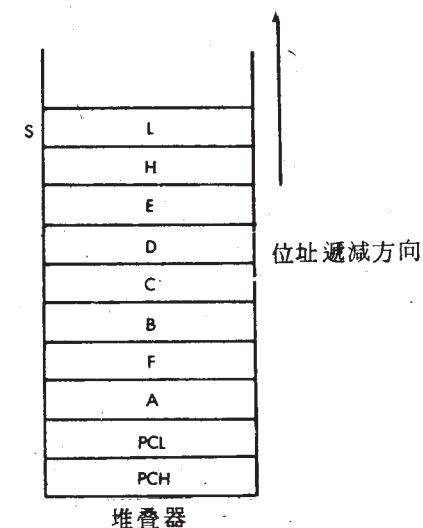


圖 14-9 存起 CPU 暫存器

如以上所述的，插斷之處理與副程式十分類似。不過，兩者仍有差別。插斷是外部設備送給微處理器的一種信號，其可在任何時刻發生。因此，並不與微處理器正在執行之程式同步。而副程式叫用則是完全在程式之控制下產生的，因此，其與程式之執行互相同步。此外，微處理器執行副程式叫用指令時，並不會自動存起狀態暫存器或程式計數器以外之其它任何暫存器的內含。

14-2-4 可罩蓋與不可罩蓋插斷

在目前之微處理器，插斷通常有兩種不同型式：可罩蓋插斷（

maskable interrupt) 或插斷請求，以及不可罩蓋插斷 (nonmaskable interrupt)。不可罩蓋插斷主要用以處理非常緊急之事件，諸如停電。無論任何情況，不可罩蓋插斷永遠能插斷微處理器，迫使其暫停原來之作業。在接收到此一輸入時，微處理器只要一完成目前正在執行之指令，即會立即加以處理。可罩蓋插斷則用於一般較不緊急之事件，此種插斷輸入僅能通知微處理器，某一外部設備正等待著您的服務，至於是否立即前往服務此該設備，微處理器自己有權決定。換言之，外部週邊設備所送給微處理器之插斷信號，僅屬於一種插斷請求。能否達成目的，還得視微處理器之狀態而定。

微處理器憑什麼拒絕或接受外部設備所提出之插斷請求呢？根據狀態暫存器之插斷致能旗號 (I)。若該旗號值為 1，則表微處理器可以接受目前之插斷請求。反之，若插斷致能旗號之值為 0，則微處理器就無法接受外部設備所提出之插斷請求。注意，有些微處理器所設置的是插斷“禁能”旗號，而非插斷“致能”旗號，故其旗號值與插斷禁致能之狀況，恰巧與上述者相反。

插斷旗號之值通常可以程式指令加以置定為 1 或清除為 0。因此，目前享有微處理器控制權（亦即微處理器正在執行）之程式，可依其是否能被插斷之狀況，適當地設定插斷旗號之值，以容許 (I = 1) 或禁止 (I = 0) 微處理器在中途接受其它之插斷請求。

14-2-5 Z80插斷之致／禁能

Z 80—CPU 亦具有上述兩種插斷輸入——軟體之可罩蓋插斷，與不可罩蓋插斷。程式設計者無法將不可罩蓋插斷 (NMI) 禁能。無論何時，只要週邊設備提出此一插斷，Z 80—CPU 即會立即接受。此一插斷輸入通常留作處理諸如停電等之最緊急事件之用。可罩蓋插斷 (INT) 則可由程式設計者將之致能或使之禁能。此一特色使程式設計能在程式無法被插斷之時限內，將所有外部之可罩蓋插斷禁能。程式設計者可分別以 EI (Enable Interrupt, 插斷致能) 或 DI

(Disable Interrupt, 插斷禁能) 指令，將 Z 80—CPU 內之一插斷致能正反器（稱為 IFF）之值設定為 1 或清除為 0。若 IFF 之值為 0，則 Z 80—CPU 就不接受進一步之插斷，所有可罩蓋插斷均遭禁能。

事實上，Z 80—CPU 內擁有兩個插斷致能正反器，分別稱為 IFF1 與 IFF2。IFF1 之狀態實際用以致／禁能插斷，而 IFF2 則僅供暫時儲存 IFF1 狀態之用。此兩正反器之詳細功用如下，



CPU 重置時，IFF1 與 IFF2 同時清除為 0，故插斷禁能。此後，程式設計者可於任意時刻，以 EI 指令將之致能。當 Z 80—CPU 執行 EI 指令時，任何懸而未決之插斷請求，都必須等到 EI 指令之次一緊接指令被執行完後，才會被接納。換言之，EI 指令之效用必須等一個指令過後才會顯現。此一延遲主要為 EI 指令後緊接即為一回返指令之情況而設。任何此種情況發生，插斷請求都必須等到回返完成後，才可被接受。EI 指令同時將 IFF1 與 IFF2 設定成致能狀態。當 CPU 接受插斷時，IFF1 與 IFF2 同時清除為零，以禁止任何進一步之插斷。必要時，程式設計者可再以一 EI 指令，將插斷重新致能。注意就以上所提之情況而言，IFF1 與 IFF2 都保持一致。

IFF2 之目的乃在當不可罩蓋插斷發生時，用以暫時儲存 IFF1 之狀態。CPU 接受不可罩蓋插斷時，IFF1 自動清除為 0，以防止 CPU 再接受任何進一步之插斷，直至程式設計者再將插斷致能之前。因此，當 Z 80—CPU 接受不可罩蓋插斷時，可罩蓋插斷被禁能。不過，IFF1 之原來狀態被存起（存入 IFF2），以便將來不可罩蓋插斷處理完成後，插斷狀態能再回至不可罩蓋插斷發生前之情況。

Z 80—CPU執行LD A, I或LD A, R指令時，IFF2之內含自動抄至狀態暫存器之極性旗號(P)，此時，程式即可將其內含值存起或加以測試。

恢復IFF1之狀態的第二種途徑是令CPU執行—RETN(不可罩蓋插斷回返)指令。由於此一指令代表不可罩蓋插斷處理常式業已完成，故IFF2之內含抄回IFF1，致使IFF1回復至不可罩蓋插斷發生前之原有狀態。

茲將插斷致能正反器IFF1與IFF2受影響的情形摘要如下：

動作	IFF1	IFF2
CPU重置	0	0
DI	0	0
EI	1	1
LD A, I	•	• IFF2→極性旗號
LD A, R	•	• IFF2→極性旗號
接受 NMI	0	•
RETN	IFF2	• IFF2→IFF1

註：“•”代表不變。

14—3 Z80之插斷系列

Z 80 總共具有三個與插斷有關之輸入：巴士請求(BUSRQ)，不可罩蓋插斷(NMI)，以及平常之可罩蓋插斷(INT)。

巴士請求輸入主要用於直接記憶體存取(DMA)。不可罩蓋插斷與可罩蓋插斷之用途區分則如前一節所述。Z 80之NMI輸入允許一個不可罩蓋插斷，而INT輸入最高則可容許128個向量式(vectored)插斷，此些插斷利用外部週邊設備或插斷邏輯所提供之寫碼。NMI輸入永遠能得Z 80微處理器之認知(或認可)。每當此一輸入線動作時，微處理器自動進入一不可罩蓋插斷之處理系列。INT輸入則唯有當Z 80微處理器處於插斷致能(interrupt enable)狀

