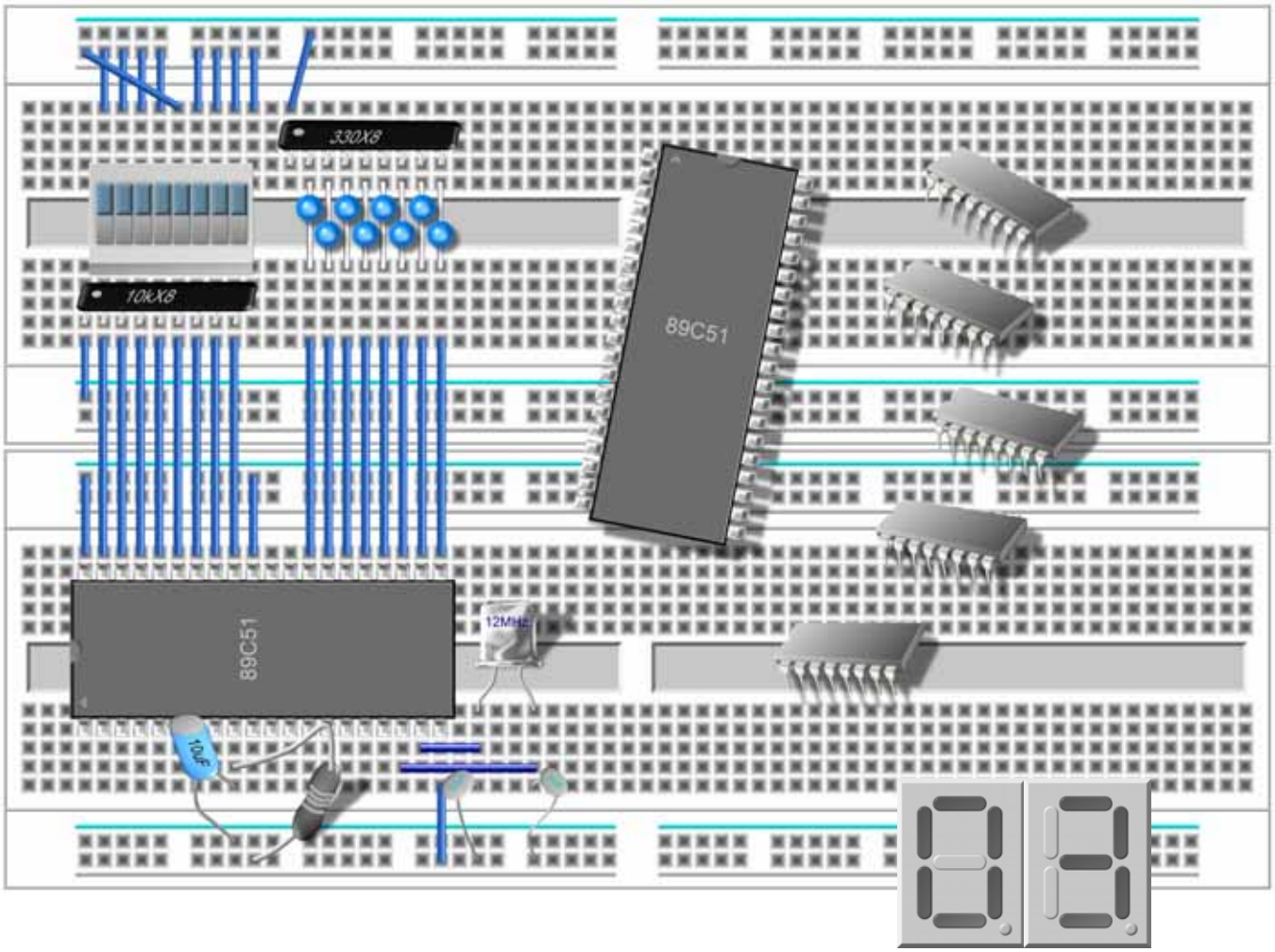


輸入埠之應用

本章內容豐富，主要包括三部分：

- 硬體部分：

  介紹了 8051 的時序分析與重置、輸入埠等。

  介紹常用的開關按鈕開關、指撥開關、BCD 指撥開關等，及其應用。

- 指令部分：

  詳細說明跳躍指令。

- 程式與實作部分：

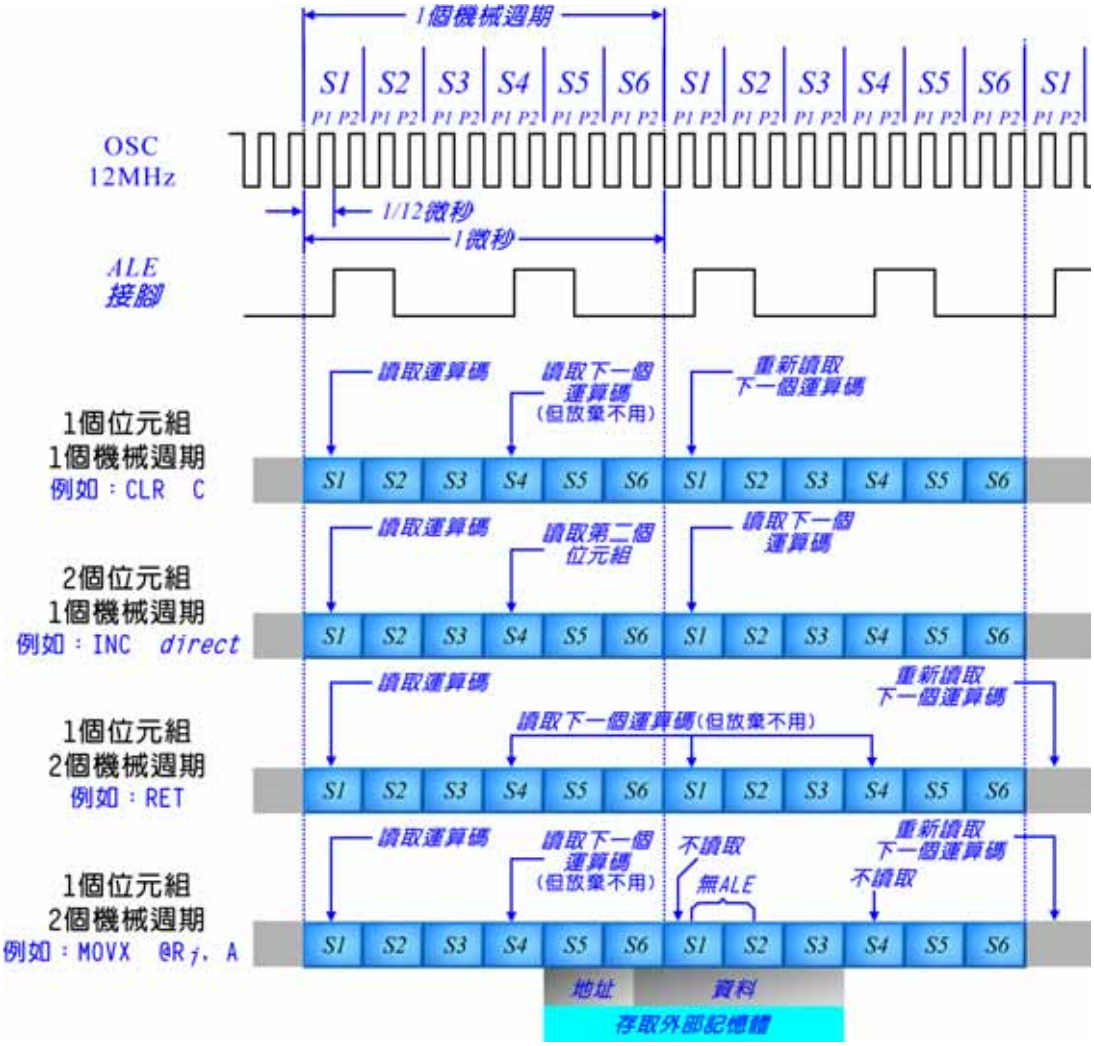
  指撥開關的應用、按鈕開關的應用、BCD 指撥開關的應用，以及防彈跳副程式等。

3-1 8051 時序分析與重置

在本單元裡將介紹 8051 的重置(RESET)與時序分析。

時序分析

時鐘脈波是微電腦系統的基本信號，在 1-2 節裡(1-10 頁)，我們曾經簡單地介紹 8051 的時鐘脈波。不管是採用內部的振盪電路，亦或由外部的時鐘脈波產生電路提供時鐘脈波，這個時鐘脈波將成爲整個系統運作的依據。以標準的 12HMz 時鐘脈波爲例，如下圖所示：



