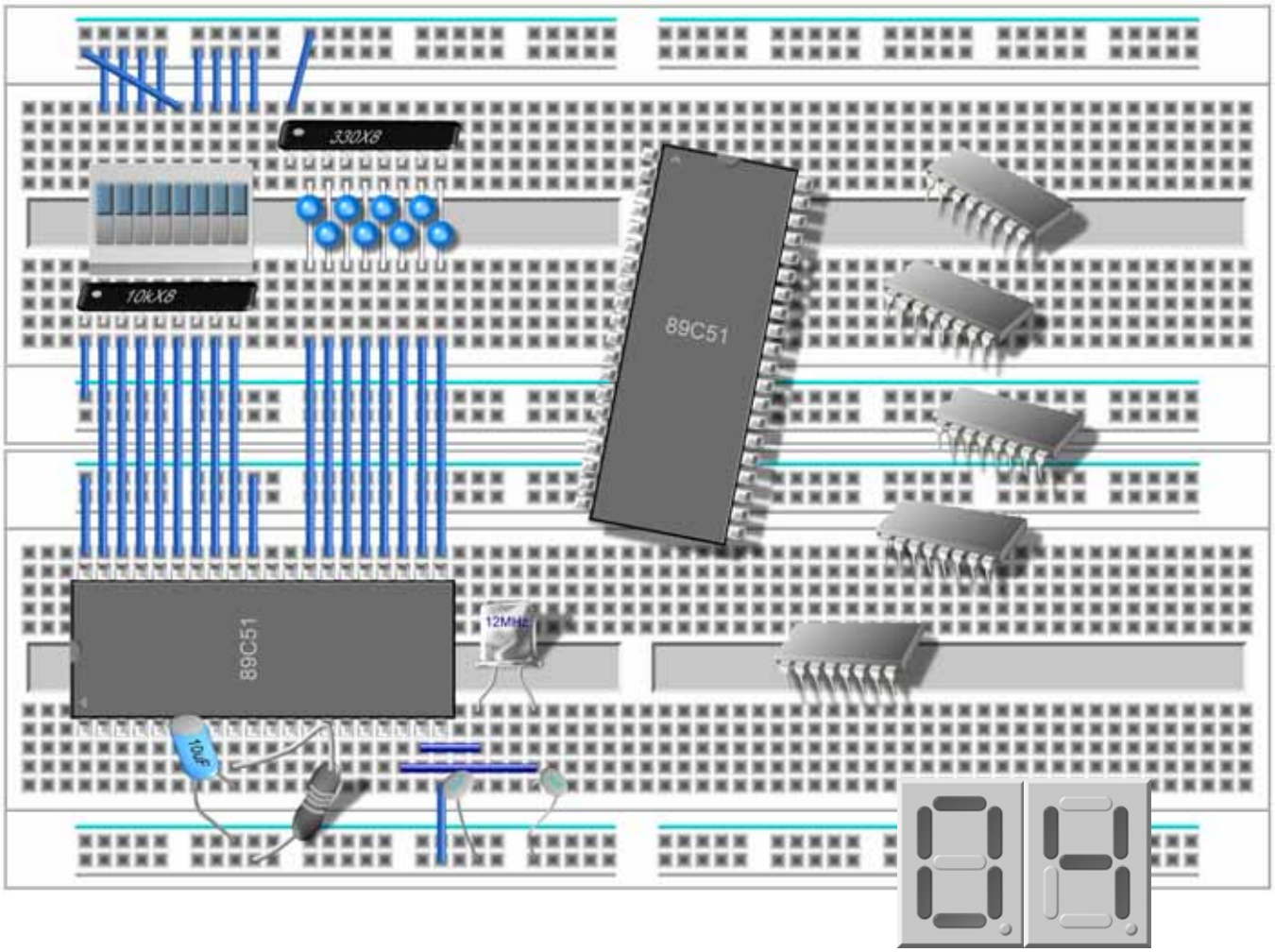


輸出入埠進階應用

本章內容豐富，主要包括三部分：

- 硬體部分：

  介紹了 8051 的省電模式。

  介紹 74138、7447、4x4 鍵盤、七節顯示器模組等。

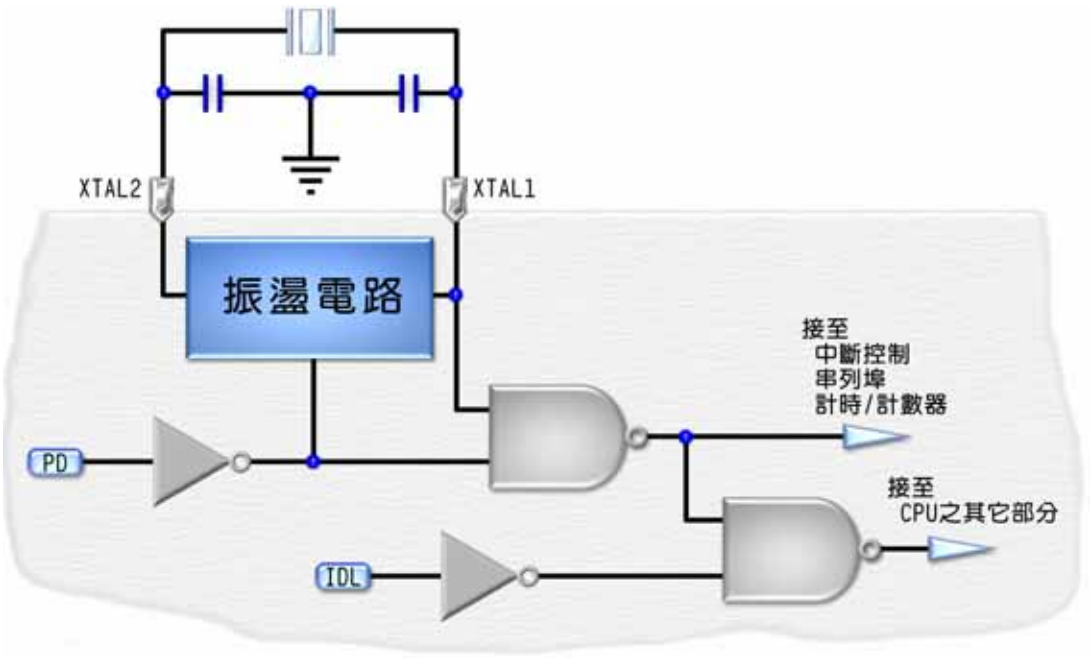
- 指令部分：

  詳細說明算術運算指令。

- 程式與實作部分：

  鍵盤掃瞄程式、七節顯示器掃瞄程式、編碼與查表法等。

4-1 8051 之省電模式

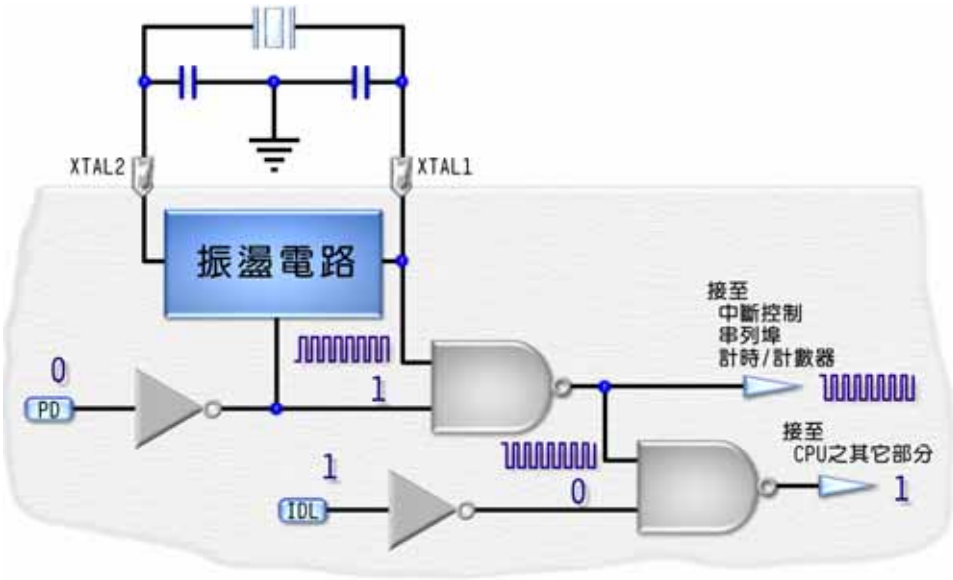


（圖1） 8051 功率控制示意圖

顧名思義，「**省電模式**」就是要讓系統的耗電量降低，同時又能保有系統中的資料，如此才能在使用電池的狀態下，長時間運作。8051 的 CHMOS 版本提供兩種省電模式，即閒置模式(**idle** mode，簡稱 **IDL** 模式)與功率下降模式(**p**ower **d**own mode，簡稱 **PD** 模式)。如圖 1 所示爲 8051 內部的功率控制示意圖，其中的 IDL 端點與 PD 端點連接到 PCON 暫存器的 IDL 及 PD 位元，而此部分控制了整個 CPU 所需的時鐘脈波，如下說明：

閒置模式

若 IDL 端點爲 1，則進入閒置模式，除了中斷、串列埠、計時/計數器等，仍正常提供時鐘脈波外，CPU 的其它部分均無時鐘脈波，因此，CPU 將停擺，而其中各暫存器、堆疊、記憶體、輸出入埠等的資料，並不會消失。如下圖所示爲閒置模式的狀態：



（圖2）　**閒置模式**

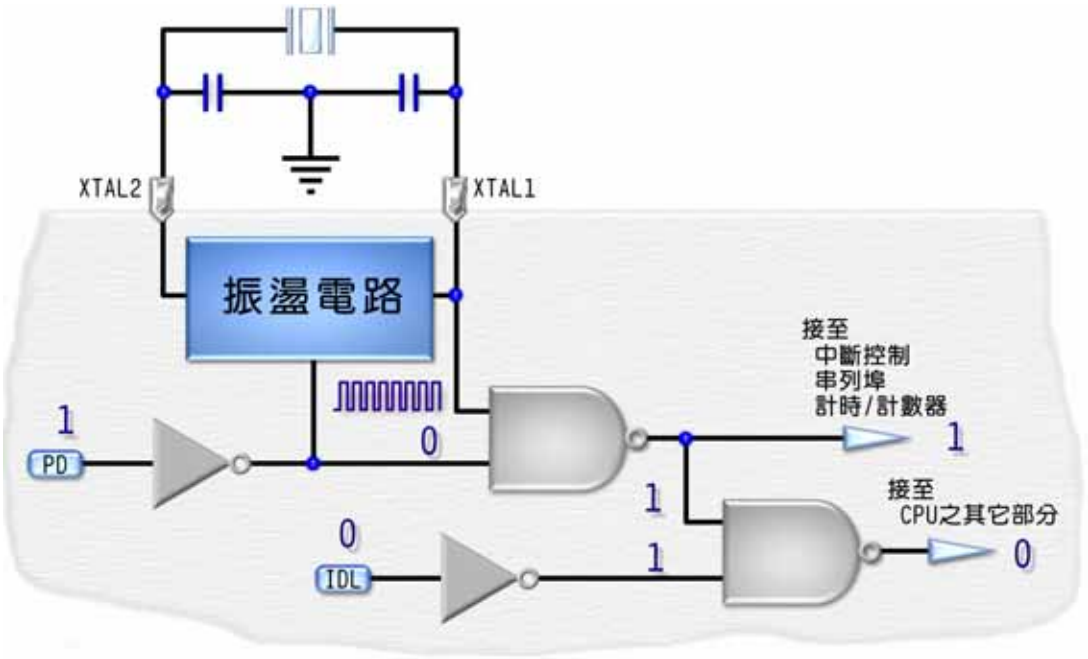
若要結束閒置模式，只要讓 IDL 端點爲 0，即可正常提供各部門時中脈波，CPU 將恢復正常運作。若要讓 IDL 端點爲 0，可以下列任一種方法達成：

1. 啓動任一個中斷致能，關於中斷，下一章再行說明。

2. 讓系統重置，也就是讓 RESET 接腳(第 9 腳)爲高準位，持續 2 個機械週期(2 微秒)，則 CPU 內各暫存器恢復爲初始狀態，PCON 暫存器裡的 IDL 位元將恢復爲 0，也就是說 IDL 端點爲 0。不過，系統重置後，各暫存器、堆疊、記憶體、輸出入埠等的資料將消失，所以，非不得已不會這樣做。

功率下降模式

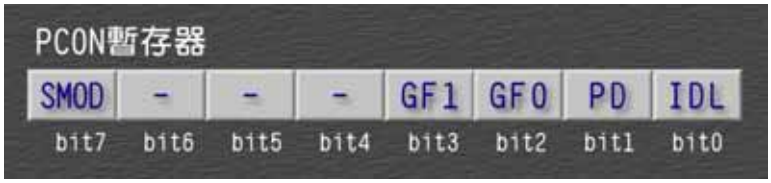
若 PD 端點爲 1，則進入功率下降模式，此時，完全不提供時鐘脈波，功率損耗降至最低。外加電源也可由原本的+5V，降至+2V。當然，各暫存器、堆疊、記憶體、輸出入埠等的資料，並不會消失。如下圖所示爲功率下降模式的狀態：



（圖3） **功率下降模式**

若要結束功率下降模式，必須先將電源恢復+5V，然後讓系統重置，即讓 RESET 接腳(第 9 腳)爲高準位，且需持續 10 毫秒。

由上述可得知，PCON 暫存器是主宰電源管理的暫存器，其位址爲 87H，是不可位元定址的暫存器，如下圖所示：



（圖4） **PCON 暫存器**

其中各位元如下說明：

- SMOD 位元爲鮑率(baud rate)倍增位元。當串列埠工作於模式 1、模式 2、模式 3，且使用計時器 1 爲其鮑率產生器時，若 SMOD 位元設定爲 1，則鮑率加倍；若 SMOD 位元設定爲 0，則鮑率正常。