態時，方能為微處理器所接受(認知)。此一插斷致能狀況由一程式化之正反器所提供。程式設計者可以EI及DI指令，分別將該正反器之值置定為1或清除為0，以使插斷致能或禁能。

圖14-10所示即為Z 80—CPU對上述三種插斷輸入之處理情形，如圖所示，巴士請求是Z 80具有最高優先之插斷輸入。籠統而言，直到目前正在進行之機器週期完成之前，Z 80不會感應到有任何插斷。詳細而言，Z 80微處理器要一直等到目前之指令執行完成，才會將NMI與INT插斷列入考慮。但是，對於巴士請求，只要目前之機器週期結束，微處理器即可加以處理，而不需等到目前之指令完全執行完畢。

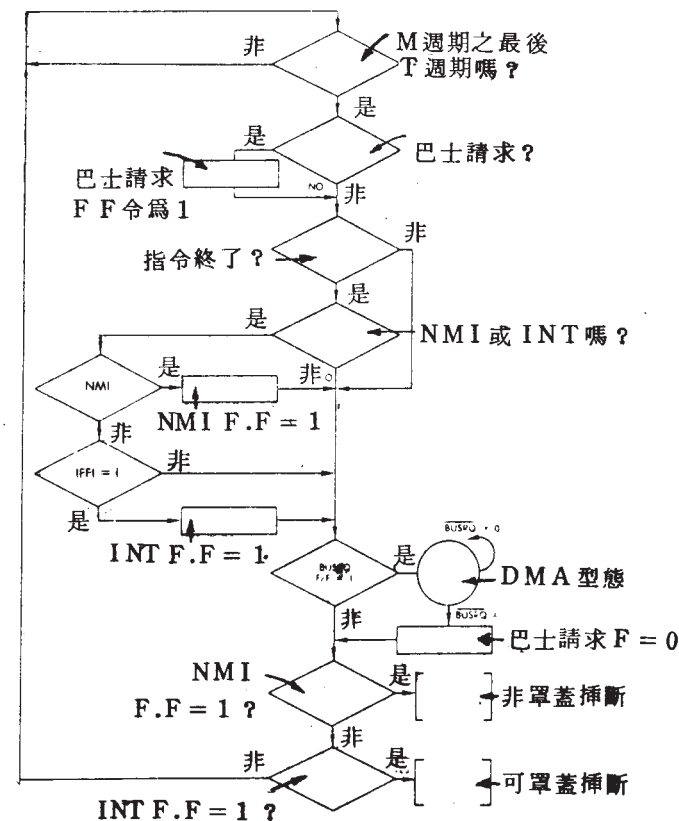


圖 14-10 Z 80之插斷系列

14-4 巴士請求

巴士請求主要用於直接記憶體存取 (DMA)。為了能直接對記憶體存取 (即讀寫) 資料, DMA 控制器必須擁有系統巴士 (包括資料巴士與位址巴士) 之控制權。巴士請求即為 DMA 控制器請求微處理器放開巴士之控制權, 以便其能進行直接記憶體存取作業所發出之信號。在察覺到巴士請求信號時, 若目前指令已經執行完畢, 而且微處理器所正在處理的是 NMI 或 INT, 則 Z80 將藉著置定兩個特殊正反器: NMI 正反器及 INT 正反器, 先記憶起此一狀態。於 DMA 作業型態下, Z80 微處理器暫停作業, 並將位址巴士與資料巴士放開——置於高阻抗狀態。此時, DMA 控制器即可利用此兩巴士, 將資料直接傳輸於記憶體與高速之輸入/輸出設備之間。DMA 作業一直進行至 BUSRQ 線之準位的改變時結束。此時, Z80 微處理器重新恢復正常之作業。微處理器將先檢查內部之 NMI 或 INT 正反器是否已被置定為 1, 若是, 則微處理器將繼續執行所對應之插斷。

對一程式設計者而言, 其所必需關心 DMA 的, 只有其時序。程式設計者只需了解 DMA 會延緩 NMI 或 INT 之反應就夠了!

14-5 不可罩蓋插斷

程式設計者無法令微處理器暫時忽視不可罩蓋插斷。無論何時, 只要無巴士請求, 一執行完目前刻在執行之指令後, 微處理器即會接受不可罩蓋插斷。若不可罩蓋插斷發生於巴士請求時, 則微處理器內部之 NMI 正反器會先置定為 1, 然後, 等巴士請求處理完後, 微處理器再進行處理不可罩蓋插斷。

當不可罩蓋插斷發生時 (NMI 輸入線變為低電位), Z80 微處理器實效上等於執行一至 0066 H 位置之重始指令。前面曾經提過, 重始指令將現有程式計數器之內含值推入堆疊器存起, 並使控制轉移至 0000 H, 0008 H, …… , 與 0038 H 等其中之一記憶位置。

NMI 之動作同於重始指令, 但控制却恒轉移至 0066 H 之記憶位置。

舉個例子而言, 圖 14-11 所示即為 Z80 微處理器處理不可罩蓋插斷之詳細過程。假若當微處理器正在執行 102AH 處之 RRCA 指令時, NMI 插斷發生。則於該指令末了, 程式計數器之現有值 102BH 被推入堆疊器中存起。然後, 控制自動轉移至位址 0066 H 處之指令。

於圖 14-11 之例子, 位址 0066 H 起之記憶位置所存的, 即為不可罩蓋插斷之處理常式。此一常式先以 EX 及 EXX 兩交換指令, 將

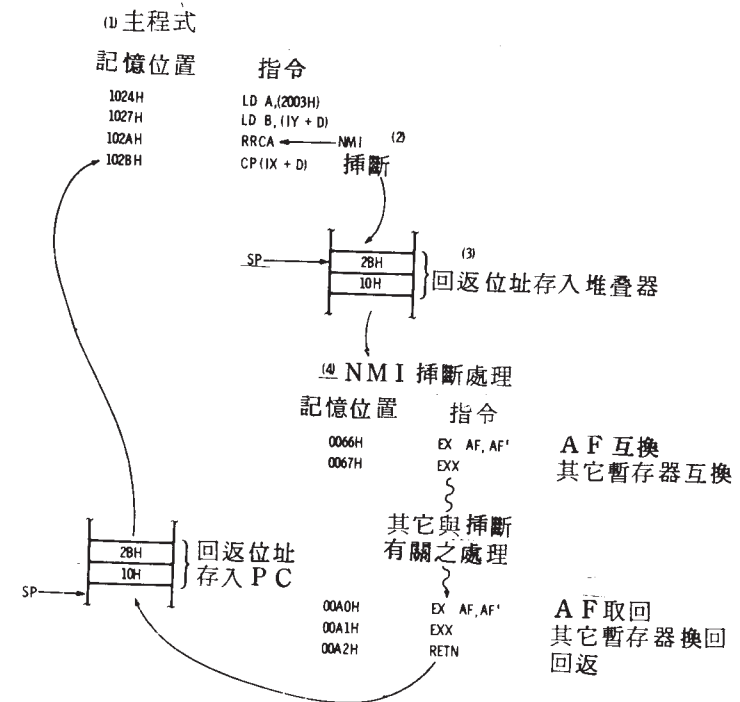


圖 14-11 NMI 插斷處理

微處理器之現有環境 (暫存器內含) 存起。然後才開始不可罩蓋插斷所需之必要處理。最後, 存起之環境再恢復原狀, 然後控制回返。

有一點值得一提的, Z80 微處理器內實際上有兩個插斷旗號, 分別稱為 IFF1 及 IFF2。IFF1 是真正用以致能或禁能可罩蓋插斷

之正反器，而 IFF2 則當 NMI 插斷發生時，用以儲存 IFF1 之狀態者。當 NMI 插斷發生時，插斷致能正反器 IFF1 之狀態自動存入 IFF2。然後，IFF1 清除為 0，使微處理器無法接受任何可罩蓋插斷。此一特色避免了不可罩蓋及可罩蓋插斷同時發生之再進入問題。