（圖1） **時序分析**

如上圖所示，一個機械週期(machine cycle)是由六個狀態週期(S1 到 S6)所構成，而每個狀態週期包括兩個時鐘脈波(即 P1、P2)。對於

12MHz 的時鐘脈波而言，一個脈波的週期爲 1/12 微秒，一個機械週期包含 12 個時鐘脈波，也就是 1 微秒。

在 8051 的 111 個指令裡，除了執行乘法與除法指令須要 4 個機械週期外，其餘指令都能在 1 個或 2 個機械週期執行完畢。儘管如此，有些指令的長度爲 1 byte、有些爲 2 bytes，還有少數指令爲 3 bytes。對於不同的指令，CPU 如何讀取(fatch)與執行呢？在此將配合在圖(1)簡要說明。首先是位址栓鎖致能接腳(ALE)，每個機械週期送出兩個脈波(分別是在 S1 及 S4 時)，以栓鎖 P0 輸出之位址(A0-A7)，CPU 將進行讀取記憶體的動作。而不同的指令類型，其動作分別說明於下：

1. 1 個機械週期、1byte 的指令，如 CLR　C 指令，在 S1 時讀取指令、S6 執行完畢；而在 S4 時讀取下個指令，但並不使用它，直到下個機械週期的 S1 時再重新讀取下個指令。

2. 1 個機械週期、2bytes 的指令，如 INC　*direct* 指令，在 S1 時讀取指令、S4 時讀取第二個 byte、S6 執行完畢。下個機械週期的 S1 時讀取下個指令...，以此類推。

3. 2 個機械週期、1byte 的指令，如 RET 指令，在 S1 時讀取指令，而在 S4、下個機械週期的 S1、S4 時分別讀取下個指令，由於指令尚未執行完畢，所以這三個階段的指令讀取，都會被放棄。直到第二個機械週期的 S6，指令執行完畢後，CPU 才會在第三個機械週期的 S1，重新讀取下個指令，才是有效的讀取。

4. 另外一種 2 個機械週期、1byte 的指令爲存取外部記憶體資料的指令，即 MOVX 指令。同樣在第一個機械週期的 S1 時讀取指令，而在 S4 讀取的下個指令，當然也會被放棄。S5 時 P0 送出的 A0 到 A7 位址將被放入栓鎖器，而 S6 到下個機械週期的 S3 之間，即由 P0 進行外部記憶體的資料存取。由於進行外部記憶體的存取，第二個機械週期的 S1 與 S4 不並進行讀取指令的動作。直到第三個機械週期的 S1，才會重新讀取下個指令。

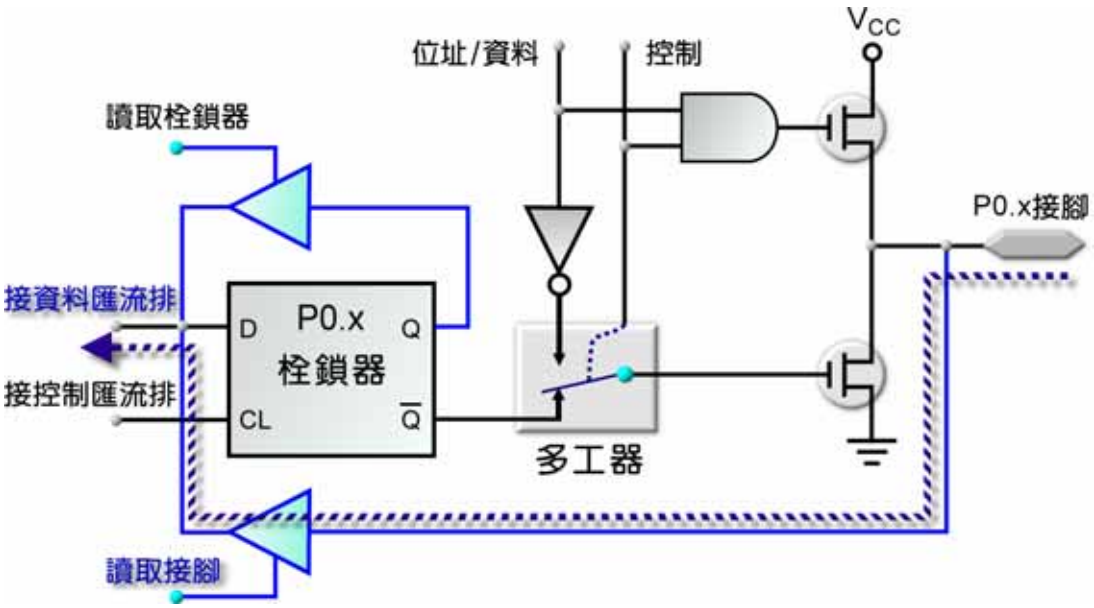
重置

對於微電腦系統而言，重置 RESET 是一項很重要的歸零調整動作。8051 的 RESET 是將高準位加到 RESET 接腳(第 9 腳)上，時間超過兩個機械週期以上，也就是 2 微秒。不管你的手有多快，按 8051 系統裡的 RESET 按鈕開關，都會超過 2 微秒，換言之，只要按 RESET 按鈕，就一定會使系統重置！當系統重置時，CPU 內部暫存器將回歸初始狀態(如下表所示)，程式將從 0000H 處開始執行。

暫存器	狀 態	暫存器	狀 態
ACC	00000000B	TMOD	00000000B
B	00000000B	TCON	00000000B
PSW	00000000B	T2CON	00000000B
SP	00000111B	TH0	00000000B
DPTR：		TL0	00000000B
DPH	00000000B	TH1	00000000B
DPL	00000000B	TL1	00000000B
P0	11111111B	TH2	00000000B
P1	11111111B	TL2	00000000B
P2	11111111B	RCAP2H	00000000B
P3	11111111B	RCAP2L	00000000B
IP：		SCON	00000000B
8051	XXX00000B	SBUF	未定
8052	XX000000B	PCON：	
IE：		NMOS	0XXXXXXXB
8051	0XX00000B	CHMOS	0XXX0000B
8052	0X000000B	PC	00000000B

3-2 輸入電路設計

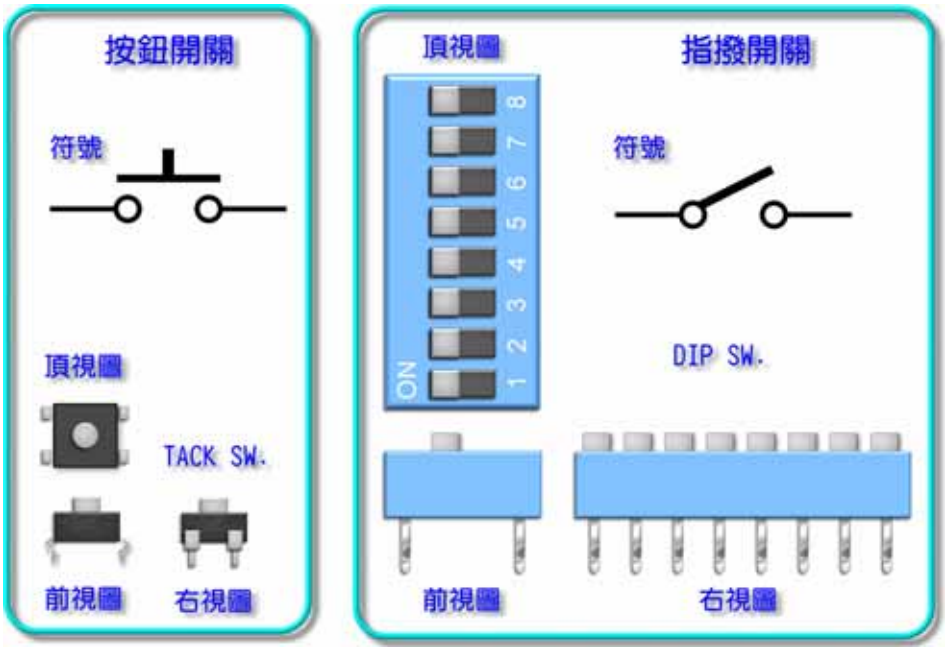
在 2-2 節裡，我們繪出 8051 四個輸出入埠的結構(圖 4、圖 5、圖 6 及圖 7)，雖然這四個輸出入埠的結構有些許的不同，但就輸入功能來看，這四個輸出入埠的結構幾乎完全一樣！基本上，輸入埠都是透過一個三態的緩衝器連接到 CPU 內部的資料匯流排，以 PORT 0 爲例，如下圖所示：