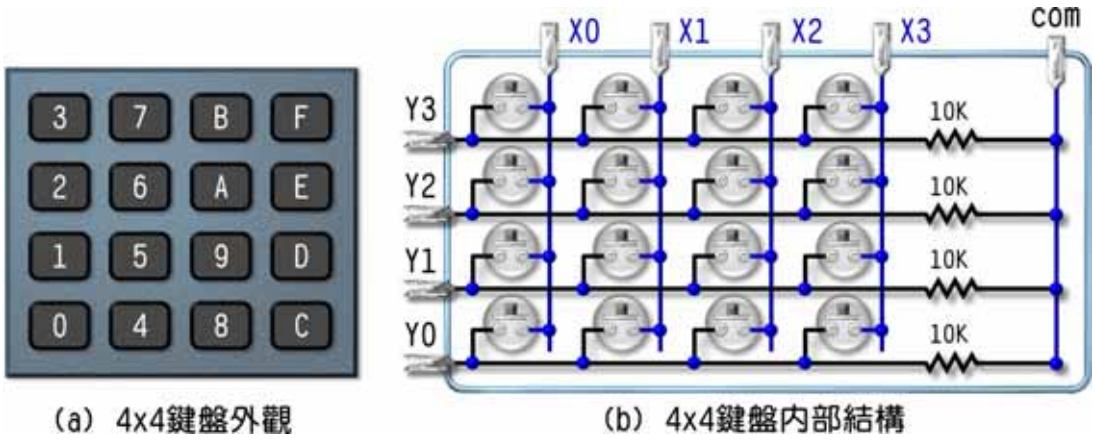
- GF1 與 GF0 位元爲一般用途旗標位元，使用者可自行設定或清

除這兩個旗標。通常，我們是使用這兩個旗標做爲由中斷喚醒閒置模式中的 8051 系統。

- PD 位元爲功率下降模式位元。當 PD=1，即可進入功率下降模式；PD=0，即可結束功率下降模式。
- IDL 位元爲閒置模式位元。當 IDL=1，即可進入閒置模式；IDL=0，即可結束閒置模式。

當系統重置時，PCON 暫存器的初始狀態恢復爲 0XXX0000B (CHMOS 版本的 8051)，若是 HOMS 版本的 8051，則爲 0XXXXXXXB。

4-2 鍵盤掃瞄



(a) 4x4鍵盤外觀　(b) 4x4鍵盤內部結構

(圖5) 4x4 鍵盤

4-2-1 鍵盤掃瞄原理

如圖 5 之(a)所示爲 4x4 鍵盤，而(b)爲其內部結構，其中包含 4 行(column)、4 列(row)，構成一個 4x4 的陣列，在此將每行連接端點定名爲 X0、X1、X2 及 X3，而每列連接端點定名爲 Y0、Y1、Y2 及 Y3。另外，每列各連接一個電阻器到共同接點(com)上。依掃瞄方式的不同，com 可能連接到 VCC 或 GND，**當我們要進行鍵盤掃瞄時，則將掃瞄信號送至 X0 到 X3，再由 Y0 至 Y3 讀取鍵盤狀態，即可判斷哪個按鍵被按下。**鍵盤掃瞄的方式有兩種，即低態掃瞄與高態掃瞄，如下說明：

低態掃瞄

低態掃瞄是將共同點 com 連接 VCC，沒有按何按鍵被按下時，Y3、Y2、Y1、Y0 端點能保持爲高態(即 1)。送入 X3、X2、X1、X0 的掃瞄信號之中，只有一個爲低態(即 0)，其餘三個爲高態。整個工作可分爲四個階段，如下說明：

1. 在第一個階段裡，主要目的是判斷 key3、key2、key1 及 key0 有沒有被按下。首先將 1110B 信號送入 X3、X2、X1、X0，也就是只有 X0 爲低態，其它各行皆爲高態。緊接著讀取 Y3、Y2、Y1、Y0 的狀況，

   - 若 Y3、Y2、Y1、Y0 爲 1110，代表 key0 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 1101，代表 key1 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 1011，代表 key2 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 0111，代表 key3 被按下。

   若 Y3、Y2、Y1、Y0 爲 1111，代表 key0、key1、key2 及 key3 都沒被按下，進入下一個階段。

2. 在第二個階段裡，主要目的是判斷 key7、key6、key5 及 key4 有沒有被按下。首先將 1101B 信號送入 X3、X2、X1、X0，也就是只有 X1 爲低態，其它各行皆爲高態。緊接著讀取 Y3、Y2、Y1、Y0 的狀況，

   - 若 Y3、Y2、Y1、Y0 爲 1110，代表 key4 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 1101，代表 key5 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 1011，代表 key6 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 0111，代表 key7 被按下。

   若 Y3、Y2、Y1、Y0 爲 1111，代表 key4、key5、key6 及 key7 都沒被按下，進入下一個階段。

3. 在第三個階段裡，主要目的是判斷 keyB、keyA、key9 及 key8 有沒有被按下。首先將 1011B 信號送入 X3、X2、X1、X0，也就是只有 X2 爲低態，其它各行皆爲高態。緊接著讀取 Y3、Y2、Y1、Y0 的狀況，

- 若 Y3、Y2、Y1、Y0 爲 1110，代表 key8 被按下。
- 若 Y3、Y2、Y1、Y0 爲 1101，代表 key9 被按下。
- 若 Y3、Y2、Y1、Y0 爲 1011，代表 keyA 被按下。
- 若 Y3、Y2、Y1、Y0 爲 0111，代表 keyB 被按下。

若 Y3、Y2、Y1、Y0 爲 1111，代表 key8、key9、keyA 及 keyB 都沒被按下，進入下一個階段。

4. 在第四個階段裡，主要目的是判斷 keyF、keyE、keyD 及 keyC 有沒有被按下。首先將 0111B 信號送入 X3、X2、X1、X0，也就是只有 X3 爲低態，其它各行皆爲高態。緊接著讀取 Y3、Y2、Y1、Y0 的狀況，

- 若 Y3、Y2、Y1、Y0 爲 1110，代表 keyC 被按下。
- 若 Y3、Y2、Y1、Y0 爲 1101，代表 keyD 被按下。
- 若 Y3、Y2、Y1、Y0 爲 1011，代表 keyE 被按下。
- 若 Y3、Y2、Y1、Y0 爲 0111，代表 keyF 被按下。

若 Y3、Y2、Y1、Y0 爲 1111，代表 keyC、keyD、keyE 及 keyF 都沒被按下，進入第一個階段，從頭開始，繼續掃瞄。

高態掃瞄

高態掃瞄是將共同點 com 接地(GND)，沒有按何按鍵被按下時，Y3、Y2、Y1、Y0 端點能保持爲低態(即 0)。送入 X3、X2、X1、X0 的掃瞄信號之中，只有一個爲高態(即 1)，其餘三個爲低態。整個工作可分爲四個階段，如下說明：

1. 在第一個階段裡，主要目的是判斷 key3、key2、key1 及 key0 有沒有被按下。首先將 0001B 信號送入 X3、X2、X1、X0，也就是只有 X0 爲高態，其它各行皆爲低態。緊接著讀取 Y3、Y2、Y1、Y0 的狀況，

- 若 Y3、Y2、Y1、Y0 爲 0001，代表 key0 被按下。
- 若 Y3、Y2、Y1、Y0 爲 0010，代表 key1 被按下。
- 若 Y3、Y2、Y1、Y0 爲 0100，代表 key2 被按下。
- 若 Y3、Y2、Y1、Y0 爲 1000，代表 key3 被按下。

若 Y3、Y2、Y1、Y0 爲 0000，代表 key0、key1、key2 及 key3 都沒被按下，進入下一個階段。

2. 在第二個階段裡，主要目的是判斷 key7、key6、key5 及 key4 有沒有被按下。首先將 0010B 信號送入 X3、X2、X1、X0，也就是只有 X1 爲高態，其它各行皆爲低態。緊接著讀取 Y3、Y2、Y1、Y0 的狀況，

   - 若 Y3、Y2、Y1、Y0 爲 0001，代表 key4 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 0010，代表 key5 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 0100，代表 key6 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 1000，代表 key7 被按下。

   若 Y3、Y2、Y1、Y0 爲 0000，代表 key4、key5、key6 及 key7 都沒被按下，進入下一個階段。

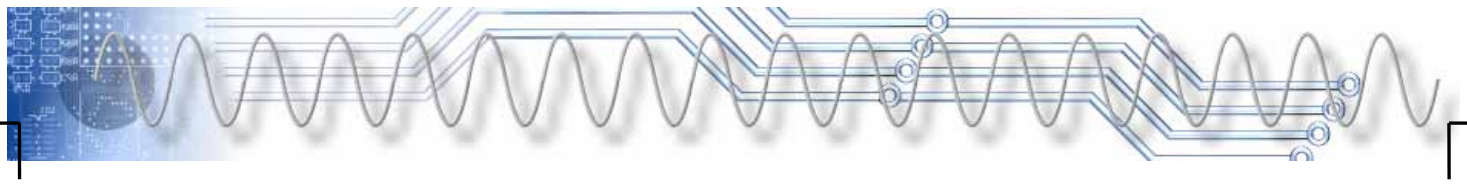
3. 在第三個階段裡，主要目的是判斷 keyB、keyA、key9 及 key8 有沒有被按下。首先將 0100B 信號送入 X3、X2、X1、X0，也就是只有 X2 爲高態，其它各行皆爲低態。緊接著讀取 Y3、Y2、Y1、Y0 的狀況，

   - 若 Y3、Y2、Y1、Y0 爲 0001，代表 key8 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 0010，代表 key9 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 0100，代表 keyA 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 1000，代表 keyB 被按下。

   若 Y3、Y2、Y1、Y0 爲 0000，代表 key8、key9、keyA 及 keyB 都沒被按下，進入下一個階段。

4. 在第四個階段裡，主要目的是判斷 keyF、keyE、keyD 及 keyC 有沒有被按下。首先將 1000B 信號送入 X3、X2、X1、X0，也就是只有 X3 爲高態，其它各行皆爲低態。緊接著讀取 Y3、Y2、Y1、Y0 的狀況，

   - 若 Y3、Y2、Y1、Y0 爲 0001，代表 keyC 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 0010，代表 keyD 被按下。
   - 若 Y3、Y2、Y1、Y0 爲 0100，代表 keyE 被按下。



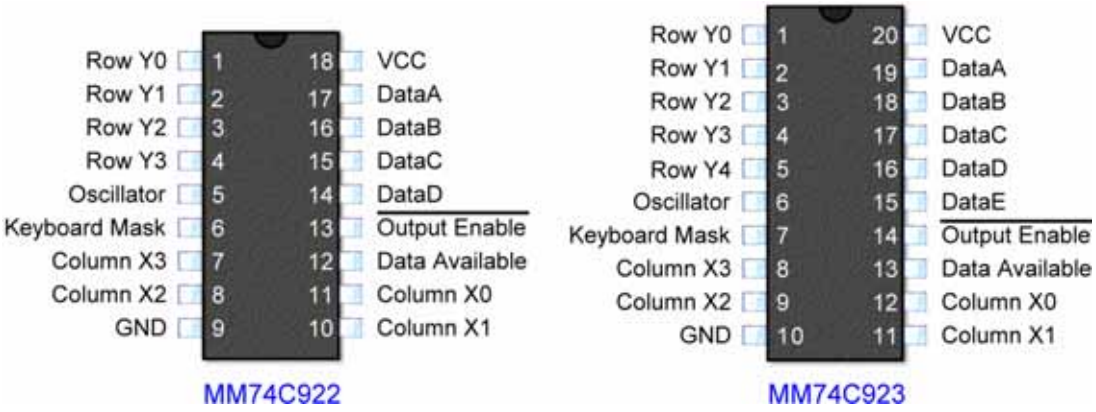
- 若 Y3、Y2、Y1、Y0 爲 1000，代表 keyF 被按下。

若 Y3、Y2、Y1、Y0 爲 0000，代表 keyC、keyD、keyE 及 keyF 都沒被按下，進入第一個階段，從頭開始，繼續掃瞄。

或許你會質疑，若在第一階段掃瞄時，有人按 key0、key1、key2、key3 以外的按鍵，是不是就偵測不到了？這你就不必擔心了，由於人類手指的動作很慢，按下按鍵到放開按鍵的時間，至少也得 0.1 秒(即 100ms)，而 CPU 的動作是以微秒計算的，從第一階段到第四階段跑一圈，只需幾毫秒(ms)而已，所以，手都還沒放開，程式就不知道掃過多少次了！另外，通常以低態掃瞄爲多，在本章中也將以低態掃瞄爲例，以探討其動作與程式分析。

4-2-2 認識 MM74C922/MM74C923

鍵盤狀態的偵測，除了可以利用鍵盤掃瞄軟體外，還可應用現成的鍵盤掃瞄 IC，例如 NS 半導體公司所提供的 MM74C922 及 MM74C923 爲鍵盤掃瞄 IC，其中 MM74C922 爲 4x4 的鍵盤掃瞄 IC、MM74C923 爲 4x5 的鍵盤掃瞄 IC，如下圖所示：