不可罩蓋插斷之回返由一特殊之 RETN 指令達成。微處理器執行此一指令時，程式計數器之內含由堆疊器中取回，同時，IFF2 之內含回存至 IFF1。此時，可罩蓋插斷之狀態（致能或禁能）又恢復 NMI 插斷發生前者。若當 NMI 插斷正在被處理期間，若程式允許外部可罩蓋插斷發生，則在存起 CPU 暫存器與旗號之內含值後，程式可以 EI 指令將插斷致能（IFF1 之值置定為 1）。不過，一般典型之應用並不如此。

上面提過，當不可罩蓋插斷發生時，Z80 微處理器恒跳至位址 0066 H 之記憶位置。於圖 14-11 之舉例，該位置之所含的即為不可罩蓋插斷之處理常式。事實上，我們亦可將不可罩蓋之插斷處理常式儲存於別處，而於位址 0066 H 之位置上置一跳越至該處理常式的指令。

由於不可罩蓋插斷主要設計以處理諸如停電或即時時鐘等緊急高優先事件，其著眼點為速度。因此，此種插斷就不比可罩蓋插斷有彈性。

14-6 可罩蓋插斷

可罩蓋插斷又稱插斷請求。於 Z80，可罩蓋插斷有三種不同之作業型態。不像 8080 只有一種。程式設計者可自由地令可罩蓋插斷生效或失效。插斷旗號 IFF1 與 IFF2 之值為“1”時，表微處理器可以接受可罩蓋插斷；反之，當此兩正反器之值為“0”時，微處理器對外部週邊設備所提之 INT 插斷請求，不予理會。IFF1 與 IFF2 每次皆同時被清除或置定。EI 指令可用以將此兩正反器之值置定為 1，而 DI 指令用以將之清除為 0。在 EI 與 DI 指令執行期間，I

NT 插斷請求自動失效，以防止失去任何資訊。

下面，我們分別探討 Z80 之三種插斷作業型態。

14-6-1 插斷型態 0

Z80 之插斷型態 0 即為 Intel 8080 微處理器所僅有之插斷作業型態。每當一起動（RESET 信號動作）時，Z80 微處理器即自動設定於此一插斷作業型態。此外，執行 IM 0 指令亦可使 Z80 微處理器設定於此一插斷型態。

一旦被置定於插斷型態 0，只要插斷致能正反器 IFF1 被置定為 1，而且無巴士請求或不可罩蓋插斷同時發生，Z80 微處理器即會接受外部設備所提出之插斷請求，並產生下列之動作：

1. 插斷請求發生（ $\overline{\text{INT}}$ 線輸入低電位）。
2. 於現有指令執行完後，微處理器測知插斷請求。
3. 微處理器同時送出 $\overline{\text{IORQ}}$ 與 $\overline{\text{M1}}$ 信號，表示接受插斷請求。
4. 外部設備認知 $\overline{\text{IORQ}}$ 及 $\overline{\text{M1}}$ 信號後，輸出一重始指令至資料巴士。
5. Z80 微處理器自資料巴士攫取重始指令，並加以執行。控制轉移至重始指令所指之零頁位置上。
6. 微處理器緊接執行插斷處理常式之指令。
7. 插斷處理常式之最後指令 RETI，使控制回返至原來插斷發生前之程式，繼續往下執行。

插斷型態 0 之處理極類似於不可罩蓋插斷。兩者皆執行一重始指令，控制皆轉移至一零頁位置，並且兩者於插斷處理末了均需要一回返指令。我們舉一插斷型態 0 之例子加以說明。圖 14-12 所示即為一具有插斷能力之讀紙帶機控制器，提出插斷請求後所發生之一系動作。

如圖所示，於讀取一資料位元組後，紙帶機控制器使 $\overline{\text{INT}}$ 線動作——變成低電位。當微處理器於指令末了測知插斷後， $\overline{\text{IORQ}}$ 及 $\overline{\text{M1}}$ 線變為低電位。紙帶機控制器將此兩信號釋為插斷認知，並將一

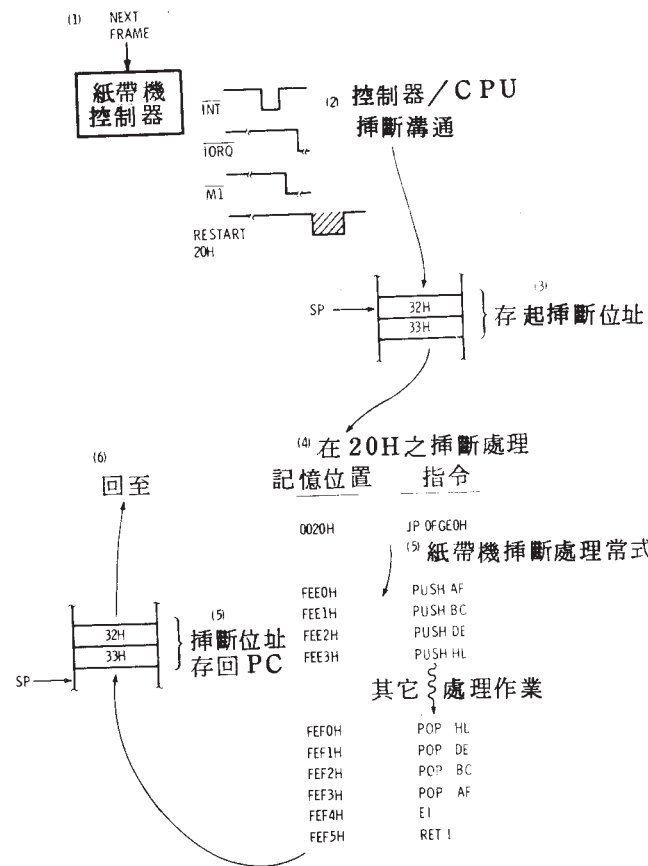


圖 14-12 型態 0 之插斷處理作業

RST 20 H 指令置於資料巴士上。Z80於讀取該指令後立即開始執行，首先將現有程式計數器之內含（假設為 3332 H）推入堆疊器中存起。然後，控制轉移至位址 20 H 之零頁位置。20 H 位置上為一 JP FEE0H 指令，微處理器執行此一指令的結果，控制又轉移至 FEE0 位置起之讀紙帶機插斷處理常式。（注意，微處理器於接受可罩蓋插斷後，插斷自動禁能，此一狀態一直維持至 Z80 再執行一 EI 指令為止。）

讀紙帶機之插斷處理常式先以一系列之推入指令，將各 CPU 暫

存器與旗號之內含推入堆疊器中存起（另一種儲存此些暫存器內含之辦法，為使用 EX 與 EXX 指令，將兩組暫存器之內含互換）。然後，常式開始進行讀取文數字資料，清除紙帶機控制器之插斷狀態，與處理資料（一般而言）等作業。插斷處理末了，相繼之拉取（POP）指令將先前存起之暫存器內含，依相反順序一一復原。然後，EI 將插斷致能，以備讀取次一文數字時能再提出插斷。最後，RETI 指令使控制回返至 3332 H 之位置。