（圖2）　PORT 0 的輸入功能

內部「**讀取接腳**」線必須爲 1，外部資料才會通過緩衝器，而送到內部資料匯流排。這也就是爲什麼在輸入之前，必須送「1」到該輸出入埠，將該輸出入埠規劃成輸入功能的原因。

輸入裝置



（圖3）　**開關**

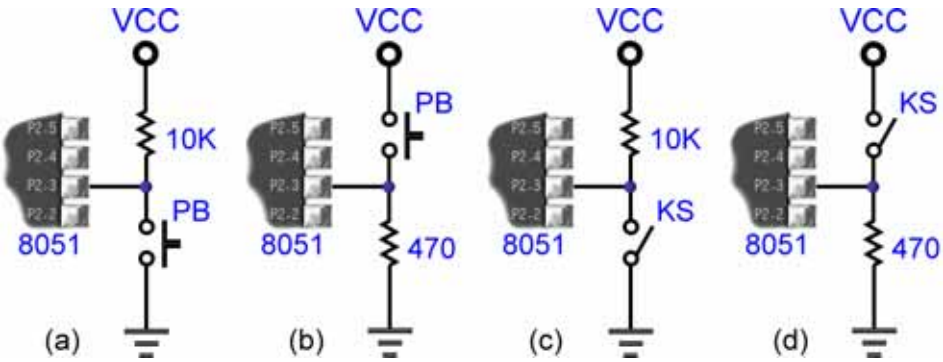
外部輸入裝置大概可分成兩類，分別是按鈕開關及單刀開關，電子

電路或微電腦電腦所使用的按鈕開關，大多採用 TACK Switch，若爲 a 接點，則按下按鈕時，其接點接通(on)、放開時，接點恢復爲不通(off)；反之，若爲 b 接點，則按下按鈕時，其接點不通(off)、放開時，接點恢復爲導通(on)，其實體圖與符號如圖 3 左邊所示。

電子電路或微電腦電腦所使用的單刀開關，大多採用指撥開關(即 DIP Switch)，若爲 4P 的 DIP Switch，則有四組單刀開關；若爲 8P 的 DIP Switch，則有八組單刀開關。通常會在 DIP Switch 上標示記號或「ON」，若將開關撥到記號或「ON」的一邊，則接點接通(on)、撥到另一邊則爲不通(off)，其實體圖與符號如圖 3 右邊所示。

輸入電路

若要以開關做爲輸入電路時，通常會接一個電阻到 VCC 或 GND，如下圖所示：



（圖4） **輸入電路**

如圖 4 之(a)所示，平時按鈕開關(PB)爲開路狀態，其中 10K 歐姆的電阻連接到 VCC，使輸入接腳上保持爲高態信號；若按下按鈕開關，則經由開關接地，輸入接腳上將變爲低態信號；放開開關時，輸入接腳上將恢復爲高態信號，如此將可產生一個負脈波。反之，在圖 4 之(b)中，平時按鈕開關爲開路狀態，其中 470 歐姆的電阻接地，使輸入接腳上保持爲低態信號；若按下按鈕開關，則經由開關接 VCC，輸入接腳上將變爲高態信號；放開開關時，輸入接腳上將恢復爲低態信號，如此將可產生一個正脈波。

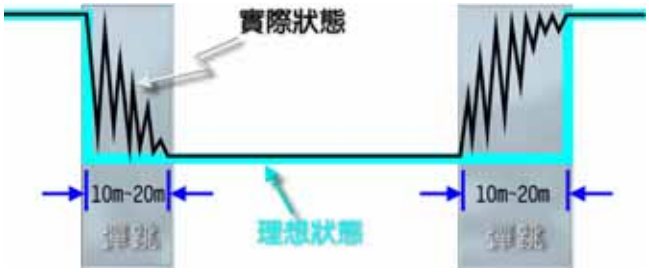
如圖 4 之(c)所示，若開關(KS)爲 off 狀態，其中 10K 歐姆的電阻連接到 VCC，使輸入接腳上保持爲高態信號；若將開關切換到 on 的

狀態，則經由開關接地，輸入接腳上將變爲低態信號，如此將可隨須要產生不同的準位。反之，在圖 4 之(d)中，若開關爲 off 狀態，其中 470 歐姆的電阻接地，使輸入接腳上保持爲低態信號；若將開關切換到 on 的狀態，則經由開關接 VCC，輸入接腳上將變爲高態信號，如此將可隨須要產生不同的準位。

通常按鈕開關是使用在產生邊緣觸發的場合，而單刀開關使用在產生準位觸發的場合。

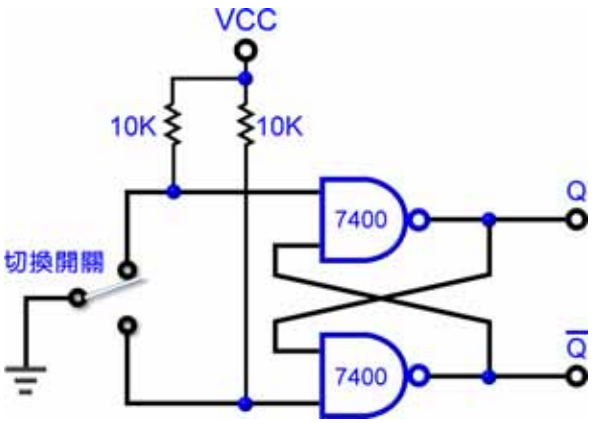
防彈跳電路

剛才所介紹的輸入電路中，開關的動作是理想的狀態，如果仔細分析開關的真實動作，將可發現許多非預期的狀況，如下圖所示：



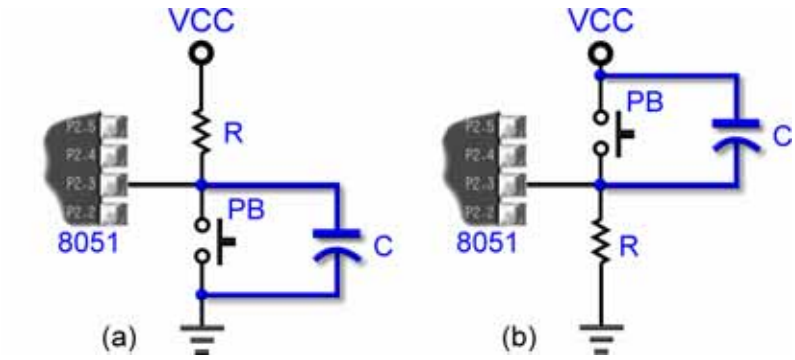
（圖5）　**開關的動作**

這種非預期的狀況稱爲**彈跳**(bounce)，這種忽高忽低，忽而非高非低，可說是不折不扣的雜訊。如果要避免這種現象，可使用一個切換開關(c 接點)及互鎖電路，組成一個防彈跳電路(debounce)，如下圖所示：



（圖6）　**互鎖電路**

雖然這個電路可降低彈跳所產生的雜訊，但所須的零件較多、所佔的電路面積較大，徒增成本與電路的複雜度，非不得已，已很少使用了。我們可利用一個簡單的 RC 電路，以壓制彈跳電壓，如下圖所示：



（圖7） RC 防彈跳電路

以(a)圖爲例，當按下按鈕開關時，開關第一次接觸時，即將電容器短路，使電容器快速放電(電阻爲 0)，電容器兩端電壓迅速爲 0；開關彈回(開路)時，整個電路形成RC充電電路，其時間常數爲RC，電容器兩端的電壓V_C爲

$$V_C = VCC \times (1 - e^{-\frac{t}{RC}})$$

通常低態準位可定義爲 $0.3 \times V_{CC}$，如果電容器兩端的電壓V_C低於 $0.3 \times V_{CC}$，即可視爲低準位，而彈跳的效應自然消失，因此

$$V_C = VCC \times (1 - e^{-\frac{t}{RC}}) < 0.3 \times VCC$$

即 $1 - e^{-\frac{t}{RC}} < 0.3$，兩邊減 1 可得 $-e^{-\frac{t}{RC}} < -0.7$，再把兩邊改號，小於變大於，即

$$e^{-\frac{t}{RC}} > 0.7$$

兩邊取對數

$$-\frac{t}{RC} > \ln 0.7 \cong -0.357$$

兩邊再改號，即

$$\frac{t}{RC} < 0.357$$

彈跳的時間約在 10mS 到 20mS 之間，以 10mS 爲例，若 R=10K 歐姆，則

$$\frac{10m}{10K \times C} < 0.357 \text{ ，}$$

$$\frac{10m}{10K \times 0.357} < C \text{ ，}$$

$$C > \frac{1\mu}{0.357} \cong 2.8\mu F$$

若是 20mS，則 C>5.6μF，因此，C 的值可定於 2.8μF 到 5.6μF 之間，筆者的習慣是 R=10K 歐姆時，C 採用 3.3μF；若 R 爲 100K 歐姆，則 C 採用 0.33μF。