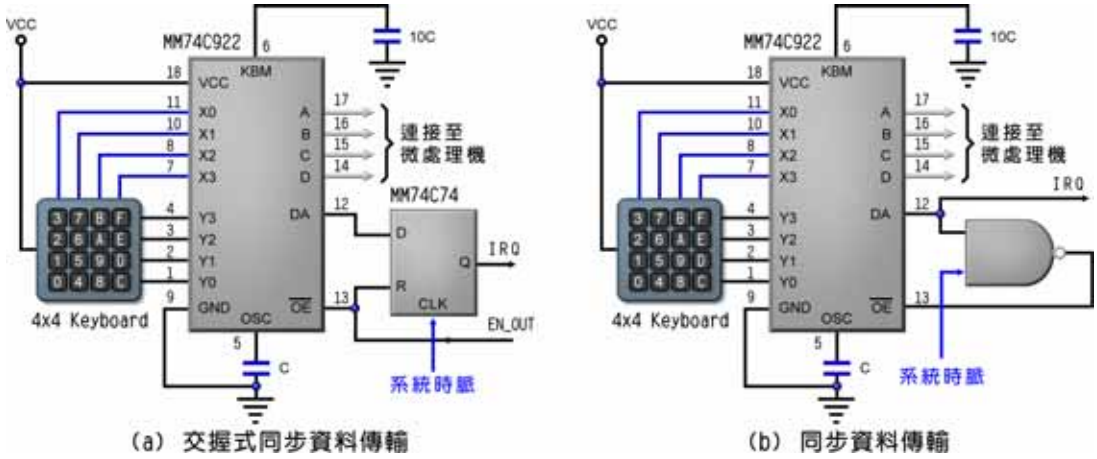
（圖6） **鍵盤掃瞄 IC**

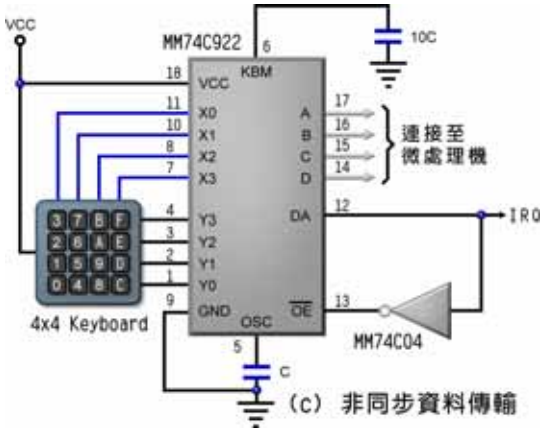
其中各接腳如下說明：

- **DataA～DataE**：輸出偵測鍵盤的結果，連接到微處理機的輸入埠。若是 MM74C922，則只有 DataA～DataD 四隻接腳。
- **ColumnX0～ColumnX3**：連接鍵盤的 X0～X3 行。
- **RowY0～RowY4**：連接鍵盤的 Y0～Y4 列，若是 MM74C922，

則只有 RowY0～RowY3 四隻接腳。

- **Oscillator**：本接腳爲振盪，簡稱 OSC 接腳，而本接腳必須連接電容器，讓 MM74C922(MM74C923)內部產生脈波，通常連接 0.1μF 即可。
- **Keyboard Mask**：本接腳爲按鍵遮沒接腳，簡稱爲 KBM 接腳，其功能是提供遮蓋按鍵彈跳的週期，也就是一種硬體防彈跳。本接腳需連接一個電容器 C，而其內部有一個 10K 電阻器 R，其所形成一個充放電週期(0.7RC)，即爲遮沒週期，而電容值可採用 OSC 接腳所連接電容器的 10 倍，約爲 1μF 即可。當按鍵按下時，即進入遮沒週期，首先將暫停 IC 內部的計數，同時 DA 接腳變爲高態，一直到按鍵被放開，DA 接腳才恢復爲低態。
- **Data Available**：本接腳爲容許資料輸出接腳，簡稱爲 DA 接腳。平時本接腳爲低態，此 IC 的資料輸出接腳 DataA～DataE 不提供正確的鍵盤狀態資料。當本接腳爲高態時，則鍵盤狀態資料將由資料輸出接腳 DataA～DataE 輸出。
- **Output Enable**：本接腳爲輸出致能接腳，簡稱爲 OE 接腳，屬於低態動作接腳。我們可透過簡單的邏輯閘或正反器，以規劃本 IC 的資料輸出接腳 DataA～DataE 與微處理機之間的傳輸方式。而資料傳輸方式可爲同步交握式資料傳輸、同步資料傳輸，及非同步資料傳輸等，如下圖所示：





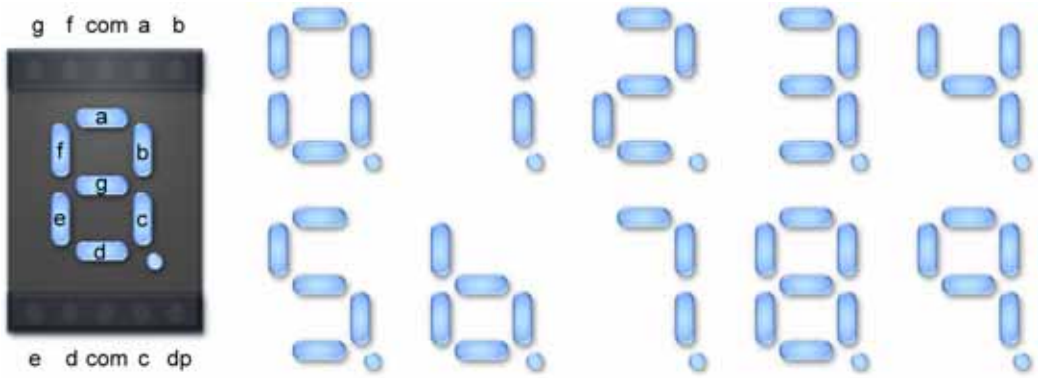
（圖7） **資料傳輸模式**

MM74C922/MM74C923 對各按鍵的反應如下表：

按鍵	連接接腳	DataE	DataD	DataC	DataB	DataA
0	X0/Y0	0	0	0	0	0
1	X1/Y0	0	0	0	0	1
2	X2/Y0	0	0	0	1	0
3	X3/Y0	0	0	0	1	1
4	X0/Y1	0	0	1	0	0
5	X1/Y1	0	0	1	0	1
6	X2/Y1	0	0	1	1	0
7	X3/Y1	0	0	1	1	1
8	X0/Y2	0	1	0	0	0
9	X1/Y2	0	1	0	0	1
10	X2/Y2	0	1	0	1	0
11	X3/Y2	0	1	0	1	1
12	X0/Y3	0	1	1	0	0
13	X1/Y3	0	1	1	0	1
14	X2/Y3	0	1	1	1	0
15	X3/Y3	0	1	1	1	1
16	X0/Y4	1	0	0	0	0
17	X1/Y4	1	0	0	0	1
18	X2/Y4	1	0	0	1	0
19	X3/Y4	1	0	0	1	1

4-3 七節顯示器掃瞄

七節顯示器是利用七個 LED 組合而成的顯示裝置，可以顯示 0 到 9 等十個數字，如下圖所示：

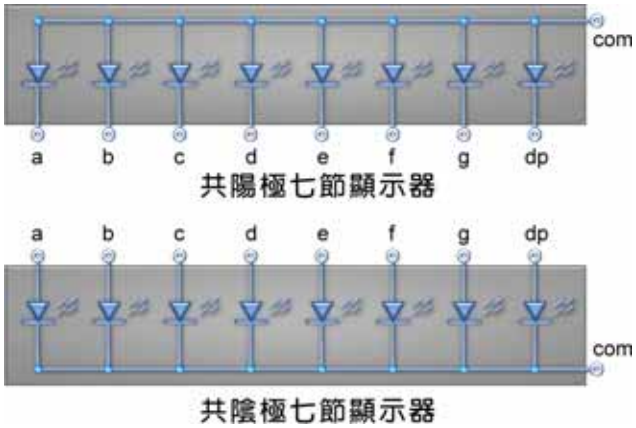


（圖8） **七節顯示器**

若要顯示多個七節顯示器，我們可分別驅動每個七節顯示器，不過，採分別驅動的方式，將耗用較多的零件與成本。若是利用人類的視覺暫態現象，則可以快速掃瞄的方式，只要一組驅動電路即可達到顯示多個七節顯示器的目的。

4-3-1 認識七節顯示器

基本上，七節顯示器可分爲共陽極與共陰極兩種，共陽極就是把所有 LED 的陽極連接到共同接點 com，而每個 LED 的陰極分別爲 a、b、c、d、e、f、g 及 dp(小數點)，如下圖所示：

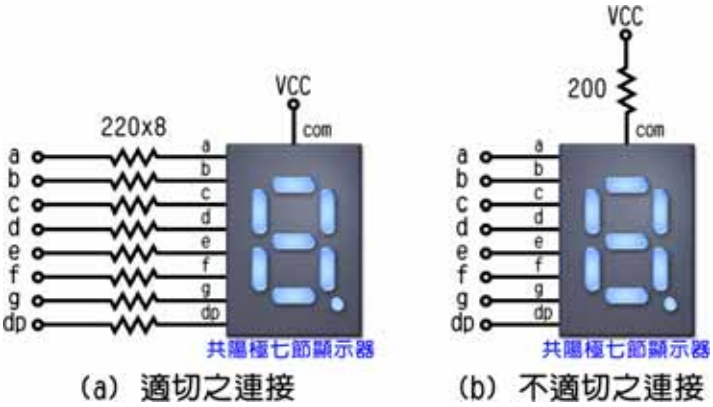


（圖9） **七節顯示器之結構**

同樣地，共陰極就是把所有 LED 的陰極連接到共同接點 com，而每個 LED 的陽極分別爲 a、b、c、d、e、f、g 及 dp(小數點)，如圖 9 所示。

共陽極七節顯示器

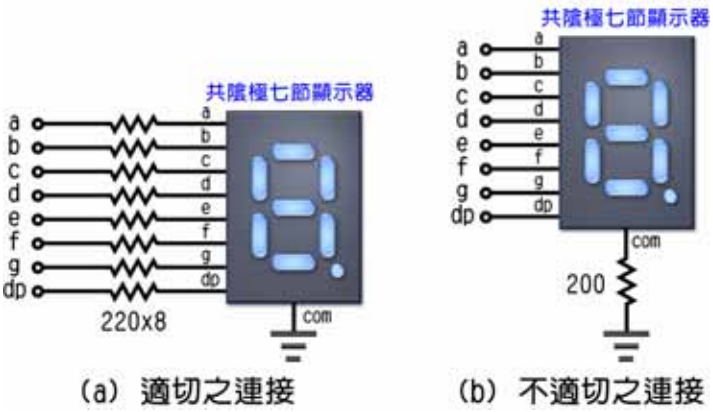
就像一般 LED 一樣，當我們要使用共陽極七節顯示器時，首先把 com 腳接+VCC，然後將每一隻陰極接腳各接一個限流電阻，如下圖(a)所示。在數位或微電腦電路裡，限流電阻可使用 200 到 330 歐姆，電阻值越大，亮度越弱、電阻值越小，電流越大。如下圖(b)所示，若只使用一個限流電阻，則顯示不同數字，將會有不同的亮度。



（圖10）**共陽極七節顯示器之應用**

共陰極七節顯示器

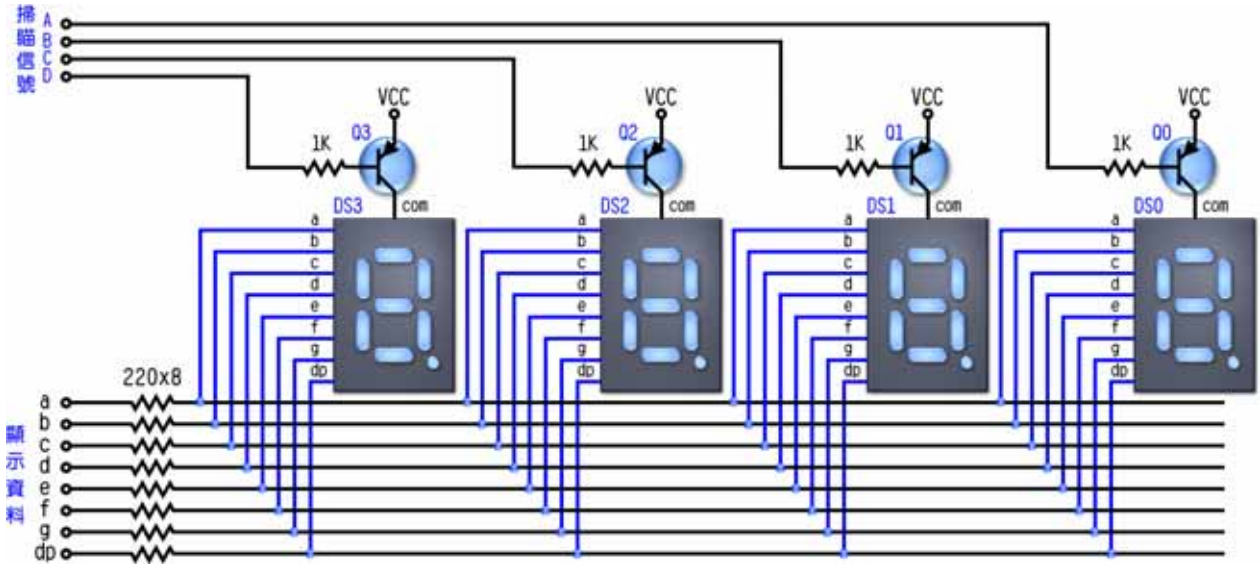
當我們要使用共陰極七節顯示器時，首先把 com 腳接地(GND)，然後將每一隻陽極接腳各接一個限流電阻，如下圖所示：



（圖11）**共陰極七節顯示器之應用**

多個七節顯示器與七節顯示模組

若要同時使用多個七節顯示器時，可採用掃瞄式顯示，也就是將每個七節顯示器的 a、b...g 都連接在一起，再使用電晶體分別驅動每個七節顯示器的共同接腳 com。以四個共陽極七節顯示器爲例，如下圖所示：