以上的例子僅考慮唯一插斷設備：讀紙帶機控制器。事實上，Z80 插斷系列之型態 0 與型態 2，均容許有多個插斷設備。當同時有數個設備可提出插斷時，系統就必須有某種決定各設備之優先順序（priority）的辦法，以避免兩個或兩個以上之設備，同時循同一條插斷線提出插斷請求。否則，就有困擾發生。因為，每一設備都會認為其已得到插斷認可。

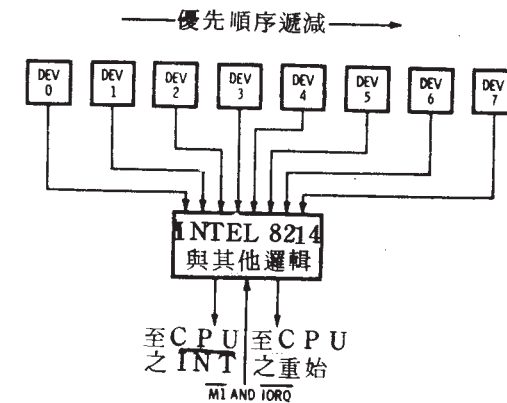


圖 14-13 插斷型態 0 之優先順序寫碼

於定優先次序法，每一外部設備均指定有一由高至低之優先順序。此一情形如圖 14-13 所示。圖中之八個設備，其優先次序由左至右遞減，最左邊者具有最高優先。每一設備均連接至一插斷優先順序控制單元（Intel 8214）。此一單元與其有關之電路，使得每一次僅

有單一設備能獲致插斷。若同時有數個插斷請求發生，此控制單元將決定一具有最高優先順序之請求，使 $\overline{\text{INT}}$ 變低電位，並於插斷認知後將適當之重始指令置於資料巴士上。

當微處理器正在服務某一插斷設備時，其它具有更高優先順序之設備可能又會提出插斷請求。此時，控制單元將使微處理器轉而先服務更高優先之設備。雖然於某些關鍵時刻，諸如正在儲存 CPU “環境” 時，插斷控制正反器 IFF1 可適當地用以防止其它設備於此時提出插斷，但微處理器同樣可毫無衝突地以巢串之方式，處理數個插斷。優先順序之進一步例子將於插斷型態 2 後繼續討論。

14-6-2 插斷型態 1

Z80 微處理器執行 IM1 指令的結果，將插斷作業型態置定於型態 1。此一插斷作業型態屬於一種自動插斷處理，其使控制自動跳至位址 0038 H 之記憶位置。因此，除了可被罩蓋，以及控制自動轉移至 0038 H 而非 0066 H 之記憶位置外，插斷型態 1 主要類似於 NMI 插斷。圖 14-14 所示即為 Z80 插斷型態 1 之自動指向的情形。如圖

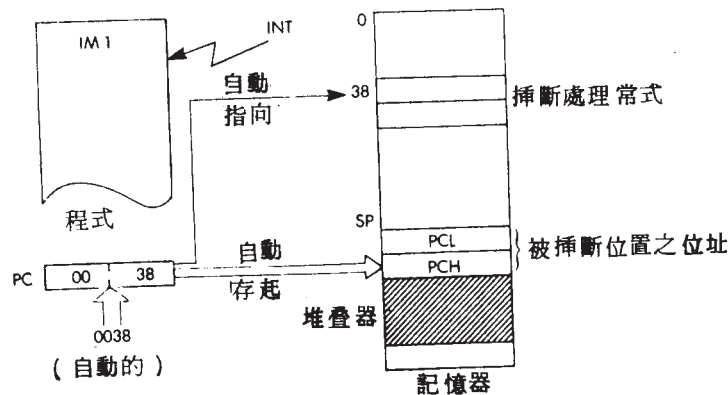


圖 14-14 插斷作業型態 1

中所示，Z80 自動將程式計數器之內含存入堆疊器。

此一恒將所有插斷皆“指向” 0038 H 之記憶位置的自動插斷反應，將使用插斷所需之外部硬體數量減至最低程度。正如 NMI 插斷一般，插斷型態 1 之優點為，將重始指令置於資料巴士之作業不需額外之電路。而其唯一可能存在之缺點為，控制僅跳至單一記憶位置。倘若插斷線上連有數個設備，位址 0038 H 開始之程式必須負責決定究竟為那一設備提出插斷請求。此一問題將於稍後討論。

對於此一插斷型態之時序有一點必須注意的是：於程式化輸入／輸出傳輸，Z80 將忽略插斷後緊接之週期內（亦即為插斷認知週期），出現在資料巴士上之任何資料。

14-6-3 插斷型態 2 (向量式插斷)

插斷型態 2 為最後一種，亦是最能幹的一種插斷作業型態。此一型態最高能處理 128 個來自外部設備之插斷。每一插斷均可以一十六位元之位址，指向儲存於記憶器中任一記憶位置之插斷處理常式。此外，Zilog 族中之週邊界面電路，諸如 Z80 PIO（並行輸入／輸出）、Z80—SIO（串行輸入／輸出）、以及 Z80—CTC（計數器—計時器電路），皆能輕易地以雛菊花環（daisy-Chain）之連接方式，使所有之插斷設備皆完全具有優先順序。

插斷型態 2 於 Z80 微處理器執行 IM2 指令後置定。此一插斷型態之重心為一能儲存於記憶器任意位置之插斷向量表格。（插斷向量乃一項用以指至插斷處理常式之始的位址。）該表格一般之長度為（ $2 \times N$ ）個位元組，其中 N 為系統中插斷設備之數目。表格之起點即為 IIIIIIII00000000 ，所指之記憶位置，其中 I 之部份為插斷向量暫存器 I 之內含。如圖 14-15 所示，對任何插斷而言，I 暫存器皆提供表格位址之最高次八位元，而表格位址之低次八位元則由插斷設備提供。由於每一插斷向量均為兩位元組長，致插斷向量指示器之最低次位元恒為 0。故每一插斷設備實際上僅需提供了七位元之位址。