當放開按鈕開關時，開關彈開時，即將電容器兩端開路，使電容器開始充電，當然電容器兩端電壓不會立即爲高準位；而開關再彈回(短路)時，又將好不容易充電的電容器兩端短路。因此，電容器兩端電壓在彈跳期間，保持爲低態，而不隨彈跳起舞。直到彈跳期間過後，電容器兩端的電壓才穩定上昇，絲毫不受彈跳影響。

軟體防彈跳

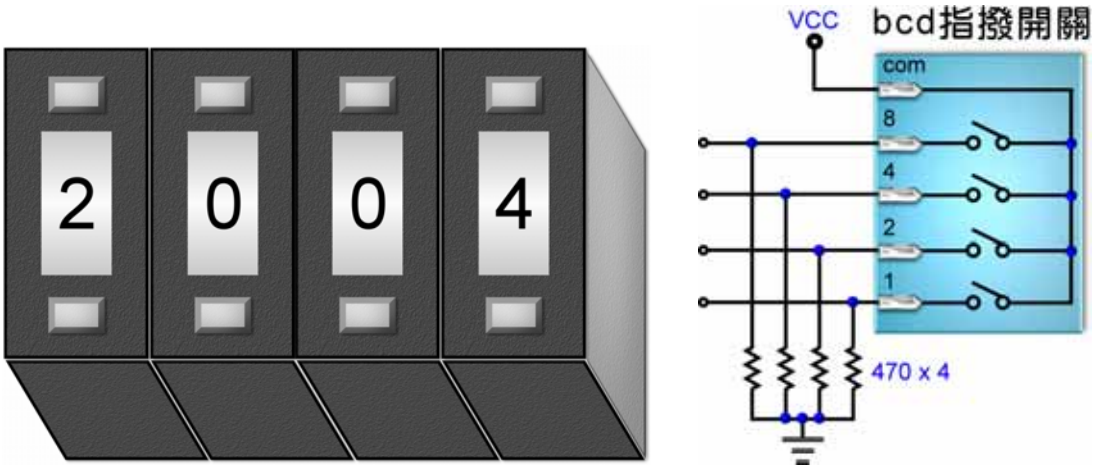
不管怎樣，利用硬體來抑制彈跳的雜訊，一定會增加電路的複雜性與成本！而我們只要在軟體上下點功夫，避開產生彈跳的那 10 至 20 毫秒，即可達到防彈跳的效果。怎麼做呢？只要在讀入第一個轉態的輸入信號，即執行 10 至 20 毫秒的延遲副程式(通常是 16 毫秒即可)，以圖 4 的(a)爲例，若按下按鈕開關 PB，程式將執行 BEEP 副程式，發出一個「嗶」聲，如下所示(其中的 BEEP 副程式省略)：

```
                ORG     0               ;程式從 0 位址開始
START:          SETB    P2.3            ;將 P2.3 規劃為輸入功能
                JB      P2.3, $         ;檢查 P2.3 有無變化
                CALL    DELAY16         ;呼叫 16ms 延遲副程式(防彈跳)
                CALL    BEEP            ;呼叫 BEEP 副程式發出「嗶」聲
                 :
                 :
;=====================================================
DELAY16:                                ;延遲副程式(16 毫秒)
                MOV     R7, #40         ;R7 暫存器載入 40 次數
D1:             MOV     R6, #200        ;R6 暫存器載入 200 次數
                DJNZ    R6, $           ;本列執行 R6 次
                DJNZ    R7, D1          ;D1 迴圈執行 R7 次
                RET                     ;返回主程式
                 :
                 :
```

軟體防彈跳

BCD 指撥開關

BCD 指撥開關是一個能產生 BCD 碼的指撥開關，如下圖所示爲四個位數的 BCD 指撥開關，以及其中一個位數的內部電路：



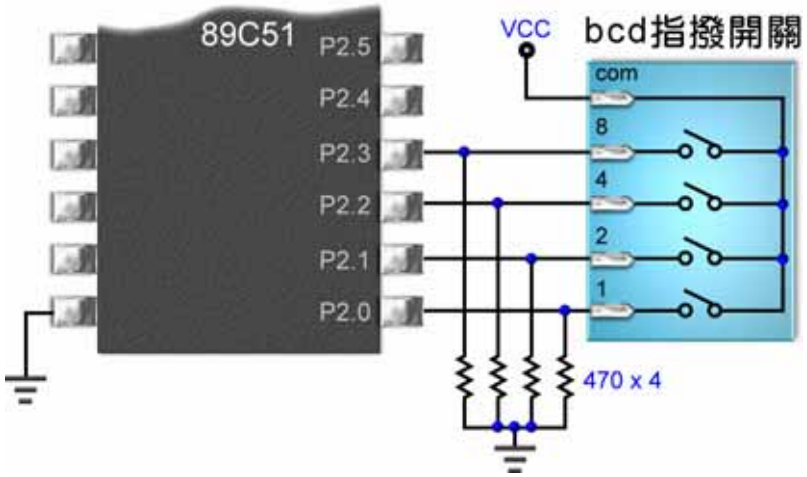
（圖8） BCD 指撥開關

每個位數都有共同端 com 與 8、4、2、1 等四個輸出端，通常是把共同端連接 VCC，而在其它四個輸出端各接一個電阻器(10K 歐姆即可)到接地端。當我們按 BCD 指撥開關上方的按鈕，則數字增加；按下方的按鈕，則數字減少。有些 BCD 指撥開關是在其右邊(或左邊)，以輪盤的方式切換數字。其數字是從 0 到 9，相對於其輸出端，

如下表所示：

數字	8 輸出端	4 輸出端	2 輸出端	1 輸出端
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

BCD 指撥開關之使用方式非常簡單，直接並接於輸入埠即可，如下圖所示就是由 P2 的低四位元輸入：



（圖9） BCD 指撥開關之使用

3-3 跳躍指令

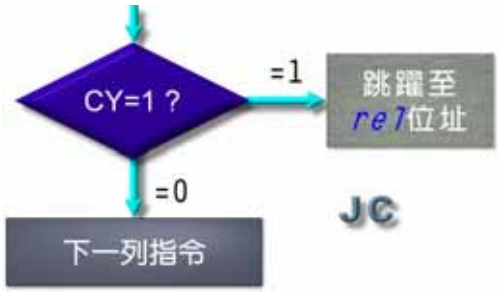
跳躍指令的功能是改變程式流程的指令，可讓程式更活潑。跳躍指令包括 22 個指令，在此將它們分爲 6 大類來介紹：

條件跳躍指令

條件跳躍指令的功能是以運算元爲跳躍與否的條件，而運算元爲一個位元或狀態，例如進位位元 CY、可位元定址的某個位元，或 ACC 內的

狀態等。

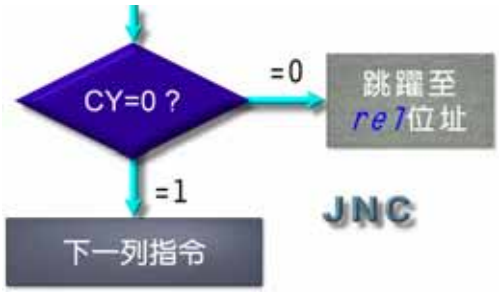
- JC *rel*
  - **說明**：以進位位元(CY)爲程式分岐的依據，若 CY=1，則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若 CY=0，則繼續執行下一列指令，即



  - **組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令： JC　　LOOP 若執行前 CY=0，將繼續執行下一列指令。 若執行前 CY=1，將執行 LOOP 位址之指令。

- JNC *rel*
  - **說明**：以進位位元(CY)爲程式分岐的依據，若 CY=0，則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若 CY=1，則繼續執行下一列指令，即



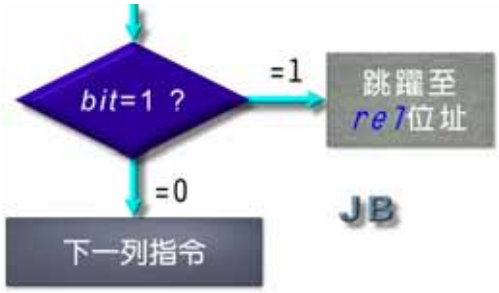
  - **組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令： JNC　　LOOP