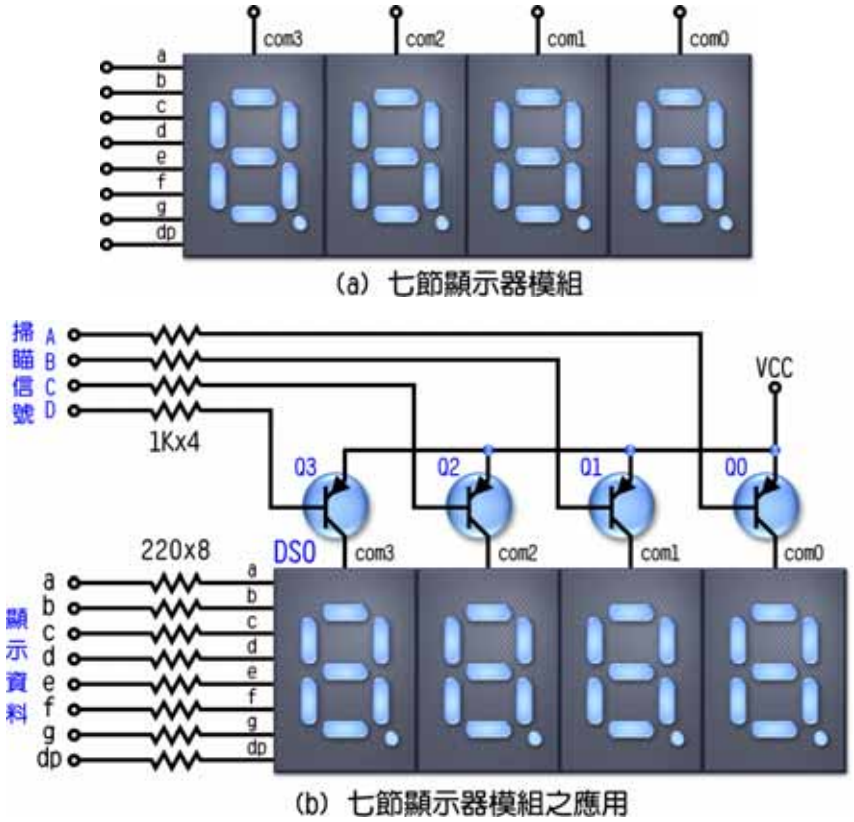


（圖12）**四個共陽極七節顯示器**

其顯示方式是將第一個七節顯示器所要顯示的資料丟到 a、b...g 匯流排上，然後將 1110 掃瞄信號送到四個電晶體的基極，即可顯示第一個七節顯示器；若要顯示第二個七節顯示器，同樣是將所要顯示的資料丟到 a、b...g 匯流排上，然後將 1101 掃瞄信號送到四個電晶體的基極，即可顯示第二個七節顯示器；若要顯示第三個七節顯示器，同樣是將所要顯示的資料丟到 a、b...g 匯流排上，然後將 1011 掃瞄信號送到四個電晶體的基極，即可顯示第三個七節顯示器；若要顯示第四個七節顯示器，同樣是將所要顯示的資料丟到 a、b...g 匯流排上，然後將 0111 掃瞄信號送到四個電晶體的基極，即可顯示第四個七節顯示器。掃瞄一圈後，再從頭開始掃瞄。雖然任何一個時間裡，只顯示一個七節顯示器，但只要從第一個到最後一個的掃瞄時間，不超過 16ms，則因人類的視覺暫態現象，而會同時看到這幾個數字。

如圖 12 所示，當我們使用四個七節顯示器時，每個七節顯示器都

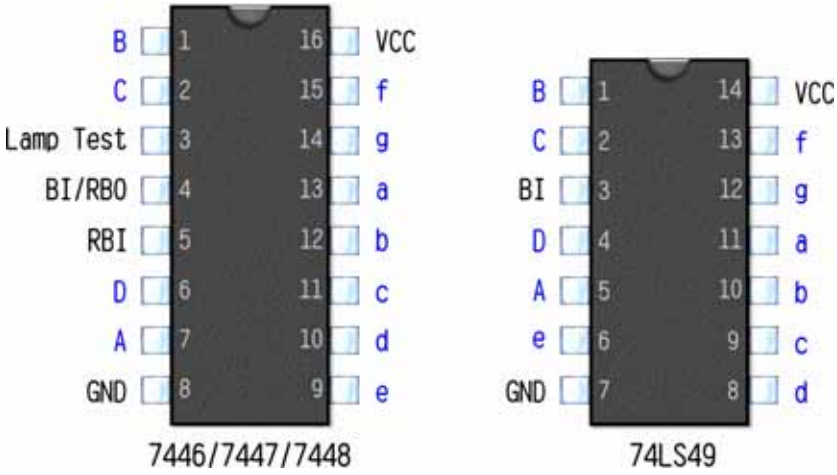
有 a、b...g，線路接很麻煩！這時候可改用四個七節顯示器包裝在一起的七節顯示器模組，如下圖所示：



（圖13）**七節顯示器模組**

4-3-2 認識 7447/7448

BCD 碼轉換成七節顯示碼的解碼驅動 IC，首推 7447 系列，包括 7446、7447、7448、74LS49，其中的 7446 及 7447 輸出低態動作的七節顯示碼，用以推動共陽極七節顯示碼；而 7448 及 74LS49 輸出高態動作的七節顯示碼，用以推動共陰極七節顯示碼。7446、7447 與 7448 的接腳相同(雙併排 16Pins)，74LS49 則爲雙併排 14Pins，如下圖所示：

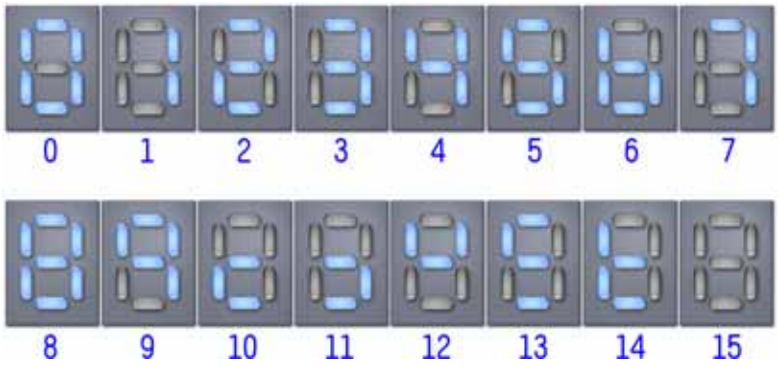


（圖14）7446、7447、7448、7449 之接腳

接腳說明

- **D、C、B、A**：BCD 碼輸入接腳。
- **a、b、c...g**：七節顯示碼輸出接腳。
- **Lamp Test**：本接腳爲測試接腳，簡稱爲 LT 接腳。當本接腳輸入低準位時，所連接的七節顯示器將全部亮。正常顯示下，本接腳應輸入高準位。
- **RBI**：本接腳爲漣波遮沒輸入接腳(ripple-blanking input)，正常顯示下，本接腳應輸入高準位。若本接腳輸入低準位(即 0)，且 D、C、B、A 接腳輸入爲 0，則該位數將不顯示，這項功能稱爲**消除前置 0**(leading zero suppression)或**消除尾端 0**(trailing zero suppression)，稍後再說明。
- **BI/RBO**：本接腳爲遮沒輸入或漣波遮沒輸出接腳(blanking in and/or ripple-blanking output)。正常顯示下，本接腳應輸入高準位或空接。若本接腳連接低準位(0)，則該位數將不顯示。當該位數不顯示時，本接腳將輸出低準位，以串接到前一個位數的 RBI 接腳，做爲消除前置 0(leading zero suppression)或消除尾端 0(trailing zero suppression)之用，稍後再說明。

真值表



數字或功能	輸入						BI/RBO	7446/7447 輸出							7448 輸出						
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g	a	b	c	d	e	f	g
0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	1	1	0
1	1	X	0	0	0	1	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0
2	1	X	0	0	1	0	1	0	0	1	0	0	1	0	1	1	0	1	1	0	1
3	1	X	0	0	1	1	1	0	0	0	0	1	1	0	1	1	1	1	0	0	1
4	1	X	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1
5	1	X	0	1	0	1	1	0	1	0	0	1	0	0	1	0	1	1	0	1	1
6	1	X	0	1	1	0	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1
7	1	X	0	1	1	1	1	0	0	0	1	1	1	1	1	1	1	0	0	0	0
8	1	X	1	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1
9	1	X	1	0	0	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	1
10	1	X	1	0	1	0	1	1	1	1	0	0	1	0	0	0	0	1	1	0	1
11	1	X	1	0	1	1	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1
12	1	X	1	1	0	0	1	1	0	1	1	1	0	0	0	1	0	0	0	1	1
13	1	X	1	1	0	1	1	0	1	1	0	1	0	0	1	0	0	1	0	1	1
14	1	X	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
15	1	X	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
BI	X	X	X	X	X	X	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
RBI	1	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
LT	0	X	X	X	X	X	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1

X：代表可爲 0 或 1

7446、7447、7448 真值表

數字或功能	輸 入					輸 出						
	D	C	B	A	BI	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	1	0	0	1
4	0	1	0	0	1	0	1	1	0	0	1	1
5	0	1	0	1	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	0	1	1
10	1	0	1	0	1	0	0	0	1	1	0	1
11	1	0	1	1	1	0	0	1	1	0	0	1
12	1	1	0	0	1	0	1	0	0	0	1	1
13	1	1	0	1	1	1	0	0	1	0	1	1
14	1	1	1	0	1	0	0	0	1	1	1	1
15	1	1	1	1	1	0	0	0	0	0	0	0
BI	X	X	X	X	0	0	0	0	0	0	0	0

X：代表可爲 0 或 1

74LS49 真值表

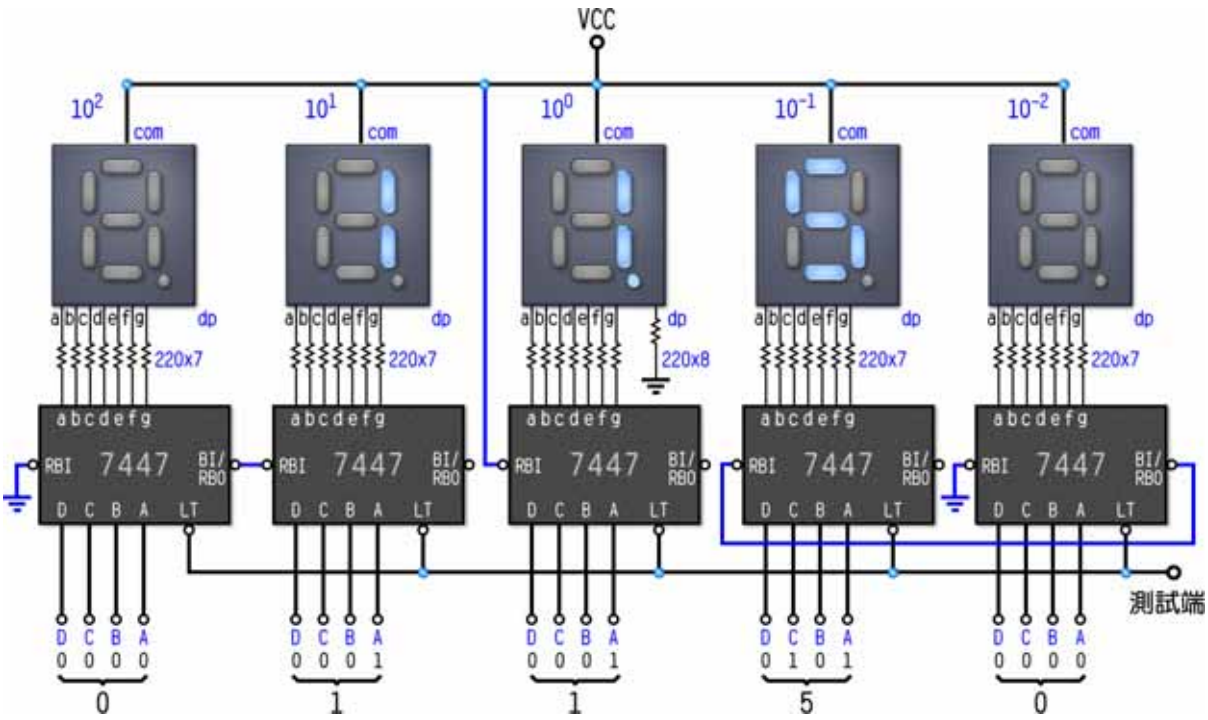
消除前置 0

所謂「消除前置 0」是指在數字整數部份的左邊位數若爲 0，則不顯示該位數，例如「012」則只顯示「12」、「002」則只顯示「2」。若要使用 7446、7447、7448 所提供的消除前置 0 功能，則可將整數部分最左邊位數的 RBI 接腳接地、BI/RBO 接腳連接到其右邊位數的 RBI 接腳...，以此類推。不過，個位數的 RBI 接腳並不與前左邊的 BI/RBO 接腳連接，以避免全部整數都不顯示的可能；同時，將其 dp 接腳連接限流電阻後接地，以顯示小數點。如圖 15 所示，其中整數部分輸入「0000 0001 0001」，則七節顯示器顯示「11.」

消除尾端 0

所謂「消除尾端 0」是指在數字小數部份的右邊位數若爲 0，則不顯示該位數，例如「0.150」則只顯示「0.15」、「0.200」則只顯示「0.2」。若要使用 7446、7447、7448 所提供的消除尾端 0 功能，則可將小數部分最右邊位數的 RBI 接腳接地、BI/RBO 接腳連接到

其左邊位數的 RBI 接腳...，以此類推。如圖 15 所示，其中整數部分輸入「0101 0000」，則七節顯示器顯示「.5」。



（圖15）**應用範例**

測試七節顯示器

若將 LT 接腳接地，則七節顯示器將全部亮，以檢查是否有任一節不顯示。對於多個七節顯示器的電路(如圖 15)，則可將每個 LT 接腳連接起來做爲測試端，平時將此測試端連接高準位，若要測試七節顯示器時，則將它接地即可。

這四個 IC 都是開集極式輸出，所以其輸出電流足以推動七節顯示器，其中 7448 的輸出端已內建 2K 歐姆的提升電阻，其它三個 IC 的輸出端都沒有內建提升電阻。除了低態輸出與高態輸出之不同外，對於開集極式輸出所連接的負載，其所連接的電源電壓也有些差異，如下表所示：

	最大負載電壓	最大吸入電流
7446	30V	40 ma
7447	15V	40 ma
7448	5.5V	6.4 ma
74LS49	5.5V	8 ma

4-3-3 編碼與查表法之應用

從前述 7447/7448 的真值表中，我們可得知推動七節顯示器所需要的信號，或者說是「編碼」，以共陽極爲例，7447 輸出「1100000」編碼，即可使其所連接的七節顯示器顯示「6」，即「」；如果我們能從 8051 的輸出埠輸出「1100000」，則不必使用 7447，直接連接到七節顯示器的「abcdefg」，也可以讓它顯示「」，甚至，改爲輸出「0100000」，即可顯示另一款的「6」，即「」，是不是很有個性呢？所以，不同的編碼會有不同的結果！利用軟體輸出編碼，以驅動七節顯示器，就可排除 7447 之類解碼 IC，一成不變的造型。