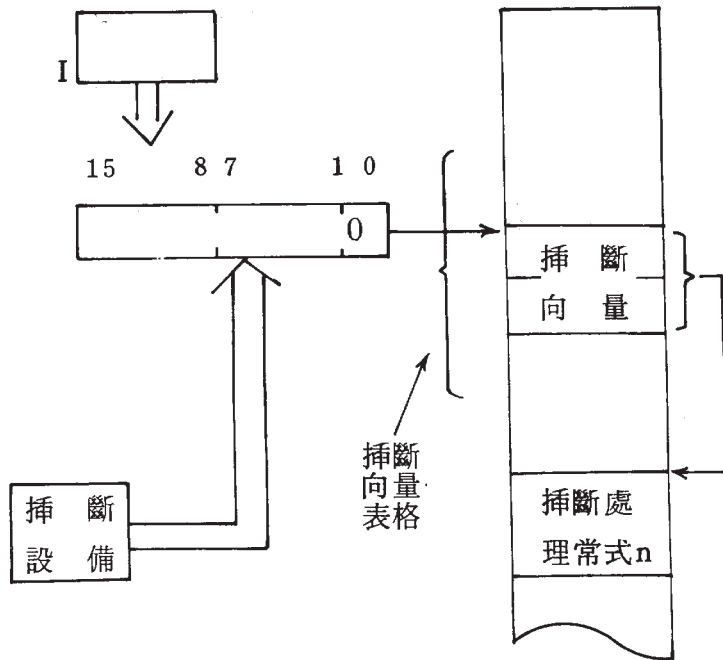


圖 14-15 插斷型態 2 之自動指向

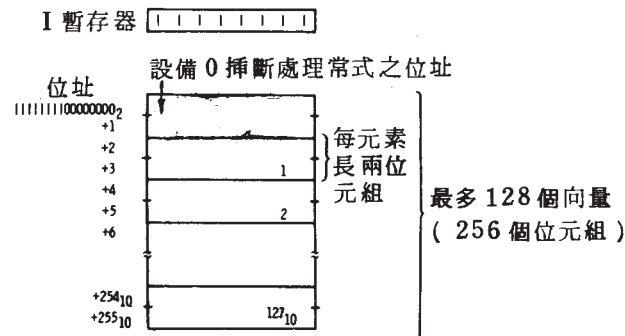


圖 14-16 插斷型態 2 之插斷向量表格

圖 14-16 所示即為圖 14-15 中之插斷向量表格的詳細圖。如圖所示，此一表格最高可含 128 個元素（插斷向量）。每一元素皆長兩位元組，且代表某一特定週邊設備之插斷處理常式的起始位址。

插斷型態 2 之一般動作系列如下：

- 1 若為插斷型態 2，且 $IFF1 = 1$ 及 \overline{INT} 動作，則 Z80 微處理器將於次一 M1 週期時認知插斷。
- 2 插斷設備對插斷認知之反應為送出一八位元數值（事實上為七位元）。
- 3 此一八位元數值然後與 I 暫存器之內含合成一十六位元之記憶位址。該記憶位址即指至對應插斷設備之插斷向量所在（表格中）之位置。
- 4 程式計數器之內含被推入堆疊器中存起。
- 5 微處理器以步驟 3 所得之位址，自插斷向量表格中拿取插斷設備所對應之插斷向量。該插斷向量即為對應插斷設備之一插斷處理常式的起始位址。
- 6 取得之插斷向量存入程式計數器，自此，控制轉移至插斷處理常式。
- 7 插斷處理常式執行完後，RETI 指令使先前存起之程式計數器內含復原，因而控制回返至被插斷之程式。

圖 14-17 所示即為此過程之一例子。於例中，插斷向量表格座落於位址 F000H 起之記憶位置。表格中含十個雙位元組之元素，每一元素指至一不同之插斷處理常式。注意，有兩項元素完全相同，這代表此兩設備所需求之插斷服務完全一致。

插斷向量暫存器原先即存 F0H。當第 5 號設備提出插斷請求，且微處理器認知插斷後，設備控制器即將一八位元向量（在此例為 02H）置於資料巴士上。該向量與 I 暫存器之內含合成一十六位元記憶位址——F002H。於將程式計數器內含推入堆疊器中存起後，微處理器即根據此一位址，自位址 F002H 及 F003H 兩記憶位置讀取

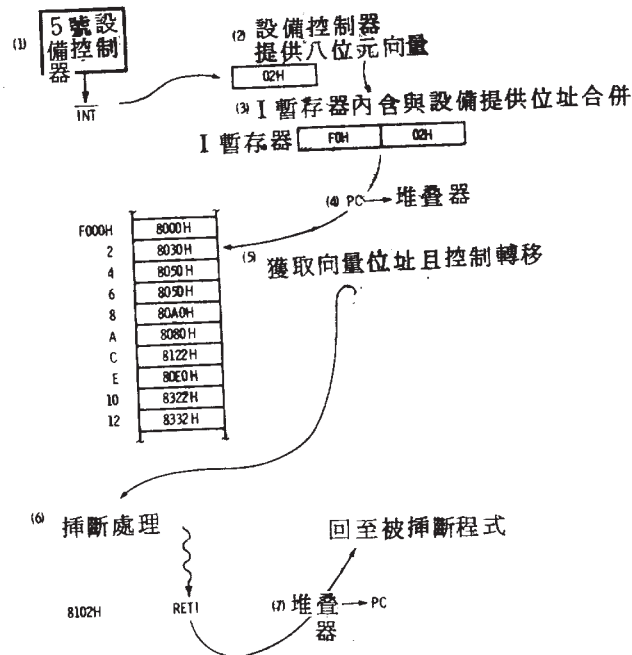


圖 14-17 插斷型態 2 舉例

插斷處理常式之入口位址（在例中為 8030H），並令控制轉移至插斷處理常式。此一反應過程總共需 19 個時序週期之時間完成：自插斷設備拿取低次位址位元組需時 7 個週期，存起程式計數器之內含需 6 個週期，而所剩 6 個週期為獲取跳越位址（即插斷處理常式之起始位址）。

控制轉移至處理常式後，微處理器開始履行插斷服務。最後，微處理器執行 RETI 指令，結束插斷作業，並將回返位址自堆疊器取回至程式計數器，令控制回返至原先被插斷之位置。

注意到，雖然使設備 0 之表格元素位於 F000H，設備 1 所對應之表格元素位於 F002H 等位置之作法較為方便，但插斷設備仍可提供任意八位元之資料作為向量，而不必像 IN 及 OUT 指令之情況，一定非提供設備所獨有之位址不可。此外，每一插斷設備實際上僅供應

了七位元之位址。若表格起始位址之最低次八位元正巧為 00H，則插斷所需供應之向量正巧為 2N。

可罩蓋插斷摘要

圖 14-18 摘要了 Z 80 之可罩蓋插斷的三種作業型態：型態 0，型態 1，與型態 2。如圖所示，一旦插斷處理作業開始，任何進一步之插斷皆自動失效。IFF1 及 IFF2 自動清除為 0。因此，是否再將插斷致能，就是程式設計者的事了。若希望於插斷回返前將插斷致能（即容許被一步被插斷），則程式可於適當之位置加上一 EI（Enable Interrupt，插斷致能）指令。

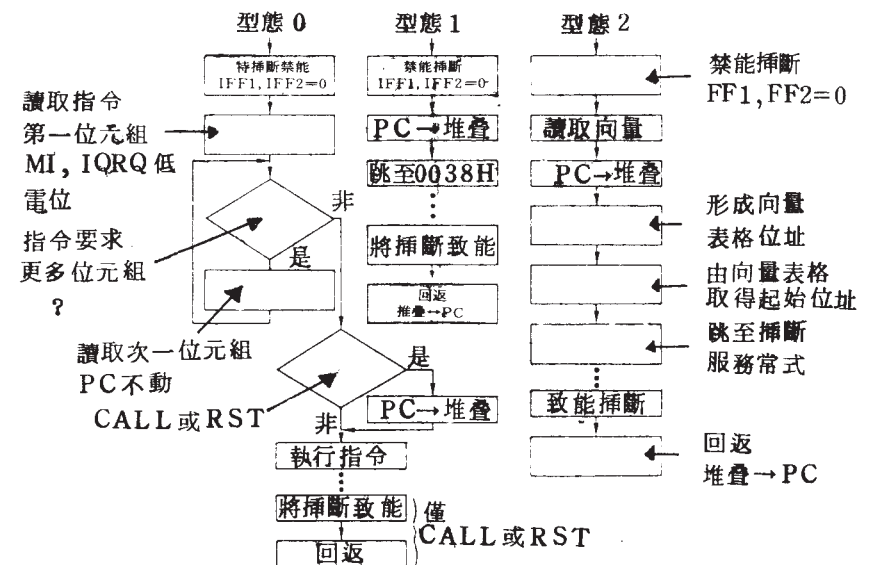


圖 14-18 Z80 可罩蓋插斷之三種作業型態

14-6-4 多個設備共用一插斷線

由 Z 80 微處理器僅具有一平常之插斷輸入線，因此，若系統具

有多個輸入／輸出設備，此些設備必須一同連接至此唯一之插斷輸入線 $\overline{\text{INT}}$ ，如圖 14-19 所示。如此，每當 $\overline{\text{INT}}$ 輸入線動作（變為低電位），即表連接至此一輸入線之所有設備中，其中有一個或一個以上之設備需求服務。在能開始作任何有效處理之前，Z80 微處理器必須先完成兩件工作：第一，其必須先知道究竟是那一設備提出插斷。第二，若有兩個或兩個以上之設備同時提出插斷，其必須決定先服務那一設備。有兩種方法可以完成上述工作：軟體法與硬體法。