若執行前 **CY=1**，將繼續執行下一列指令。

若執行前 **CY=0**，將執行 **LOOP** 位址之指令。

- JB　*bit*, *rel*
  - **說明**：以 *bit* 位元爲程式分岐的依據，若 *bit* =1，則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若 *bit* =0，則繼續執行下一列指令，即



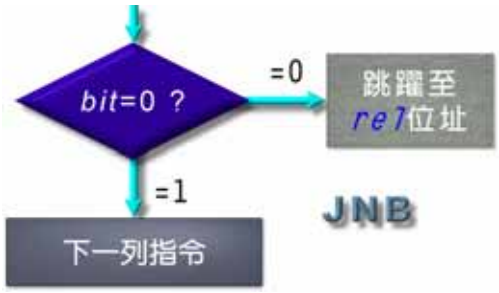
  - **組譯後大小**：3 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令： **JB　　P1.0, LOOP**

若執行前 **P1.0=0**，將繼續執行下一列指令。

若執行前 **P1.0=1**，將執行 **LOOP** 位址之指令。

- JNB　*bit*, *rel*
  - **說明**：以 *bit* 位元爲程式分岐的依據，若 *bit* =0，則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若 *bit* =1，則繼續執行下一列指令，即



  - **組譯後大小**：3 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

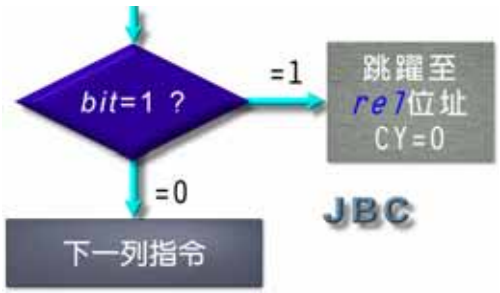
指令： JB P1.0, LOOP

若執行前 P1.0=1，將繼續執行下一列指令。

若執行前 P1.0=0，將執行 LOOP 位址之指令。

JBC *bit*, *rel*

說明：以 *bit* 位元爲程式分岐的依據，若 *bit* =1，則跳躍到 *rel* 的相對位址，並清除進位位元(即 CY=0)，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若 *bit* =0，則繼續執行下一列指令，即



組譯後大小：3 bytes **執行時間**：24 個時鐘脈波

範例：

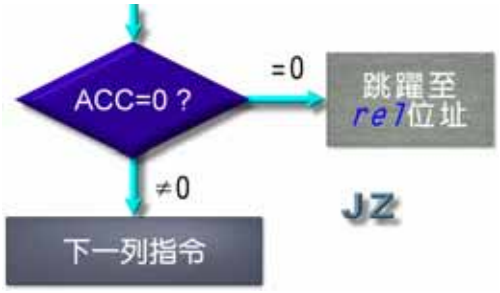
指令： JBC P1.3, LOOP

若執行前 P1.3=0，將繼續執行下一列指令。

若執行前 P1.3=1，將執行 LOOP 位址之指令，並使 CY=0。

JZ *rel*

說明：以 ACC 的內容爲程式分岐的依據，若 ACC =0，則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若 ACC ≠0，則繼續執行下一列指令，即



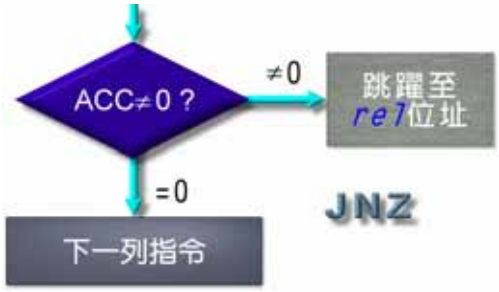
組譯後大小：2 bytes **執行時間**：24 個時鐘脈波

▶**範例**：

指令： JZ　　LOOP
若執行前 ACC≠0，將繼續執行下一列指令。
若執行前 ACC=0，將執行 LOOP 位址之指令。

- JNZ　*rel*

▶**說明**：以 ACC 的內容爲程式分岐的依據，若 ACC ≠0，則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若 ACC =0，則繼續執行下一列指令，即



▶**組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波

▶**範例**：

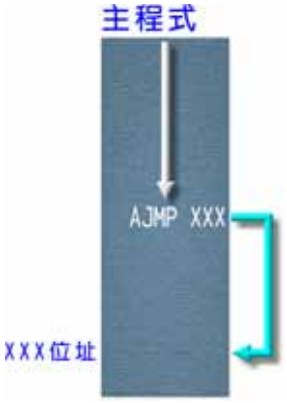
指令： JNZ　　LOOP
若執行前 ACC=0，將繼續執行下一列指令。
若執行前 ACC≠0，將執行 LOOP 位址之指令。

無條件跳躍指令

無條件跳躍指令就是不管三七二十一，程式即跳到指定的位置。

- AJMP *addr11*

▶**說明**：改變程式的流程，執行(*addr11*)位址的指令，也就是 2K 範圍內的跳躍，又稱爲近程跳躍，即



▶**組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波

▶**範例**：

指令： AJMP L1

執行前，PC 指向「AJMP L1」指令的位址。

執行後，PC 指向 L1 標籤的位址，將程式流程改至 L1 位址。

LJMP *addr16*

▶**說明**：改變程式的流程，執行(*addr16*)位址的指令，也就是 64K 範圍內的跳躍，又稱爲遠程跳躍，即

▶**組譯後大小**：3 bytes　　**執行時間**：24 個時鐘脈波

▶**範例**：

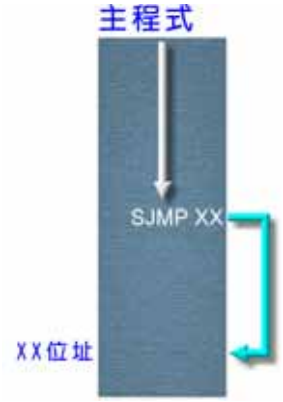
指令： LJMP L2

執行前，PC 指向「LJMP L2」指令的位址。

執行後，PC 指向 L2 標籤的位址，將程式流程改至 L2 位址。

SJMP *rel*

說明：改變程式的流程，執行(*rel*)位址的指令，可往往前 128 個位置、往後 127 個位置的跳躍，又稱爲短程跳躍或相對位置跳躍，即



組譯後大小：2 bytes　　**執行時間**：24 個時鐘脈波

範例：

指令： SJMP　X1
執行前，PC 指向「SJMP X1」指令的位址。
執行後，PC 指向 X1 標籤的位址，將程式流程改至 X1 位址。

JMP @A+DPTR

說明：改變程式的流程，跳至(A+DPTR)位址的指令。

組譯後大小：1 bytes　　**執行時間**：24 個時鐘脈波

範例：

指令： JMP　@A+DPTR
執行前，PC 指向「JMP X1」指令的位址，A=02H、DPTR=1234H。
執行後，PC 指向 1236H 位址，將程式流程改至 1236H 位址。

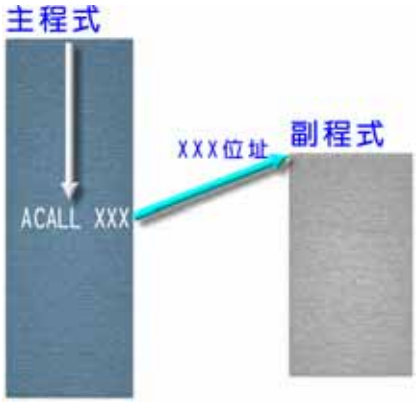
副程式操作指令

副程式操作指令包括在主程式呼叫副程式的指令，以及由副程式返回主程式的指令。

ACALL *addr11*

說明：呼叫(*addr11*)位址的副程式，也就是 2K 範圍內的副程式，

又稱爲近程呼叫，即



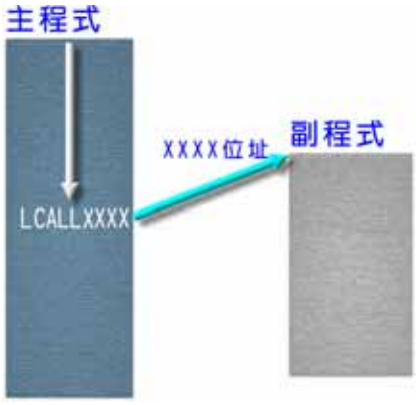
▶**組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波