當我們要利用 8051 輸出自己編排的七節顯示碼，則需要一個編碼表，以及「查表法」程式，如下所示爲共陽極的七節顯示碼 0 到 9 編碼表：

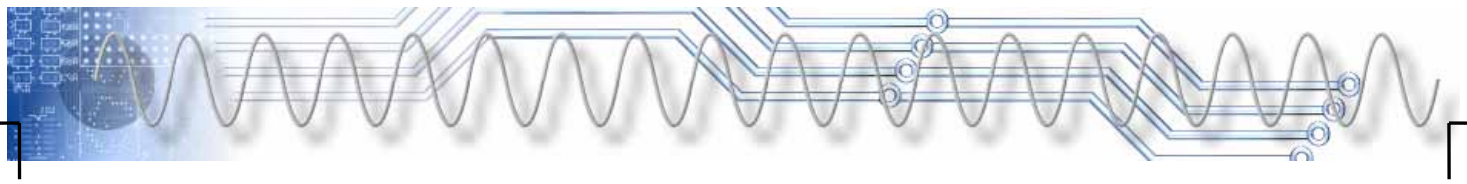
```
TABLE:      ;       abcdefgx
            DB      00000011B       ;共陽極之 0
            DB      10011111B       ;共陽極之 1
            DB      00100101B       ;共陽極之 2
            DB      00001101B       ;共陽極之 3
            DB      10011001B       ;共陽極之 4
            DB      01001001B       ;共陽極之 5
            DB      01000001B       ;共陽極之 6
            DB      00011111B       ;共陽極之 7
            DB      00000001B       ;共陽極之 8
            DB      00001001B       ;共陽極之 9
```

共陽極的七節顯示碼 0 到 9 編碼表

這個編碼表可放置在程式的後面，END 指令之前。如果我們要顯示的數字放置在 R3 暫存器，而連接共陽極七節顯示器的方式是

P0.7 => a
P0.6 => b
P0.5 => c
P0.4 => d
P0.3 => e
P0.2 => f
P0.1 => g

則查表法程式如下所示：

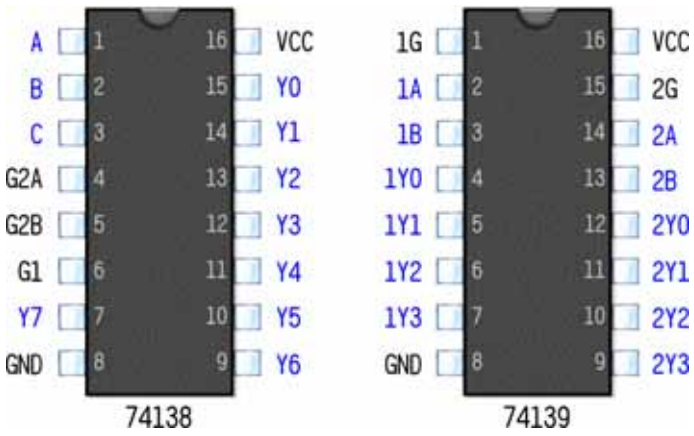


```
                ORG     0
START:          MOV     DPTR, #TABLE    ;將 DPTR 指向編碼表位置
                :
                :
                MOV     A, R3           ;將所要顯示的數字放入 ACC
                MOVC    A, @A+DPTR      ;將根據 R3 取出編碼表裡的編碼
                MOV     P0, A           ;輸出到七節顯示器
                :
                :
```

查表法

4-3-4 認識 74138/74139

不管是鍵盤掃瞄，還是多個七節顯示器的掃瞄，掃瞄信號是不可或缺的！當然，我們可以在微處理機裡，以軟體產生掃瞄信號，以 8051 而言，若第一個掃瞄信號爲「11111110」，則可利用「RL　A」左移指令產生下一個掃瞄信號。不過，如果直接由微處理機輸出掃瞄信號，必須使用較多的輸出埠。我們可以利用「2 對 4 解碼 IC」或「3 對 8 解碼 IC」，即可輕易產生 4 個位元或 8 個位元的掃瞄信號。74139 就是內含兩個「2 對 4」的解碼 IC，而 74139 則爲「3 對 8」的解碼 IC，如下圖所示：



（圖16）74139、74138 之接腳

74138 接腳說明

- **C、B、A**：BCD 碼輸入接腳。
- **Y7、Y6、Y5...Y0**：掃瞄碼輸出接腳。
- **G2A、G2B**：低態致能接腳，若 G2A 或 G2B 爲 1，則此解碼 IC 不工作，輸出接腳(Y7、Y6、Y5...Y0)將全部輸出爲 1。若 G2A

與 G2B 都爲 0，則此解碼 IC 才可能正常工作。

- **G1**：高態致能接腳，若本接腳爲 0，則此解碼 IC 不工作，輸出接腳(Y7、Y6、Y5...Y0)將全部輸出爲 1。若本接腳爲 1，則此解碼 IC 才可能正常工作。

74139 接腳說明

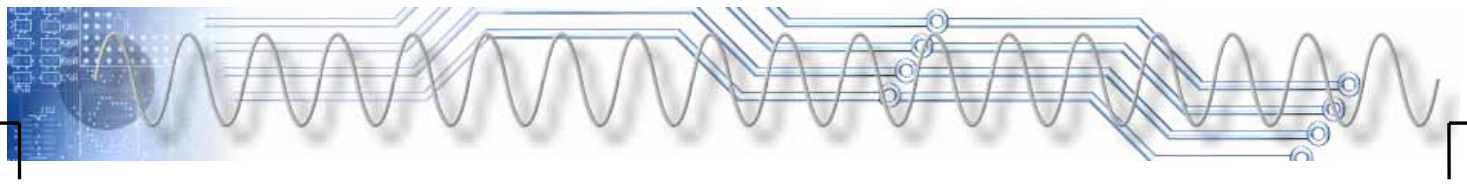
- **1B、1A**：第一個解碼器的 BCD 碼輸入接腳。
- **1Y3、1Y2、1Y1、1Y0**：第一個解碼器的掃瞄碼輸出接腳。
- **1G**：第一個解碼器的低態致能接腳，若 1G 爲 0，則第一個解碼器不工作，輸出接腳(1Y3、1Y2、1Y1、1Y0)將全部輸出爲 1。若 1G 爲 0，則第一個解碼器將正常工作。
- **2B、2A**：第二個解碼器的 BCD 碼輸入接腳。
- **2Y3、2Y2、2Y1、2Y0**：第二個解碼器的掃瞄碼輸出接腳。
- **2G**：第二個解碼器的低態致能接腳，若 2G 爲 0，則第二個解碼器不工作，輸出接腳(2Y3、2Y2、2Y1、2Y0)將全部輸出爲 1。若 2G 爲 0，則第二個解碼器將正常工作。

真值表

輸入						輸出							
致能			資料										
G1	G2A	G2B	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	1	1	X	X	X	1	1	1	1	1	1	1	1
X	0	1	X	X	X	1	1	1	1	1	1	1	1
X	1	0	X	X	X	1	1	1	1	1	1	1	1
0	X	X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

X：代表可爲 0 或 1

74138 真值表



輸　入			輸　出			
1G	1B	1A	1Y0	1Y1	1Y2	1Y3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

輸　入			輸　出			
2G	2B	2A	2Y0	2Y1	2Y2	2Y3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

X：代表可爲 0 或 1

74138 真值表

4-4 算術運算指令

算術運算指令的功能是將來源運算元的資料，與目的運算元裡的資料進行加、減、乘、除的算術運算，而其結果將放入目的運算元。資料轉移指令包括 24 個指令，在此將它們分爲 7 大類來介紹：

加法運算指令

加法運算指令的功能是將運算元的資料加到 ACC 裡，其中包括不考慮進位輸入 CY 與考慮進位輸入 CY 兩種，如下圖所示：

其中的 ADD 爲不考慮進位輸入，而 ADDC 是會將進位輸入 CY 一併加到 ACC。在此運算元可爲記憶體(RAM)位址 *direct* 的資料、暫存器 R*n* 的內容、以索引暫存器 R*i* 內容爲地址(@R*i*)的資料、立即值# *data* 等。

- ADD　A, *direct*
  - **說明**：將(*direct*)位址的內容加到 ACC 裡，即(*direct*)+ACC→ACC。
  - **組譯後大小**：2 bytes　　**執行時間**：12 個時鐘脈波
  - **範例**：

指令：ADD A, 20H
若執行前：ACC=12H、記憶體(20H)位址的內容爲 ABH
執行後：ACC=BDH、記憶體(20H)位址的內容爲 ABH

- ADD A, R*n*
  - **說明**：將暫存器 R*n* 的內容加到 ACC 裡，即 R*n* +ACC →ACC。
  - **組譯後大小**：1 byte **執行時間**：12 個時鐘脈波
  - **範例**：

指令：ADD A, R1
若執行前：ACC=12H、R1=27H
執行後：ACC=39H、R1=27H

- ADD A, @R*i*
  - **說明**：以索引暫存器 R*i* 的內容爲地址，將記憶體中該地址裡的資料加到 ACC 裡，即(R*i*)+ACC →ACC。
  - **組譯後大小**：1 byte **執行時間**：12 個時鐘脈波
  - **範例**：

指令：ADD A, @R0
若執行前：ACC=35H、R0=21H、記憶體(21H)位址的內容爲 A3H
執行後：ACC=D8H、R0=21H、記憶體(21H)位址的內容爲 A3H

- ADD A, #*data*
  - **說明**：將 *data* 立即值加到 ACC 裡，即 *data*+ACC→ACC。
  - **組譯後大小**：2 bytes **執行時間**：12 個時鐘脈波
  - **範例**：

指令：ADD A, #20H
若執行前：ACC=35H
執行後：ACC=55H

- ADDC A, *direct*
  - **說明**：將(*direct*)位址的內容及進位位元 CY 加到 ACC 裡，即

(*direct*)+CY+ACC→ACC。

組譯後大小：2 bytes　　**執行時間**：12 個時鐘脈波

範例：

指令：ADDC　A, 20H

若執行前：ACC=12H、CY=1、記憶體(20H)位址的內容爲 ABH

執行後：ACC=BEH、CY=0、記憶體(20H)位址的內容爲 ABH

- ADDC　A, R*n*

說明：將暫存器 R*n* 的內容及進位位元 CY 加到 ACC 裡，即 R*n* +CY+ACC →ACC。

組譯後大小：1 byte　　**執行時間**：12 個時鐘脈波

範例：

指令：ADDC　A, R1

若執行前：ACC=12H、CY=1、R1=27H

執行後：ACC=3AH、CY=0、R1=27H

- ADDC　A, @R*i*

說明：以索引暫存器 R*i* 的內容爲地址，將記憶體中該地址裡的資料及進位位元 CY 加到 ACC 裡，即(R*i*)+CY+ACC →ACC。

組譯後大小：1 byte　　**執行時間**：12 個時鐘脈波

範例：

指令：ADDC　A, @R0

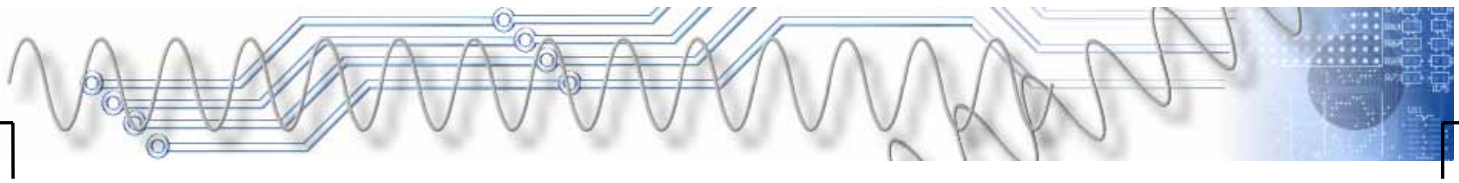
若執行前：ACC=35H、CY=1、R0=21H、記憶體(21H)位址的內容爲 A3H

執行後：ACC=D8H、CCY=0、R0=21H、記憶體(21H)位址的內容爲 A3H

- ADDC　A, #*data*

說明：將 *data* 立即值及進位位元 CY 加到 ACC 裡，即 *data*+CY+ACC→ACC。

組譯後大小：2 bytes　　**執行時間**：12 個時鐘脈波



▶**範例**：

指令：ADDC A, #20H
若執行前：ACC=F5H、CY=1
執行後：ACC=16H、CY=1

減法運算指令

減法運算指令的功能是將 ACC 的內容減去進位輸入 CY，再減去運算元的資料，如下圖所示：

SUBB A, 運算元
direct
R*n*
@R*i*
#*data*

其中的運算元可爲記憶體(RAM)位址 *direct* 的資料、暫存器 R*n* 的內容、以索引暫存器 R*i* 內容爲地址(@R*i*)的資料、立即值# *data* 等。

SUBB A, *direct*

▶**說明**：將 ACC 的內容減去(*direct*)位址的內容及進位位元 CY，即 ACC-(*direct*)-CY→ACC。

▶**組譯後大小**：2 bytes **執行時間**：12 個時鐘脈波

▶**範例**：

指令：SUBB A, 20H
若執行前：ACC=12H、CY=1、記憶體(20H)位址的內容爲 ABH
執行後：ACC=66H、CY=1、記憶體(20H)位址的內容爲 ABH

SUBB A, R*n*

▶**說明**：將 ACC 的內容減去暫存器 R*n* 的內容及進位位元 CY，即 ACC-R*n* -CY →ACC。

▶**組譯後大小**：1 byte **執行時間**：12 個時鐘脈波

▶**範例**：

指令：**SUBB　A, R1**

若執行前：**ACC=12H、CY=1、R1=27H**

執行後：**ACC=EAH、CY=1、R1=27H**

- SUBB　A, @R*i*
  - **說明**：將 ACC 的內容減去(R*i*)的內容及進位位元 CY，即 ACC-(R*i*)-CY →ACC。
  - **組譯後大小**：1 byte　　**執行時間**：12 個時鐘脈波
  - **範例**：

指令：**ADDC　A, @R0**

若執行前：**ACC=35H、CY=1、R0=21H、記憶體(21H)位址的內容爲 A3H**

執行後：**ACC=91H、CY=1、R0=21H、記憶體(21H)位址的內容爲 A3H**

- SUBB　A, #*data*
  - **說明**：將 ACC 的內容減去 *data* 立即值及進位位元 CY，即 ACC-*data*-CY →ACC
  - **組譯後大小**：2 bytes　　**執行時間**：12 個時鐘脈波
  - **範例**：