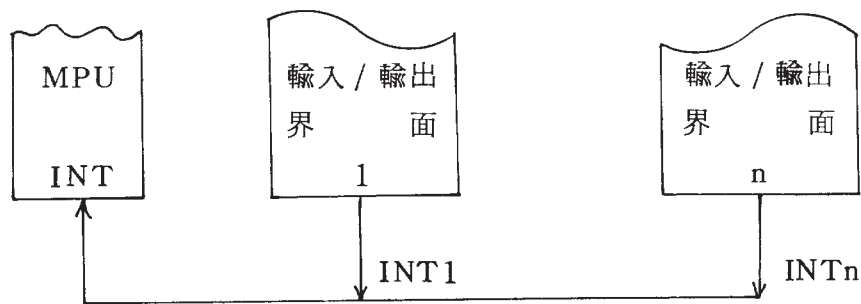


圖 14-19 多個設備使用同一插斷線

軟體方式

軟體法就是採用詢問法：微處理器輪流詢問每一設備，“是你提出插斷嗎？”若答案否定，則再問下一個。圖 14-20 所示的即為此一過程。其程式如下：

例 14-3 一般性之插斷處理常式

```

POLINT  IN  A, (STATUS1) ; 是第 1 設備提出插斷請求
        BIT  7, A          ; 嗎？
        JP   NZ, ONE       ; 若是，則跳至其處理常式
                                ; ONE。
        IN   A, (STATUS2) ; 若非，那是第 2 設備嗎？
        BIT  7, A

```

```

JP      NZ, TWO           ; 若是，則跳至其插斷處理
        ; 常式 TWO。
        ⋮

```

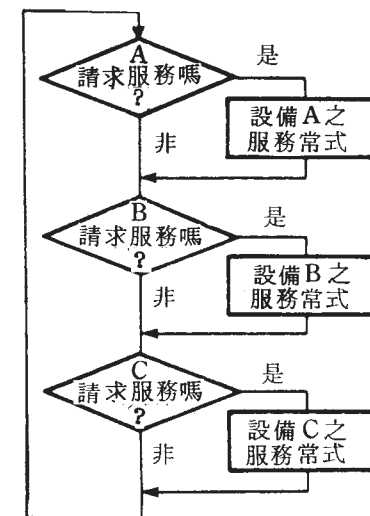


圖 14-20 以詢問（軟體）法找出該服務之設備

換言之，於測知有外部設備提出插斷後，微處理器先進入一段一般之插斷處理常式，以找出提出插斷請求之設備，然後再跳去該設備之插斷處理常式上，如上例中之 ONE, TWO, ……等。特別注意，於執行例 14-3 所示之一般插斷處理常式過程中，事實上微處理器已經同時完成了前面所列舉的兩項工作。第一項工作已經很顯明，若某一設備提出插斷，其狀態暫存器之第 7 位元必為 1，故微處理器於讀取狀態字組，並加以測試後，必可找到提出插斷的設備。第二，例 14-3 之程式顯然已將所有連接至插斷輸入線之設備，建立起一套優先順序：設備 1 具最高優先，設備 2 次之，……。假若設備 1 與設備 2 同時

提出插斷請求，由於程式先詢問設備 1，故設備 1 將先獲得微處理器之服務。這順利地解決了同時提出插斷時，究竟微處理器應先服務那一設備的問題。

硬體方式

硬體的方式是使用額外的電路，使插斷設備於提出插斷時順便提供設備處理常式之位址，致微處理器於接受插斷後能直接進入插斷處理常式。換言之，此即為所謂的**向量式插斷**。此種方法雖然成本較高，但反應速度却較迅速。圖 14-21 所示即為詢問式插斷與向量式插斷的比較。

目前而言，提供以上設施之電路即稱為**插斷優先順序控制器**（Priority Interrupt Controller，簡記為**PIC**）。此一電路能自動將插斷設備真正所需之跳越位址，置於資料巴士上。更明確言，當動作於插斷型態 0 時，PIC 對插斷認知之反應就是將一單位元組之 RST 或一三位元組之 CALL 指令，置於資料巴士上。此一自動插斷指向，將所需浪費之虛功減至最低程度。

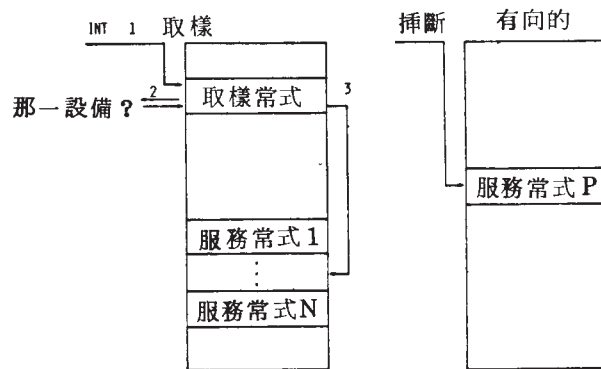


圖 14-21 詢問式與向量式插斷

以上所述僅解決了找到插斷設備（處理常式）之問題。若同時有兩個或兩個以上之設備提出插斷，怎麼辦呢？Z 80 所採用之硬體決定優先順序法，即為有名之**雛菊花環法**（daisy-chain）。

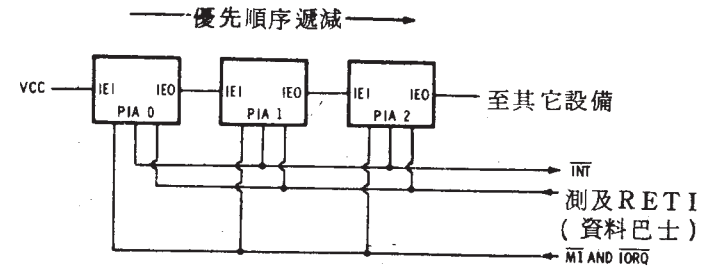


圖 14-22 Z80 插斷型態 2 之優先順序決定

如圖 14-22 所示，於 Z 80 週邊設備所使用之優先順序法，每一設備均具有一隱含之優先順序。在圖中，每一設備即為一 Z 80 PIO。每一 Z 80 PIO 均有一特別為插斷型態 2 作業所設計之“內含”插斷排序電路。所有 PIO 均以“結合 OR”（wired-OR）之結構（ \overline{INT} 直接連接，無電路作緩衝。）連接至 CPU 之 \overline{INT} 輸入線。若來自較高優先順序設備的 IEI（插斷致能輸入）信號為高電位（正），則表沒有任何具有較高優先順序之設備提出插斷請求，因此，此 PIO 可產生插斷請求。於產生插斷請求前，此 PIO 之 IEO（插斷致能輸出）先變為低電位，以顯示此時所有較低優先之設備均不可產生插斷請求而將 \overline{INT} 線變為低電位。當微處理器插斷認知發生後，PIO 即自動將一八位元向量“塞入”資料巴士，使控制轉移至適當之記憶位置，開始插斷服務常式。直至微處理器碰到 RETI 指令後，目前 PIO 之插斷服務結束。此 PIO 之 IEO 輸出由低電位變為高電位，允許其後較低優先之設備能開始提出插斷請求。