▶**範例**：

指令： ACALL　DELAY
執行前，PC 指向「ACALL DELAY」指令的位址。
執行後，PC 指向 DELAY 標籤的位址，執行 DELAY 副程式。

LCALL *addr16*

▶**說明**：呼叫(*addr16*)位址的副程式，也就是 64K 範圍內的副程式，又稱爲遠程呼叫，即

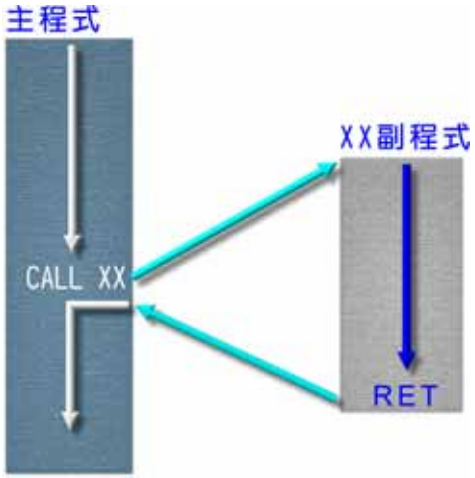


▶**組譯後大小**：3 bytes　　**執行時間**：24 個時鐘脈波

▶**範例**：

指令： LCALL　DISP
執行前，PC 指向「LCALL DISP」指令的位址。
執行後，PC 指向 DISP 標籤的位址，執行 DISP 副程式。

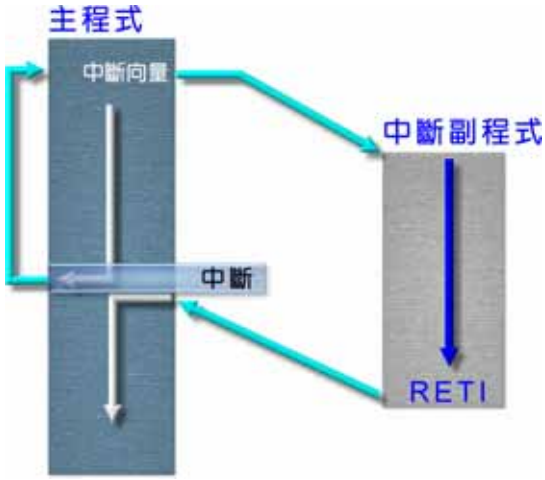
- RET
  - **說明**：由副程式返回主程式，即



  - **組譯後大小**：1 byte　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令： RET **執行前，PC 指向副程式 RET 指令位址。** **執行後，PC 指向主程式呼叫該副程式指令的下一個位址。**

- RETI
  - **說明**：由中斷副程式返回主程式，即



  - **組譯後大小**：1 byte　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令： RETI

執行前，**PC** 指向中斷副程式 **RETI** 指令位址。

執行後，**PC** 取回主程式中斷之前的 **PC** 值，執行其下一個位址的指令。

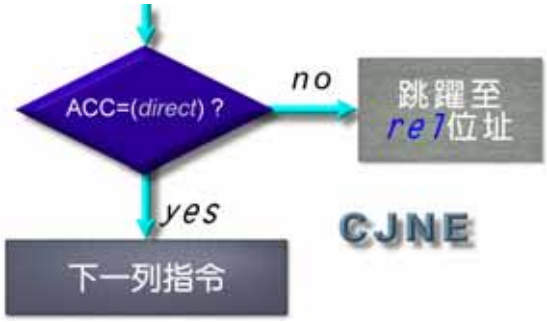
比較式跳躍指令

比較式跳躍指令的功能是將兩個運算元比較後，做爲跳躍與否的依據，如下圖所示：

其中的運算元 1 可爲 ACC、R*n*、(@R*i*)等，運算元 2 可爲 *direct*、*data* 等。

- CJNE A, *direct*, *rel*

  說明：將 ACC 的內容與(*direct*)位址的內容比較，若不相等則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若相等則繼續執行下一列指令，即



  組譯後大小：3 bytes　　**執行時間**：24 個時鐘脈波

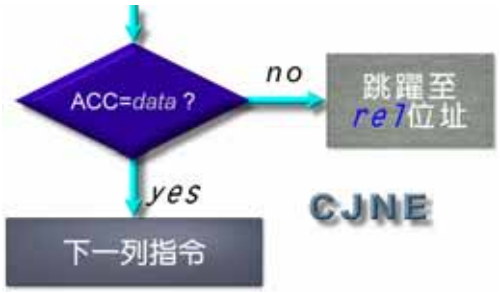
  範例：

指令： **CJNE A, 20H, LOOP**

若執行前 **ACC=(20H)**，執行後將繼續執行下一列指令。

若執行前 **ACC≠(20H)**，執行後將執行 **LOOP** 位址之指令。

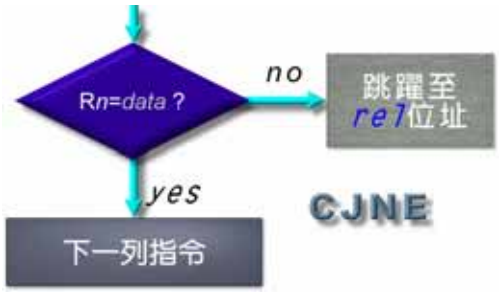
- CJNE　A, #*date*, *rel*
  - **說明**：將 ACC 的內容與立即值 *data* 比較，若不相等則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若相等則繼續執行下一列指令，即



  - **組譯後大小**：3 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

> **指令： CJNE　A, #FFH, LOOP**
>
> **若執行前 ACC=FFH，將繼續執行下一列指令。**
>
> **若執行前 ACC≠FFH，將執行 LOOP 位址之指令。**

- CJNE　R*n*, #*date*, *rel*
  - **說明**：將 R*n* 的內容與立即值 *data* 比較，若不相等則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若相等則繼續執行下一列指令，即

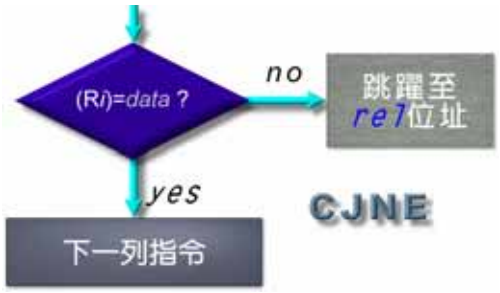


  - **組譯後大小**：3 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

> **指令： CJNE　R1, #100, LOOP**
>
> **若執行前 R1=100，將繼續執行下一列指令。**
>
> **若執行前 R1≠100，將執行 LOOP 位址之指令。**

- CJNE @R*i*, #*date*, *rel*

  說明：將(R*i*)的內容與立即值 *data* 比較，若不相等則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若相等則繼續執行下一列指令，即



  組譯後大小：3 bytes　　**執行時間**：24 個時鐘脈波

  範例：

> **指令： CJNE @R0, #100, LOOP**
>
> **若執行前(R0)=100，將繼續執行下一列指令。**
>
> **若執行前(R0)≠100，將執行 LOOP 位址之指令。**

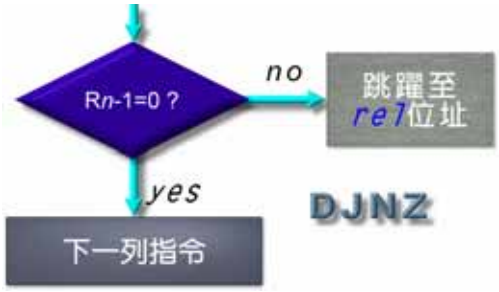
計次迴圈指令

計次迴圈指令的功能是以暫存器或記憶體位址爲計數器，每次執行一次就把計數器減 1，若計數器爲 0 即跳躍，如下圖所示：

其中的運算元可爲 R*n*、*direct* 等，其中所存放的就是迴圈所要執行的次數。

- DJNZ Rn, *rel*

  說明：將 R*n* 的內容減 1 後，若 R*n*≠0，則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若 R*n*=0，則繼續執行下一列指令，即



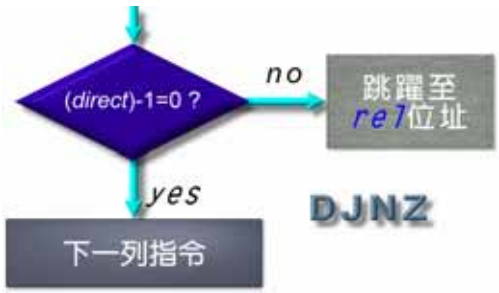
▶**組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波