指令：**SUBB　A, #20H**

若執行前：**ACC=F5H、CY=1**

執行後：**ACC=D4H、CY=0**

加 1 指令

加 1 指令的功能是將運算元加 1，如下圖所示：

其中的運算元可爲 ACC、記憶體(RAM)位址 *direct* 的資料、暫存器

R*n* 的內容、以索引暫存器 R*i* 內容爲地址(@R*i*)的資料、資料指標暫存器 DPTR 的內容等。

- INC A
  - **說明**：將 ACC 的內容加 1，即 ACC+1→ACC
  - **組譯後大小**：1 byte **執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：INC  A
若執行前：ACC=12H
執行後：ACC=13H
```

- INC *direct*
  - **說明**：將(*direct*)位址的內容加 1，即(*direct*)+1→(*direct*)
  - **組譯後大小**：2 bytes **執行時間**：12 個時鐘脈波
  - **範例**：

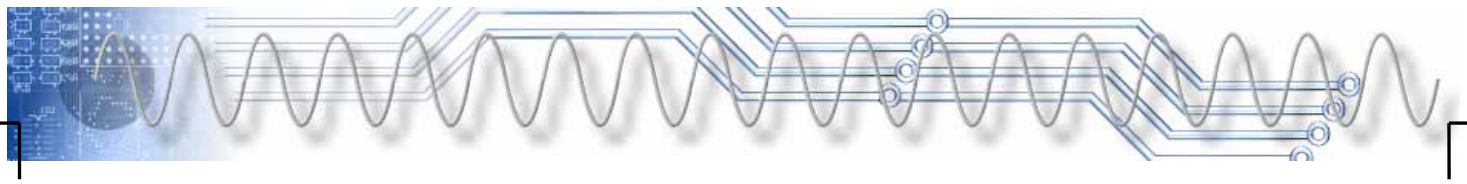
```
指令：INC  20H
若執行前：記憶體(20H)位址的內容爲 A3H
執行後：記憶體(20H)位址的內容爲 A4H
```

- INC Rn
  - **說明**：將 R*n* 的內容加 1，即 R*n* +1→ R*n*
  - **組譯後大小**：1 byte **執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：INC  R4
若執行前：R4=00H
執行後：R4=01H
```

- INC @R*i*
  - **說明**：將(R*i*)位址的內容加 1，即(R*i*) +1→(R*i*)
  - **組譯後大小**：1 byte **執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：INC  @R0
```



若執行前：**R0=30H**、記憶體**(30H)**位址的內容爲 **33H**
執行後：**R0=30H**、記憶體**(30H)**位址的內容爲 **34H**

- INC　DPTR
  - **說明**：將 DPTR 的內容加 1，即 DPTR+1→ DPTR
  - **組譯後大小**：1 byte　　**執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：INC　DPTR
若執行前：DPTR=0000H
執行後：DPTR=0001H
```

減 1 指令

減 1 指令的功能是將運算元加 1，如下圖所示：

其中的運算元可爲 ACC、記憶體(RAM)位址 *direct* 的資料、暫存器 R*n* 的內容、以索引暫存器 R*i* 內容爲地址(@R*i*)的資料等。

- DEC　A
  - **說明**：將 ACC 的內容減 1，即 ACC-1→ACC
  - **組譯後大小**：1 byte　　**執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：DEC　A
若執行前：ACC=12H
執行後：ACC=11H
```

- DEC　R*n*
  - **說明**：將 R*n* 的內容減 1，即 R*n* -1→ R*n*
  - **組譯後大小**：1 byte　　**執行時間**：12 個時鐘脈波

▶範例：

```
指令：DEC   R5
若執行前：R5=01H
執行後：R4=00H
```

- DEC *direct*
  - ▶說明：將(*direct*)位址的內容減 1，即(*direct*)-1→(*direct*)
  - ▶組譯後大小：2 bytes　　執行時間：12 個時鐘脈波
  - ▶範例：

```
指令：DEC   20H
若執行前：記憶體(20H)位址的內容爲 A3H
執行後：記憶體(20H)位址的內容爲 A2H
```

- DEC @R*i*
  - ▶說明：將(R*i*)位址的內容減 1，即(R*i*) +1→(R*i*)
  - ▶組譯後大小：1 byte　　執行時間：12 個時鐘脈波
  - ▶範例：

```
指令：DEC   @R0
若執行前：R0=30H、記憶體(30H)位址的內容爲 33H
執行後：R0=30H、記憶體(30H)位址的內容爲 32H
```

乘法運算指令

- MUL AB
  - ▶說明：ACC 的內容爲被乘數(8 位元)、B 暫存器的內容爲乘數(8 位元)，進行乘法運算，而其乘積(16 位元)之高 8 位元存放在 B 暫存器、低 8 位元存放在 ACC，即 A×B→BA。
  - ▶組譯後大小：1 byte　　執行時間：48 個時鐘脈波
  - ▶範例：

```
指令：MUL   AB
若執行前：ACC=20H、B=12H
執行後：ACC=40H、B=02H
```

除法運算指令

DIV　AB

說明：以 ACC 的內容爲被除數(8 位元)、B 暫存器的內容爲除數(8 位元)，進行除法運算，而其餘數(8 位元)存放在 B 暫存器、商數存放在 ACC，即 A÷B→A 餘 B。

組譯後大小：1 byte　　**執行時間**：48 個時鐘脈波

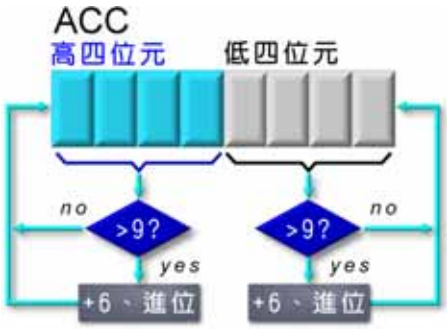
範例：

```
指令：DIV　AB
若執行前：ACC=20H、B=12H
執行後：ACC=01H、B=0EH
```

BCD 調整指令

DA　A

說明：將 ACC 的內容進行 BCD 調整，即



組譯後大小：1 byte　　**執行時間**：12 個時鐘脈波

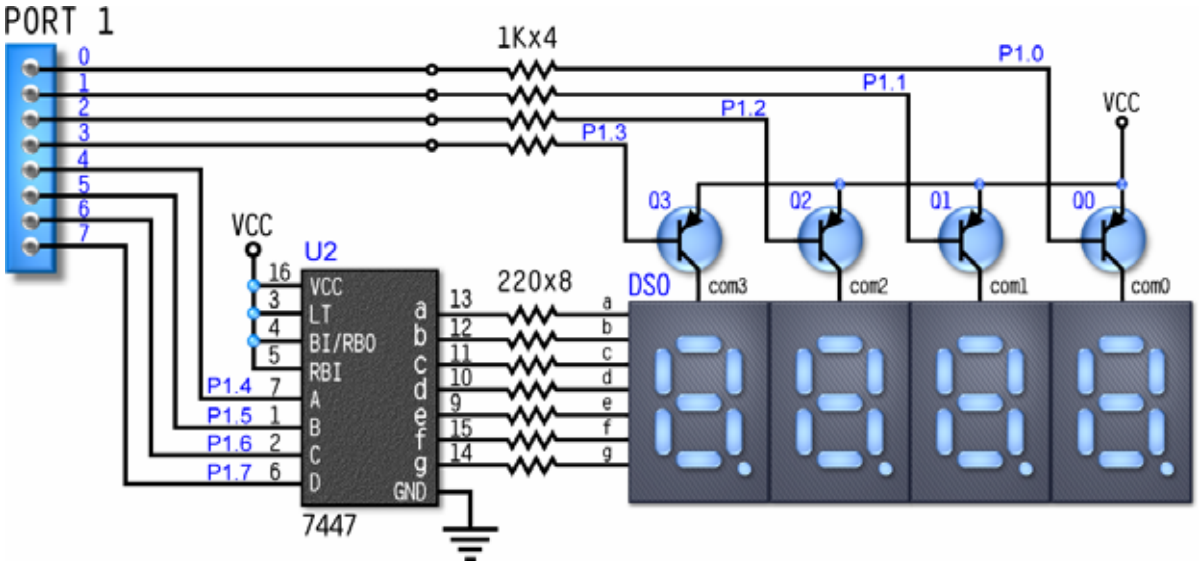
範例：

```
指令：DA　A
若執行前：ACC=ABH、CY=0
執行後：ACC=11H、CY=1
```

4-5 實例演練

在本單元裡提供五個範例，分別展示鍵盤掃瞄、七節顯示器掃瞄、查表法、解碼 IC，以及其混合應用，如下所示：

4-5-1 四位數七節顯示器實例演練



（圖17）**電路圖**

功能說明

如圖 17 所示，由 P1 高四位元將所要顯示的數字(BCD 碼)輸出到 7447，經 7447 解碼後，驅動 4 位數字的七節顯示器模組；而由 P1 低四位元將掃瞄碼分送到七節顯示器模組的四個共同端，使這個七節顯示器模組顯示「8051」。

參考程式

依功能需求與電路結構得知，四個位數不能同時顯示不同的數字，所以要採取掃瞄方式，也就是一個一個位數顯示。若要最右邊位數顯示「1」，則 P1 的高四位元輸出 1、P1 的低四位元輸出 1110B(即 P1 輸出 1EH)，只有 Q0 電晶體導通，所以「1」的七節碼將在最右邊的七節顯示器顯示。

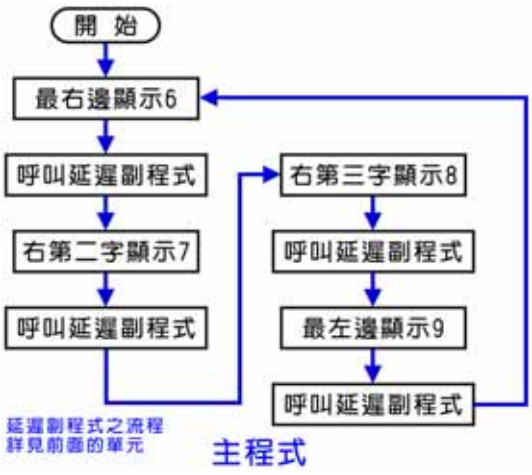
若要右邊第二個位數顯示「5」，則 P1 的高四位元輸出 5、P1 的低四位元輸出 1101B(即 P1 輸出 5DH)，只有 Q1 電晶體導通，所以「5」

的七節碼將在右邊第二個的七節顯示器顯示。

若要右邊第三個位數顯示「0」，則 P1 的高四位元輸出 0、P1 的低四位元輸出 1011B(即 P1 輸出 0BH)，只有 Q2 電晶體導通，所以「0」的七節碼將在右邊第三個的七節顯示器顯示。

若要最左邊位數顯示「8」，則 P1 的高四位元輸出 8、P1 的低四位元輸出 0111B(即 P1 輸出 87H)，只有 Q3 電晶體導通，所以「8」的七節碼將在最左邊的七節顯示器顯示。

爲了讓人看清楚顯示的數字，每點亮一個數字後，必須等一下，也就是延遲一段時間，才點亮下一個數字。若延遲的時間太短，則數字的亮度較低；若延遲的時間太長，則可看出數字是一個接一個的亮(閃爍的感覺)，不像同時亮四個數字。而從點亮第一個位數到點亮最後一個位數，若在 16ms 之內完成，則每秒鐘將可掃瞄 60 次，如此就不會有閃爍的感覺。換言之，每個數字的點亮時間約 4ms，最簡單的方式是在 P0 輸出一個數字之後，呼叫一個 4ms 的延遲副程式即可。