14-6-5 巢串插斷

以上所述之定優先順序法，不僅能解決數個設備同時提出插斷之問題，同時亦容許多層次之插斷巢串。現在我們看看巢串插斷到底是怎麼一回事，以及如何以堆疊器解決此一問題。前面曾一直提過，堆疊器的最大用途之一就是用於插斷之處理，現在已到澄清此一謎題的時候了。

若微處理器正在執行之插斷處理常式容許自己被進一步插斷，則上述之多重插斷的情形就會發生。一般而言，只要系統具有堆疊記憶設施，多重插斷即不成問題。讓我們以圖 14-23，進一步說明如何處置此一狀況。

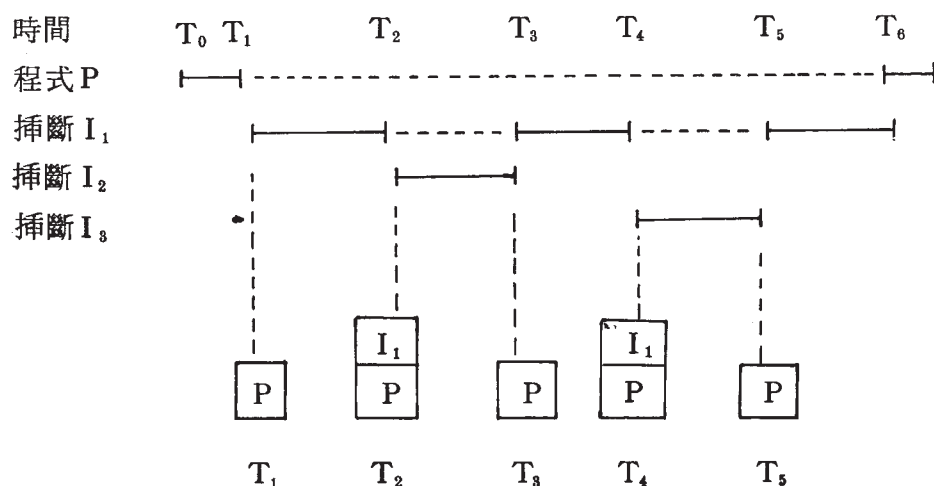


圖 14-23 多重插斷時，堆疊內含之變化情形

圖之上半為時間流失過程中，插斷發生的情形，時間之起點位於最左邊。而下半部則為堆疊器內含之變化情形。由最左邊開始，假設時間 T₀ 時，微處理器正執行程式 P。時間 T₁，插斷 I₁ 發生。假設此

時插斷被致能，容許插斷發生，則微處理器將暫緩程式 P 之執行，並接受 I₁ 插斷。如圖之下方所示地，此時程式 P 之程式計數器、狀態暫存器、以及其它任何會受 I₁ 之插斷處理常式所影響之暫存器的內含，被堆入堆疊器中存起，此些暫存器的內含，在圖中統一以 P 表示。

T₁ 時間後，I₁ 插斷開始執行，直至 T₂ 時，插斷 I₂ 又發生。若 I₂ 之優先次序較低（換言之，此時 I₁ 插斷處理常式將插斷禁能），則微處理器將暫時忽略它，直至 I₁ 執行完為止。若插斷 I₂ 比插斷 I₁ 具有較高優先（此例假設如此），則於 T₂ 時，I₁ 之“環境”（即各暫存器之內含，包括程式計數器與狀態暫存器）亦被堆入堆疊器中存起。微處理器然後開始執行插斷 I₂。

假設 I₂ 沒有進一步被插斷，並於時間 T₃ 時執行完成。I₂ 一結束，其最後一個被執行的指令必為插斷回返指令 RETI，該指令自動將堆疊器中插斷 I₁ 之暫存器的內含，自堆疊器提取，存回 Z 80 微處理器。結果，程式計數器的內含復原，微處理器又回至 I₁ 插斷處理常式中，先前離開之處，往下繼續執行其未完之部份。

倘若時間 T₄ 時，I₁ 尚在執行之刻，一更高優先之插斷 I₃ 又發生。此時，I₁ 之現有環境再度被堆入堆疊器中存起，並且微處理器開始執行插斷 I₃。假設 I₃ 執行過程中無更高優先之插斷發生，並於 T₅ 時刻執行完畢。T₅ 一執行完畢，微處理器自動自堆疊器取回 I₁ 之環境，並繼續執行 I₁。若 I₁ 無進一步被插斷，並於 T₆ 時執行完畢。則 T₆ 時，程式 P 之存起暫存器內含重新復原，程式執行繼續。

由以上之敘述可看出，解決兩個或兩個以上插斷同時發生之問題，必須具備兩個條件：第一，各插斷之間必須建立起一套優先次序，以決定誰能先接受微處理器之服務。此一優先次序由插斷處理常式自行建立。其可視情況需要，於適當時刻以 DI 將外部插斷禁能，若 DI 指令置於插斷處理程式之最開端，則表該設備具有最高之優先。反之，在插斷處理常式中皆未將插斷禁能之設備，代表其具最後優先，因任何人都可將之插斷，搶先被執行。第二、系統必須具有堆疊記憶。

此一設施除了具備容易存取之特色外，最主要的是其天賦之“後進先出”特性。此一特性使插斷之作業，能依“正確”的次序恢復執行。

習題 14-1：假設每次插斷時，所有暫存器的內含均須存入堆疊器，試問 Z 80 微處理器最多能同時處理幾個插斷？（假設堆疊位置限於 300 個。）

不過，必須強調的是，實用上，通常微處理器系統所連接之使用插斷的設備，為數並不多。因此，不太可能有許多插斷同時發生之情況產生。

14-6-6 與插斷有關之 Z 80 指令

與 Z 80 系統作業有關之插斷作業，有點像瑣碎之輸入／輸出作業——不針對某一特定系統，而祇談一般狀況實在很難。與插斷作業有關之 Z 80—CPU 指令計有 IM 0，IM 1，IM 2，EI，DI，RETI 與 RETN 等七個。前三者用以選定 Z 80—CPU 之插斷反應型態，這與系統如何配組以作插斷反應有關。前面說過，於執行過 IM 指令後，Z 80—CPU 自動設定於該指令所指明之插斷作業型態上。由於系統中每一設備控制器或 PIO 均擁有一各自之位址與硬體接好之插斷反應系列，是故，一已知系統很可能僅動作於三種作業型態之其中一種。

DI 與 EI 指令主要用以將插斷禁能或致能，以禁止或允許可罩蓋之外部插斷。當插斷致能正反器之值為 1，插斷請求將引起一與選定型態有關之插斷動作。最後，使控制轉移至一每一種插斷所獨有之插斷處理常式。