▶**範例**：

指令： DJNZ　R1, LOOP
若執行前 R1>1，將繼續執行下一列指令。
若執行前 R1=1，將執行 LOOP 位址之指令。

● DJNZ　*direct*, *rel*

▶**說明**：將(*direct*)的內容減 1 後，若(*direct*)的內容≠0，則跳躍到 *rel* 的相對位址，而 *rel* 的範圍是從下個位置算起，往前 128 個位置、往後 127 個位置，通常是利用標籤來指引跳躍的位置。若(*direct*)的內容=0，則繼續執行下一列指令，即



▶**組譯後大小**：3 bytes　　**執行時間**：24 個時鐘脈波

▶**範例**：

指令： DJNZ　30h, LOOP
若執行前(30H)>1，將繼續執行下一列指令。
若執行前(30H)=1，將執行 LOOP 位址之指令。

無作為指令

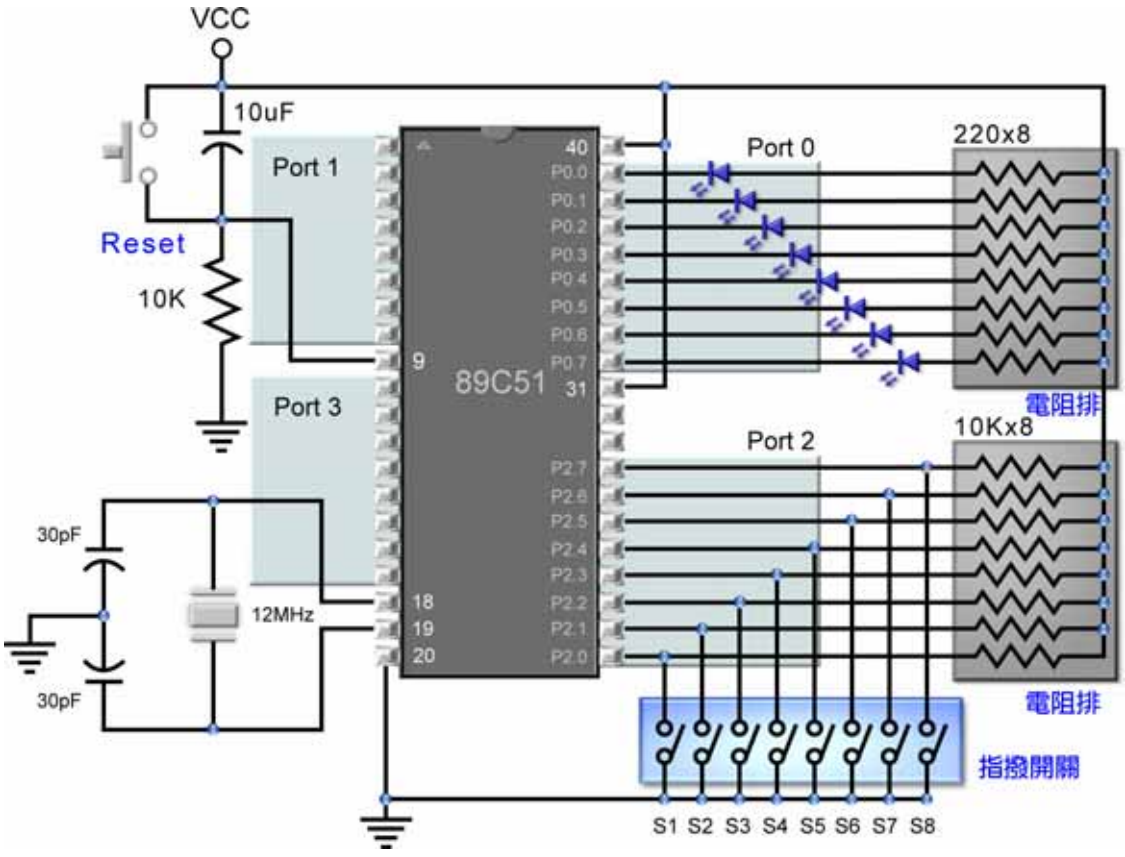
無作爲指令的功能是只耗時間，不做任何動作的指令。

- NOP
  - **說明**：沒有任何動作。
  - **組譯後大小**：1 bytes　　**執行時間**：12 個時鐘脈波

3-4 實例演練

在本單元裡提供四個範例，如下所示：

3-4-1 指撥開關實例演練



（圖10）**電路圖**

- **功能說明**

如圖 10 所示，指撥開關的狀態由 P2 輸入，而其狀態將反應到 P0

所連接的 LED 上。若 P2.0 所連接的開關 on，則 P0.0 所連接的 LED 將會亮、若 P2.0 所連接的開關 off，則 P0.0 所連接的 LED 將不亮...，以此類推。

參考程式

依功能需求與電路結構得知，當指撥開關 on 時，將可由其連接的輸入埠讀取到低準位(即 0)；而若要連接在 P0 的 LED 亮，則由 P0 輸出低準位即可。因此，在程式裡，只要將 P2 讀取到的指撥開關直接輸出到 P0 即可。當然，可不要忘了，事先將 P2 規劃成輸入功能。


```
                ORG     0               ;程式從 0 位址開始
START:          MOV     P2, #FFH        ;將 P2 規劃為輸入功能
LOOP:           MOV     A, P2           ;讀入指撥開關狀況
                MOV     P0, A           ;將開關狀況反應到 P0
                JMP     LOOP            ;跳至 LOOP 形成一個迴圈
                END
```

指撥開關實驗(ch3-1.asm)

操作

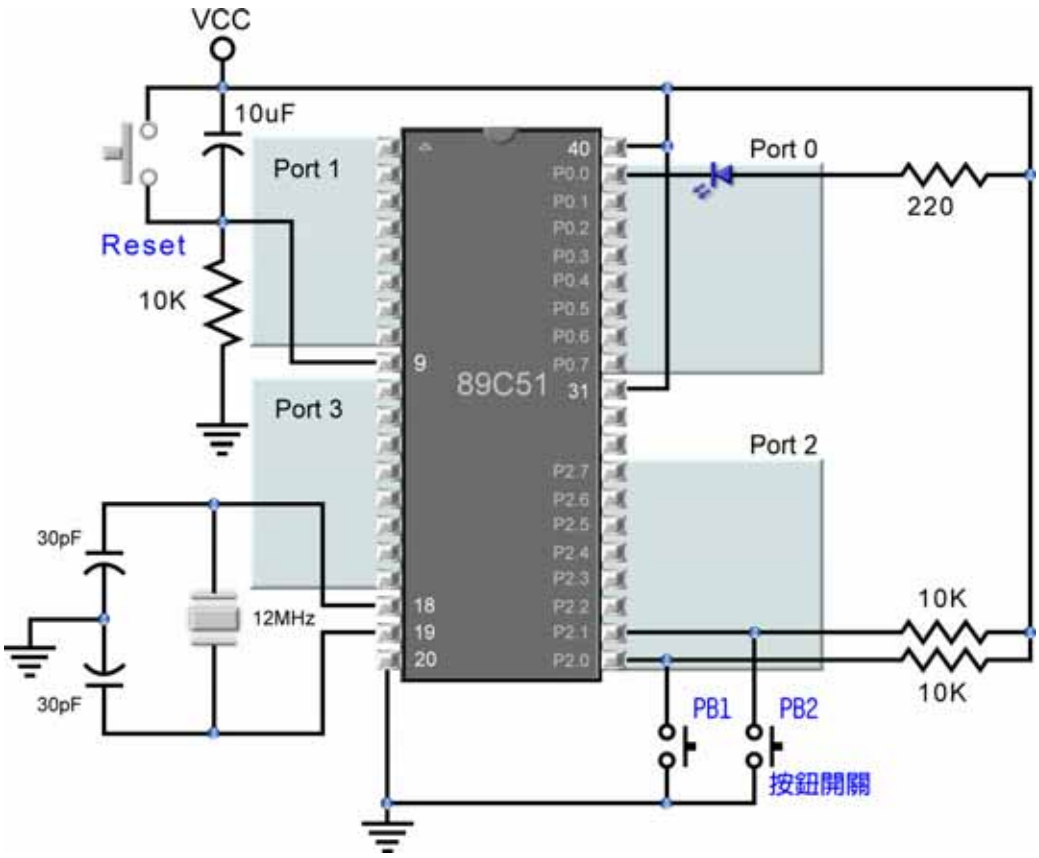
1. 依功能需求與電路結構撰寫程式，然後將該程式組譯與連結，以產生*.HEX 檔。
2. 請利用 AVSIM51 之類的模擬軟體，模擬其功能。若有非預期的狀況，則檢視原始程式，看看哪裡出問題？並將它記錄在實驗報告裡。
3. 請按圖 10 連接線路，再使用實體模擬器，載入該程式(*.HEX)，以模擬該電路的動作。若有非預期的狀況，則檢視線路的連接狀況，看看哪裡出問題？並將它記錄在實驗報告裡。