```
                ORG     0               ;程式從 0 位址開始
START:          MOV     P1, #1EH        ;將最右邊數字顯示 6
                CALL    DELAY           ;延遲 4ms
                MOV     P1, #5DH        ;將右邊第二個數字顯示 7
                CALL    DELAY           ;延遲 4ms
                MOV     P1, #0BH        ;將右邊第三個數字顯示 8
                CALL    DELAY           ;延遲 4ms
                MOV     P1, #87H        ;將最左邊數字顯示 9
                CALL    DELAY           ;延遲 4ms
                JMP     START           ;從頭開始掃瞄
;=============延遲約 4ms=(即 2usxR7xR7)====================
DELAY:          MOV     R7, #10
D1:             MOV     R6, #200
                DJNZ    R6, $
                DJNZ    R7, D1
                RET
                END
```

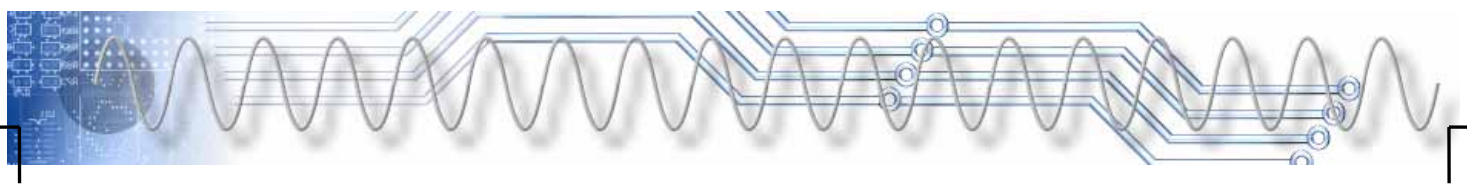
四位數七節顯示器實驗(ch4-1.asm)

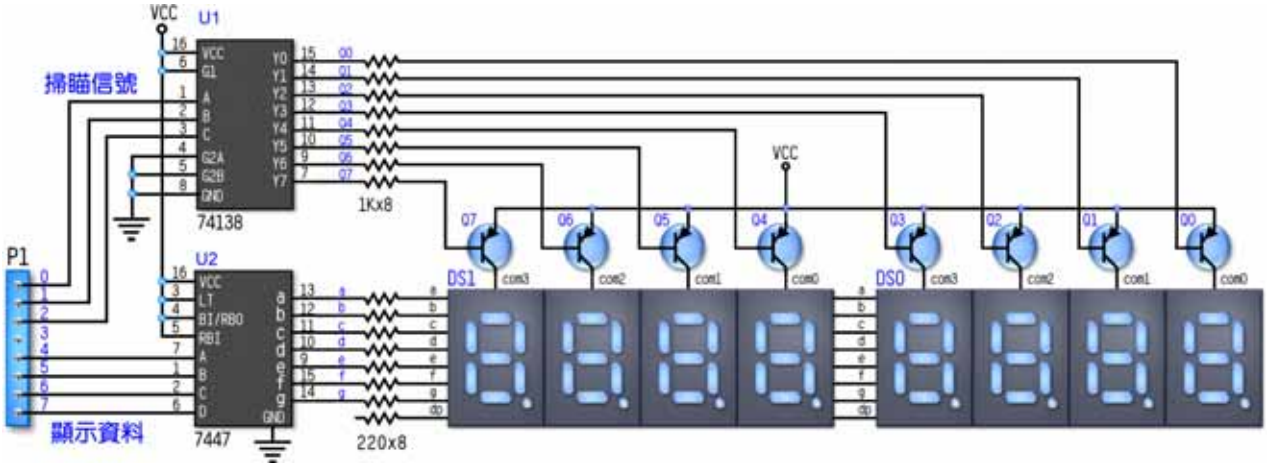
操作

1. 依功能需求與電路結構撰寫程式，然後將該程式組譯與連結，以產生*.HEX 檔。
2. 請按圖 17 連接線路，再使用實體模擬器，載入該程式(*.HEX)，以模擬該電路的動作。若有非預期的狀況，則檢視線路的連接狀況，看看哪裡出問題？並將它記錄在實驗報告裡。
3. 若實體模擬功能正常，請將程式燒錄到 89C51，再把該 89C51 放入實體電路，以取代剛才的實體模擬器，然後直接送電，看看是否正常？
4. 撰寫實驗報告。

思考一下

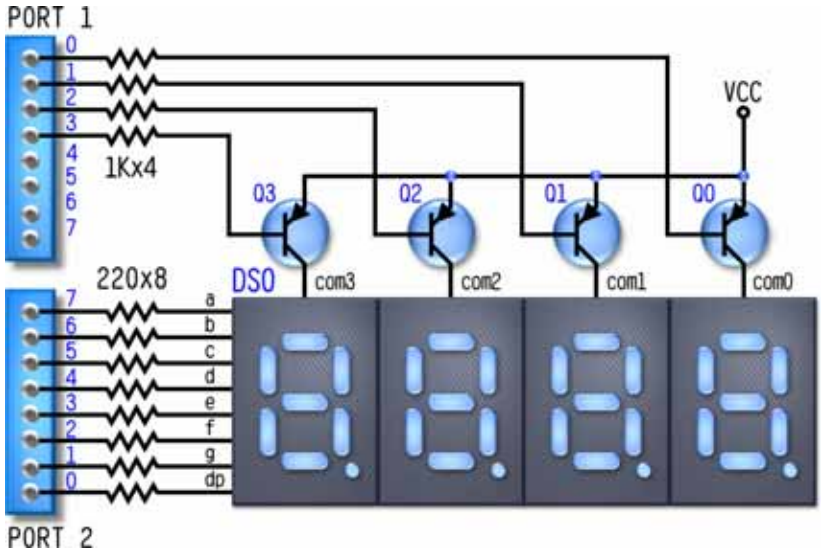
1. 請將程式中的延遲時間縮短爲原本的十分之一，結果會如何？同樣地，請將程式中的延遲時間縮短爲原本的十倍，結果會如何？
2. 使用兩個四位數的七節顯示器模組，連接成 8 個數字的顯示電路，再使用 74138 與 7447 做爲解碼之用，如圖 18 所示，若要顯示「12345678」程式應如何撰寫？





（圖18）**八位數顯示電路**

4-5-2 直接驅動七節顯示器實例演練



（圖19）**電路圖**

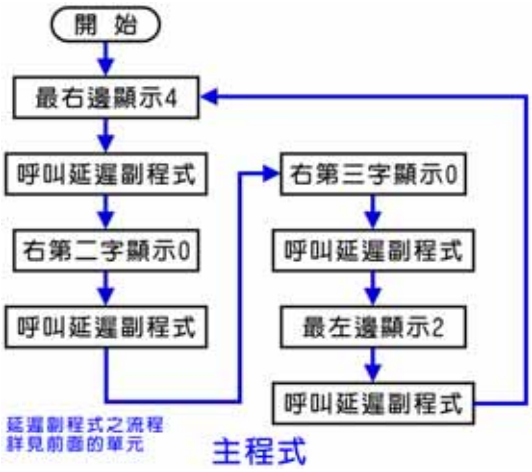
功能說明

如圖 19 所示，由 P2 將所要顯示的七節顯示碼直接輸出到 4 位數字的七節顯示器模組，再由 P1 的低四位元將掃瞄碼直接分送到七節顯示器模組的四個共同端，使這個七節顯示器模組顯示「2004」。

參考程式

依功能需求與電路結構得知，若要看到顯示四個位數就要採取掃瞄方式。其中掃瞄碼分別爲 1110、1101、1011 及 0111，所以 PORT 1

的低四位元依序輸出這四個掃瞄碼，其高四位元並不影響，設爲 1111 即可；而所要顯示的資料爲「2004」，根據電路圖 PORT 2 的連接方式，剛好符合 4-20 頁的編碼表，所以，2 就是 001001001B、0 就是 00000011B、4 就是 10011001B。若要顯示某個位數，則輸出該位數所要顯示的資料，以及其掃瞄碼即可。




```
                ORG     0               ;程式從 0 位址開始
START:          MOV     P1, #FFH        ;關閉所有數字
                MOV     P2, #10011001B  ;輸出「4」的七節顯示碼
                MOV     P1, #11111110B  ;點亮最右邊位數
                CALL    DELAY           ;延遲 4ms
;=============================================================
                MOV     P1, #FFH        ;關閉所有數字
                MOV     P2, #00000011B  ;輸出「0」的七節顯示碼
                MOV     P1, #11111101B  ;點亮右邊第二個位數
                CALL    DELAY           ;延遲 4ms
;=============================================================
                MOV     P1, #FFH        ;關閉所有數字
                MOV     P2, #00000011B  ;輸出「0」的七節顯示碼
                MOV     P1, #11111011B  ;點亮右邊第三個位數
                CALL    DELAY           ;延遲 4ms
;=============================================================
                MOV     P1, #FFH        ;關閉所有數字
                MOV     P2, #00100101B  ;輸出「2」的七節顯示碼
                MOV     P1, #11110111B  ;點亮最左邊位數
                CALL    DELAY           ;延遲 4ms
                JMP     START           ;從頭開始掃瞄
;=============延遲約 4ms=(即 2usxR7xR7)====================
DELAY:          MOV     R7, #10
D1:             MOV     R6, #200
                DJNZ    R6, $
```

```
        DJNZ   R7, D1
        RET
        END
```

直接驅動七節顯示器實驗(ch4-2.asm)

操作

1. 依功能需求與電路結構撰寫程式，然後將該程式組譯與連結，以產生*.HEX 檔。
2. 請按圖 19 連接線路，再使用實體模擬器，載入該程式(*.HEX)，以模擬該電路的動作。若有非預期的狀況，則檢視線路的連接狀況，看看哪裡出問題？並將它記錄在實驗報告裡。
3. 若實體模擬功能正常，請將程式燒錄到 89C51，再把該 89C51 放入實體電路，以取代剛才的實體模擬器，然後直接送電，看看是否正常？
4. 撰寫實驗報告。

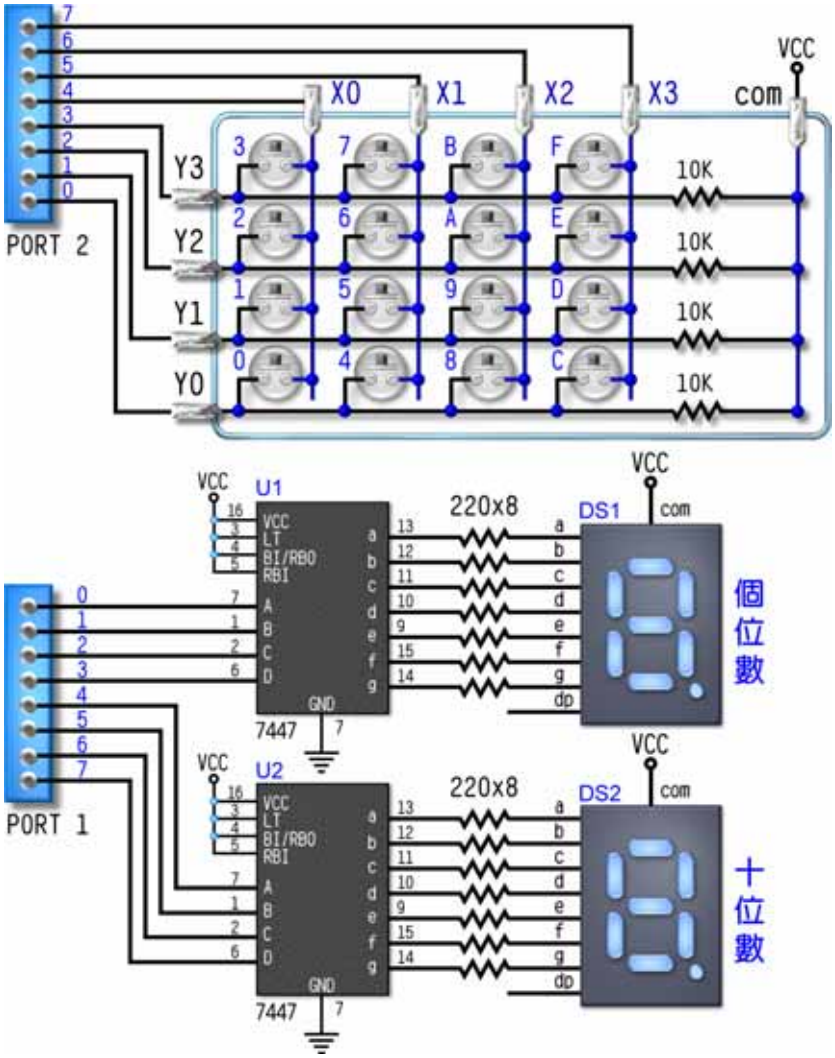
思考一下

1. 請修改程式，將掃瞄碼的產生與輸出，改採左移指令？
2. 在本實驗裡，數字的顯示器是由最右邊開始，由右而左循序顯示；請修改程式，讓數字的顯示器是由最左邊開始，由左而右循序顯示？

4-5-3 4x4 鍵盤與七節顯示器實例演練

功能說明

如圖 20 所示，兩個七節顯示器的 a、b、c...g，分別連接到限流電阻，再連接到 7447 的輸出，而個位數 7447 的輸入 DCBA，連接到 P1 的低四位元、十位數 7447 的輸入 DCBA，連接到 P1 的高四位元。P2 的高四位元連接 4x4 鍵盤的 Y3、Y2、Y1 及 Y0，P2 的低四位元連接 4x4 鍵盤的 X3、X2、X1 及 X0。當我們按「0」鍵時，七節顯示器顯示「00」；按「1」鍵時，七節顯示器顯示「01」...；按「F」鍵時，七節顯示器顯示「15」。



（圖20）**電路圖**

參考程式

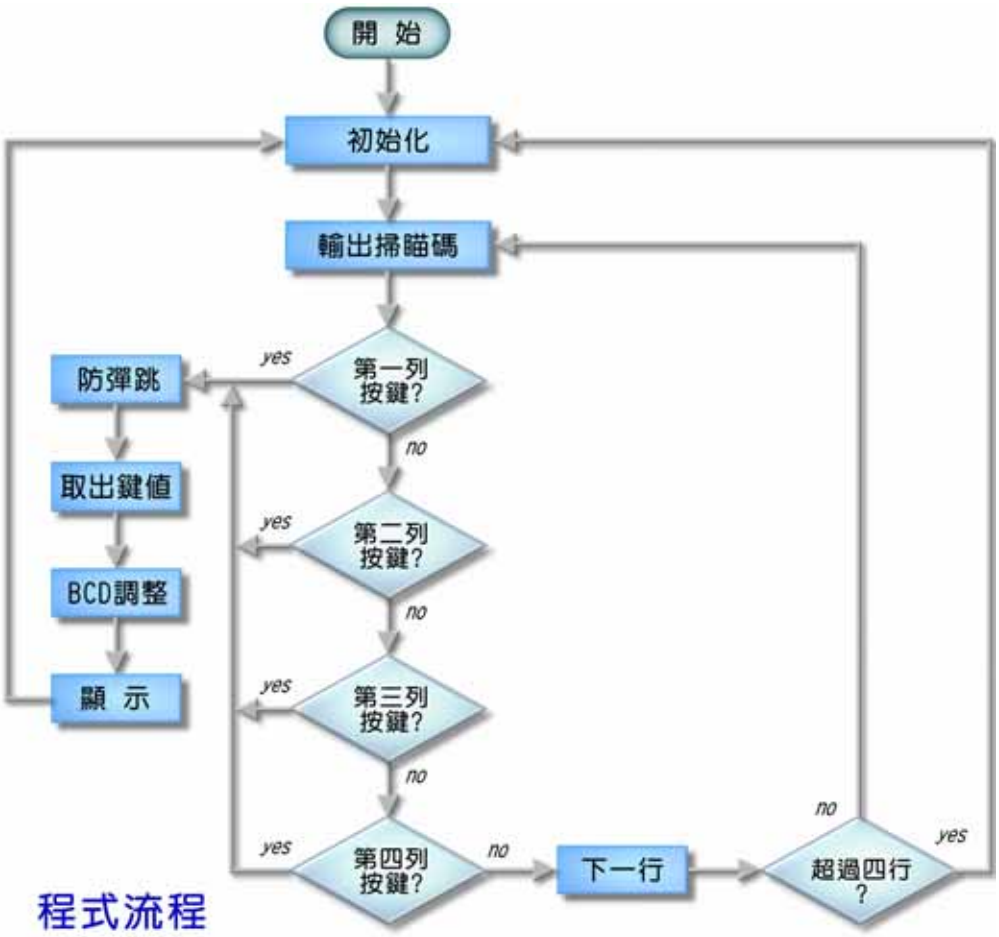
依功能需求與電路結構得知

P1.0～P1.3：個位數
P1.4～P1.7：十位數
P2.0～P2.3：鍵盤之 Y0～Y3
P2.4～P2.7：鍵盤之 X0～X3

整個功能建構在按鍵的掃瞄，而按鍵的值存放在 R0，掃瞄碼由 P2.4～P2.7 送到鍵盤的 X0～X3，第一個掃瞄碼爲 EFH(即 11101111B)，這個數值除了將掃瞄碼送到 P2.4～P2.7 外，還將 P2.0～P2.3 規劃成輸入功能。緊接著，利用 JNB 指令，直接判斷 P2.0，若 P2.0=0 表

示 X0 行的 Y0 列交集的按鍵被按下去(即 0 鍵)，則跳至 KEYIN 副程式將 R0 的內容顯示；若 P2.0≠0 表示 X0 行的 Y0 列交集的按鍵沒被按下去，則 R0 的內容加 1，指向下一個按鍵的值，再以同樣的方式判斷 P2.1...，以此類推。若 P2.0～P2.3 都沒被按下，則將掃瞄碼左移一位，再重新輸入掃瞄碼，以掃瞄下一行(Y1)，而下一行四個按鍵判斷方式，與前一行的判斷方式一樣，所以，我們可以同樣的方法，繼續掃瞄 Y1、Y2、Y3 行。當全部掃瞄完畢，再重新開始，如此週而復始，即可隨時反應這 16 個按鍵的狀況。

另外，在 KEYIN 副程式，必須先呼叫防彈跳副程式，才不會造成 CPU 的誤動作而無法正常進行。在輸出到 7447 之前，先把所要輸出的資料，以 DA 指令進行 BCD 調整，才能正確反應 A、B、C、D、E、F 按鍵。



程式流程