DI 與 EI 指令主要是用於插斷處理常式。前面說過，於插斷處理常式，首要的工作就是確保在插斷處理常式內所用之暫存器與旗號內含存起前，其它所有插斷均禁能。於 Z 80，除非 Z 80—CPU 再執

行一 EI 指令，否則，一插斷後，所有其它插斷均遭禁能。因此，一插斷處理常式之開頭常為：

PTAPE	EQU	\$	；紙帶機插斷處理常式。
此一系列動作過程中，任何可罩蓋插斷均不准發生。	PUSH	AF	將 CPU 所有暫存器與旗號值存起。
	PUSH	BC	
	PUSH	DE	
	PUSH	HL	
	PUSH	IX	
	PUSH	IY	
	EI		←將插斷致能。
	⋮		

於此，所有 Z 80—CPU 內部之暫存器與旗號值均推入堆疊器中存起，顯示插斷處理非常廣泛，且很可能用及所有的 CPU 暫存器。倘若插斷處理相當有限，且已知某些暫存器必用不上，那這些暫存器內含就可不必貯儲。不過，由於在插斷處理過程中，旗號將如何受影響根本無從想像，故 PUSH AF（或 EX AF, AF'）指令一定要有。一旦 CPU 環境存起後，更高優先之插斷設備此時即可自由提出插斷，並接受處理。

除了在極良好之系統，插斷處置常被設計得比系統其它工作相當簡短外，一般而言，插斷處理實際上是無所限制的。於某一插斷之插斷處理常式末了，SP 會指至最後一個存起之暫存器對，在以 POP（或 EX）指令將此些環境取回後，控制即可以 RETI（自可罩蓋插斷回返）或 RETN（自不可罩蓋插斷回返）指令回返至原先被插斷之位置。

此一系列過程中，任何可罩蓋插斷均不可發生。

RETURN	DI	← 暫將插斷禁能。
POP	IY	} 取回收存起之環境。
POP	IX	
POP	HL	
POP	DE	
POP	BC	
POP	AF	
EI		← 使插斷恢復致能。
RETI		

特別注意，DI 與 EI 兩指令將所有取回環境之指令包起，以防止在此一過程中有任何可罩蓋插斷發生。EI 指令執行之結果必須再等一個指令過後才會見效，因此，確保插斷回返能順利完成。

由於不可罩蓋插斷 (NMI) 可於任何時刻發生，故其可發生於可罩蓋插斷禁能且插斷處理常式正在存起或取回暫存器內含之時，或再進入之強迫“門禁”(lock-out)之時。不過，若 NMI 插斷常式避免使用堆疊器，且使用一不同之記憶區儲存環境，則順序仍能維持。於處理作業完畢，RETN 指令執行過後，先前可罩蓋插斷之狀態再度恢復，處理繼續。不過，由於 NMI 經常是用以處置系統之災難狀況，因此，在 NMI 常式之後，系統之有效性是否依然存在，誠屬令人懷疑。

14—7 直接記憶器存取 (DMA)

於直接記憶器存取 (DMA)，設備控制器之硬體電路，經由 $\overline{\text{BUSRQ}}$ (巴士請求) 與 $\overline{\text{BUSAR}}$ (巴士認知) 兩信號自動與中央處理器溝通。當需要 DMA 傳輸時，DMA 控制器對中央處理器發出巴士請求信號。處理器於接受請求並放開巴士控制權時，以巴士認知信號告知 DMA 控制器。DMA 傳輸然後可進行。此時，處理器暫停作

業，資料於此一週期竊取 (cycle-stealing) 期間，直接傳輸於週邊設備與記憶器之間。

DMA 傳輸之前，DMA 控制器必須先起始，雖然送至界面電路之命令 (command) 格式因系統而異，但典型的情況包括送出緩衝區位址，位元組計數，以及一界定 DMA 之特性 (讀或寫，以及其它輔助之功能) 之命令等系列。下面即是一例。

LD	C, 30H	; 磁碟緩衝區口位址。
LD	A, COMAND	; 起動命令。
OUT	(C), A	; 起動 DMA 裝置。
LD	DE, BYTES	; DMA 傳輸之位元組數。
LD	BC, BUFFER	; 緩衝區位址。
OUT	(C), B	;
OUT	(C), C	; 輸出至磁碟緩衝暫存器。
OUT	(C), D	
OUT	(C), E	; 輸出至磁碟位元組暫存器。
LOOP	IN A, (31H)	; 讀取 DMA 之狀態。
BIT	0, A	; 測試忙碌狀態。
JP	Z, LOOP	; 若忙碌，則再等。
	...	

於上述假想 (但代表性之) 系列，緩衝區位址及位元組計數值以一事先定義好之四位元組系列，輸出至位址 30 H 之輸入 / 輸出口。位址 30 H 之輸入 / 輸出口假設為一軟性磁碟控制器之 DMA 控制暫存器的位址。當控制器接收到“起動 DMA 裝置”之命令時，其期望著緊接至 30 H 之四位元組輸出，即為定義 DMA 位址的兩位元組，以及指明欲傳輸之位元組數的兩位元組。於收到這四個位元組後，D

MA 動作立即開始。此時，系列可由第二個輸入 / 輸出口位址 (31H)，讀取 DMA 之狀態。標題 LOOP 以下之迴路，即為平常所熟悉，於 DMA 活動期間不斷地檢查備好狀態之等待迴路。

由於 DMA 傳輸皆為自動完成，與用者程式無關的。因此，遭兇之指令可能並非等待 DMA 之完成，而是等待磁碟控制器所需之最後清除動作，並回至叫用程式。叫用程式然後可作些額外之處理，一直等到下一次與軟性磁碟之 DMA 有關的插斷再發生為止。

摘 要

此章，我們介紹了如何以取樣 (詢問)、插斷、以及直接記憶體存取 (DMA) 等技巧，控制多個週邊設備之作業。前兩種方式是屬於由微處理器主導 (介入) 之方式，其中，取樣法又是完全軟體化之輸入 / 輸出技巧，因此，其又稱為**程式控制式輸入 / 輸出** (program - controlled I/O)。插斷法則屬於一種硬體化之自動技巧。因此，其速度較快。

Z 80 微處理器具有不可罩蓋插斷 (NMI) 與平常之可罩蓋插斷 (INT) 兩種輸入。可罩蓋插斷有三種不同作業型態，其中型態 2，可容許 128 個向量式插斷。

直接記憶體存取則由一稱為 DMA 控制器之“處理器”達成，微處理器休手旁觀，毫不介入。此種技巧主要用於“極高速”輸入 / 輸出設備之輸入 / 輸出作業。

實用上，取樣法目前幾乎已無人使用了。於微電腦領域，最常用的是插斷法。某些較優越之系統則亦具有 DMA 之能力。

於了解各種輸入 / 輸出技巧後，下一章我們將介紹一般最常連接至 Z 80 微處理器之輸入 / 輸出界面電路。

文件名稱：	Z80 微電腦軟體硬體第 1 2、1 3、1 4 章(掃描版)	文件分類	I
		文件編號	00028
		文件批號	06

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	2007-02-11	其他加註
	再版		

文件出處	原圖書名	Z80 微電腦軟體硬體
	原圖書作者	陳金追
	原圖書出版者	儒林圖書有限公司
	原圖書出版日期	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、圖書公司或原著作者所有，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

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