```
                ORG     0               ;程式從 0 位址開始
START:          MOV     R0, #0          ;按鍵初始值
                MOV     R1, #4          ;掃瞄行數
                MOV     R2, #EFH        ;掃瞄碼初始值
SCAN:           MOV     A, R2           ;指定掃瞄碼
                MOV     P2, A           ;掃瞄，並設定輸入模式
;=============ROW 0 ======================================
                JNB     P2.0, KEYIN     ;偵測第一列
                INC     R0              ;下一個按鍵
;=============ROW 1 ======================================
                JNB     P2.1, KEYIN     ;偵測第二列
                INC     R0              ;下一個按鍵
;=============ROW 2 ======================================
                JNB     P2.2, KEYIN     ;偵測第三列
                INC     R0              ;下一個按鍵
;=============ROW 3 ======================================
                JNB     P2.3, KEYIN     ;偵測第四列
                INC     R0              ;下一個按鍵
;=============NEXT COLUMN ================================
                MOV     A, R2           ;載入掃瞄碼
                RL      A               ;下一個掃瞄碼
                MOV     R2, A           ;存回掃瞄碼
                DJNZ    R1, SCAN        ;掃瞄下一行
                JMP     START           ;重新掃瞄
;=========================================================
KEYIN:          CALL    DEBOUNCE        ;呼叫防彈跳副程式
                MOV     A, R0           ;取回按鍵值
                DA      A               ;BCD 調整
                MOV     P1, A           ;顯示按鍵值
                JMP     START           ;重新掃瞄
;=============延遲約 16ms=(即 2usxR7xR7)====================
DEBOUNCE:       MOV     R7, #40
D1:             MOV     R6, #200
                DJNZ    R6, $
                DJNZ    R7, D1
                RET
                END
```

4x4 鍵盤與七節顯示器實驗(ch4-3.asm)

操作

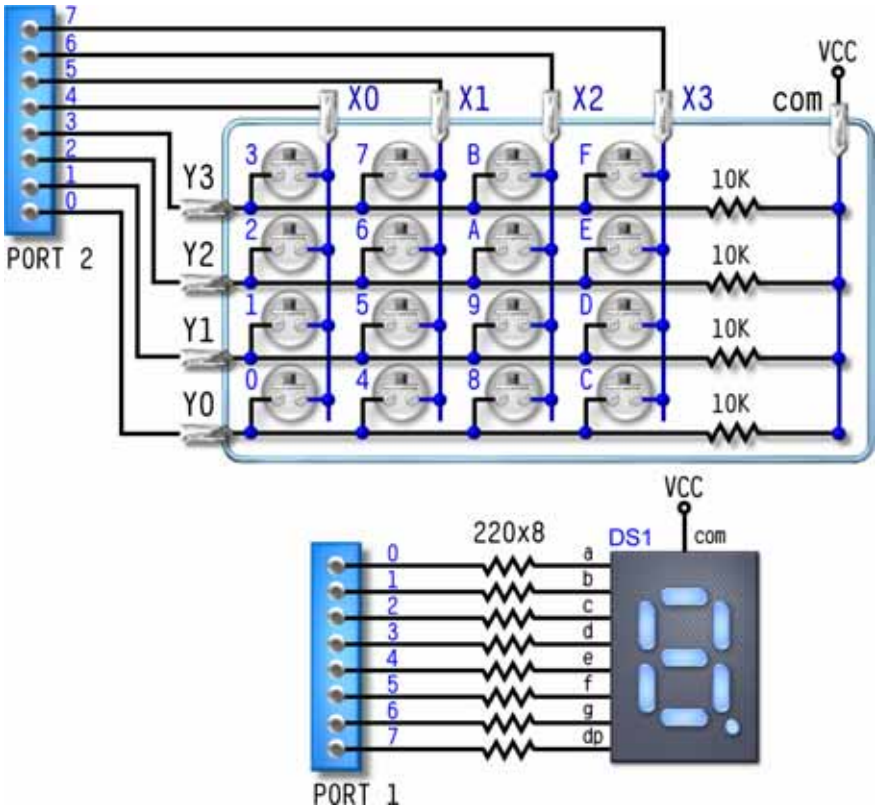
1. 依功能需求與電路結構撰寫程式，然後將該程式組譯與連結，以產生*.HEX 檔。
2. 請按圖 20 連接線路，再使用實體模擬器，載入該程式(*.HEX)，以模擬該電路的動作。若有非預期的狀況，則檢視線路的連接狀

況，看看哪裡出問題？並將它記錄在實驗報告裡。

3. 若實體模擬功能正常，請將程式燒錄到 89C51，再把該 89C51 放入實體電路，以取代剛才的實體模擬器，然後直接送電，看看是否正常？
4. 撰寫實驗報告。

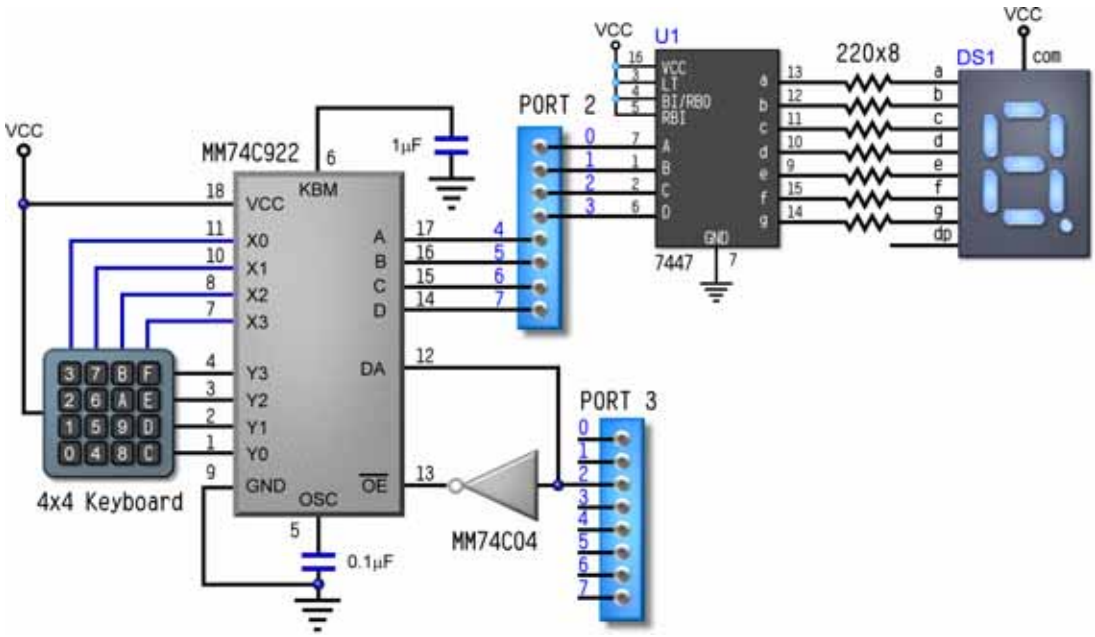
思考一下

1. 若將本實驗的程式中，不要呼叫防彈跳副程式，會有什麼結果？
2. 在本實驗裡，4x4 鍵盤可不可以連接 PORT 0？請詳述之？而七節顯示器可不可以接其它輸出入埠？
3. 若本實驗電路中的 PORT 2 接錯，使 P2.0～P2.3 與 P2.4～P2.7 對調，則程式要如何修改，才能有同樣的功能？
4. 在本實驗電路中，若將兩個 7447，直接由 CPU 輸出 0 到 F 的七節顯示碼，驅動一個七節顯示器，如圖 21 所示，程式應如何寫？



（圖21）**電路圖**

4-5-4 MM74C922 實例演練



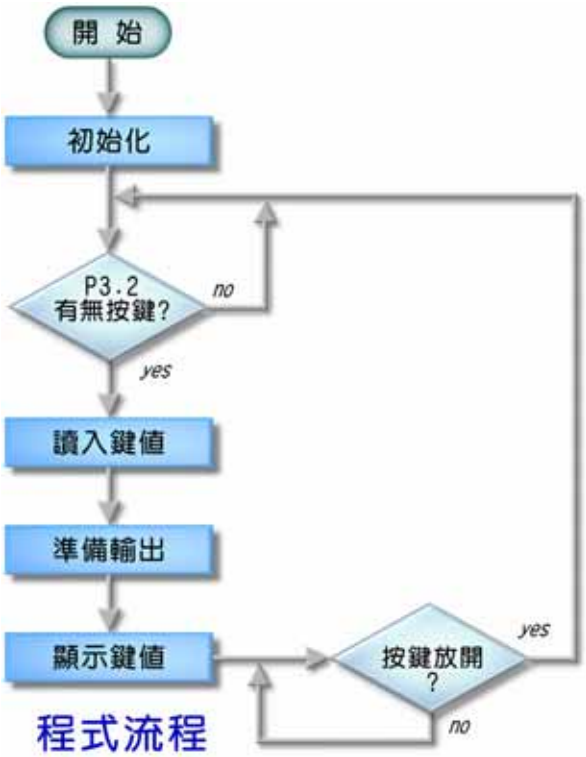
（圖22）**電路圖**

功能說明

如圖 23 所示，MM74C922 的資料 ABCD 連接到 8051 的 P2.4 到 P2.7，而 P2.0 到 P2.3 連接 7447，以輸出數字資料。MM74C922 的 DA 接腳連接到 8051 的 P3.2。若按下 4X4 鍵盤上的任一個鍵，則該鍵的數字將顯示在七節顯示器上。

參考程式

依功能需求與電路結構得知，由 P2.4 到 P2.7 讀取鍵盤資料，再將讀取到的資料，送到七節顯示器上。當 DA 接腳(連接到 P3.2)為 High 時，即讀取 2.4 到 P2.7，所以程式將持續判讀 DA 接腳，做為是否讀取鍵盤與顯示資料的依據。




```
                ORG     0               ;程式從 0 位址開始
START:          MOV     P2, #FFH        ;關閉七節顯示器，
                                        ;並將 P2.4 至 P2.7 設定為輸入模式
                SETB    P3.2            ;將 P3.2 設定為輸入模式
LOOP:           JNB     P3.2, LOOP      ;判讀是否按下按鍵
                MOV     A, P2           ;讀入鍵盤資料
                SWAP    A               ;將 A 的高四位元與低四位元互換
                ORL     A, #F0H         ;讓高四位元為高態
                MOV     P2, A           ;輸出到七節顯示器
RELEASE:        JB      P3.2, RELEASE   ;判讀是否放開按鍵
                JMP     LOOP            ;跳至 LOOP 形成一個迴圈
                END
```

MM74C922 實驗(ch4-4.asm)

操作

1. 依功能需求與電路結構撰寫程式，然後將該程式組譯與連結，以產生*.HEX 檔。
2. 請按圖 17 連接線路，再使用實體模擬器，載入該程式(*.HEX)，以模擬該電路的動作。若有非預期的狀況，則檢視線路的連接狀況，看看哪裡出問題？並將它記錄在實驗報告裡。
3. 若實體模擬功能正常，請將程式燒錄到 89C51，再把該 89C51 放

入實體電路，以取代剛才的實體模擬器，然後直接送電，看看是否正常？

4. 撰寫實驗報告。

思考一下

1. 在本實驗裡，只是在等待按鍵，程式可否做其它事情？
2. 在本實驗裡，有沒有「彈跳」的困擾？

4-6 即時練習

在本章裡探討 8051 的省電模式、鍵盤掃瞄、七節顯示器掃瞄，以及算術運算指令，相當實用。在此請試著回答下列問題，以確認對於此部分的認識程度。

1. 8051 系統提供哪兩種省電模式？如何進入省電模式？如何喚醒？
2. 在 MM74C922 與 MM74C923 都是鍵盤掃瞄 IC，其不同在哪裡？
3. 試說明何謂「高態掃瞄」？何謂「低態掃瞄」？
4. 如圖 5 所示，若將 X0～X3 連接 P2.4～P2.7、Y0～Y3 連接 P2.0～P2.3，則執行「MOV　P2, #FBH」，再執行「MOV　A, P2」，ACC 的內容變成 DFH，則是哪個按鍵被按下？
5. 試說明 7446、7447、7448 及 7449 之不同？
6. 試說明 74138 及 74139 之不同？
7. 試說明 ADD 指令與 ADDC 指令之異同？
8. 試說明 DA 指令之功能？
9. 試述如何應用 8051 之乘法指令？
10. 試述如何應用 8051 之除法指令？