4. 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？
5. 撰寫實驗報告。

思考一下

1. 在本實驗裡，有沒有「彈跳」的困擾？
2. 若希望指撥開關中的 S1、S3、S5 三個開關都 on，則前四個 LED 亮；S2 或 S4 或 S6 開關 on，則後四個 LED 亮；S7 及 S8 開關 on，則所有 LED 全亮，程式應如何撰寫？
3. 若將指撥開關換成一般家裡牆壁上的開關，而 LED 換成繼電器 (RELAY)，是否可做爲家裡的負載控制？

3-4-2 按鈕開關實例演練



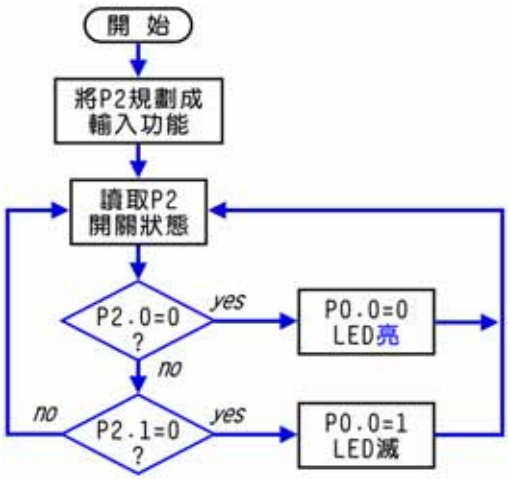
（圖11）**電路圖**

功能說明

如圖 11 所示，若按一下 PB1，則 P0.0 所連接的 LED 亮；若按一下 PB2，則關閉 P0.0 所連接的 LED(不亮)。

參考程式

依功能需求與電路結構得知，當按下按鈕開關時，將可由其連接的輸入埠讀取到低準位(即 0)；而若要連接在 P0.0 的 LED 亮，則由 P0.0 輸出低準位即可。因此，在程式裡，若 P2.0 讀取到 0，則將 P0.0 設為 0；若 P2.1 讀取到 0，則將 P0.0 設為 1。同樣地，可不要忘了，事先將 P2 規劃成輸入功能。




```
                ORG     0               ;程式從 0 位址開始
START:          MOV     P2, #FFH        ;將 P2 規劃為輸入功能
LOOP:           JNB     P2.0, ON        ;若 PB1 on，則跳至 ON
                JNB     P2.1, OFF       ;若 PB2 on，則跳至 OFF
                JMP     LOOP            ;跳至 LOOP 形成一個迴圈
ON:             CLR     P0.0            ; P0.0=0
ON_1:           JB      P2.0, LOOP      ; DEBOUNCE
                JMP     ON_1            ;跳至 ON_1
OFF:            SETB    P0.0            ; P0.0=1
OFF_1:          JB      P2.1, LOOP      ; DEBOUNCE
                JMP     OFF_1           ;跳至 OFF_1
                END
```

指撥開關實驗(ch3-2.asm)

操作

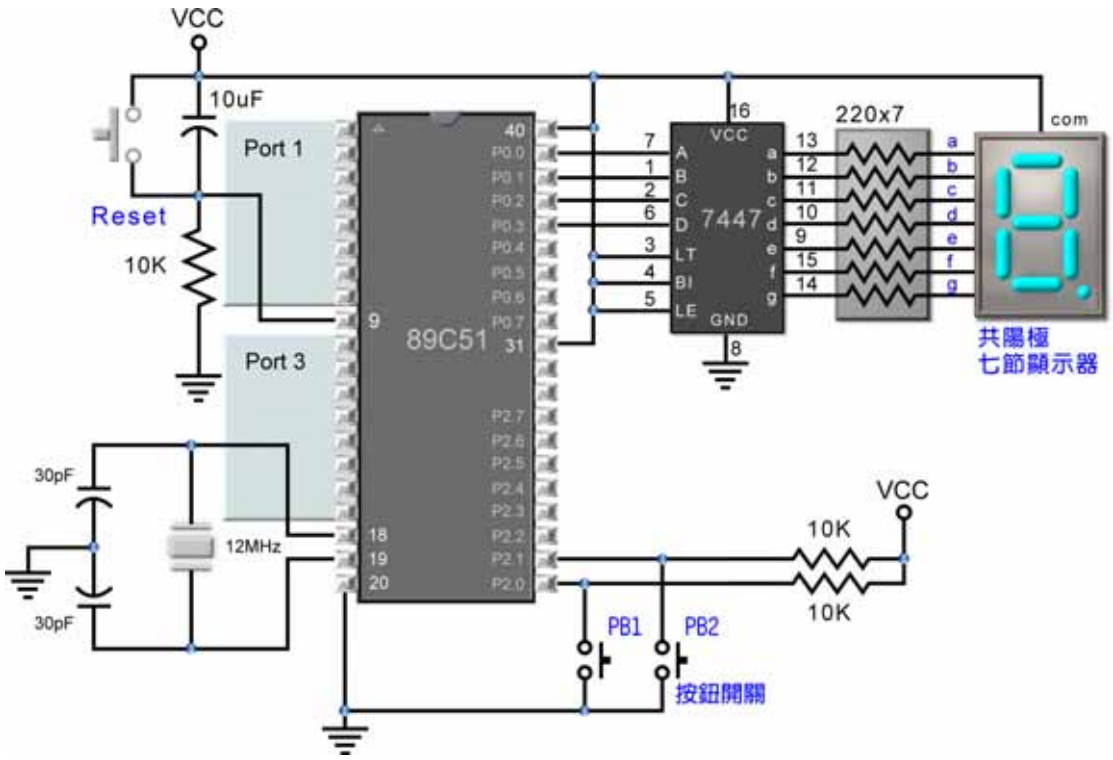
1. 依功能需求與電路結構撰寫程式，然後將該程式組譯與連結，以產生*.HEX 檔。

2. 請利用 AVSIM51 之類的模擬軟體，模擬其功能。若有非預期的狀況，則檢視原始程式，看看哪裡出問題？並將它記錄在實驗報告裡。
3. 請按圖 11 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。
4. 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？
5. 撰寫實驗報告。

思考一下

1. 在本實驗裡，有沒有「彈跳」的困擾？
2. 若將按鈕開關當成啓動馬達的 ON-OFF 開關，而 LED 換成繼電器(RELAY)，是否可做爲馬達控制？

3-4-3 計數器實例演練



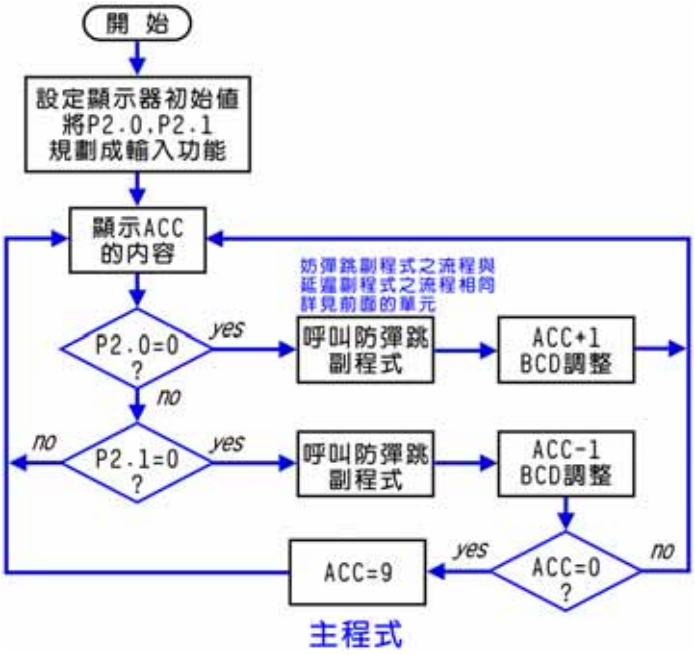
（圖12）**電路圖**

功能說明

如圖 12 所示，P0 的低四位元連接到 7447(BCD 碼對七節顯示碼之解碼與驅動器)。PB1 具有增數的功能、PB2 具有減數的功能，若程式剛開始時，七節顯示器顯示 0，按一下 PB1，則七節顯示器顯示 1、再按一下 PB1，則七節顯示器顯示 2...；若七節顯示器顯示 9，按一下 PB1，則七節顯示器顯示 0。同樣地，若七節顯示器顯示 0，按一下 PB2，則七節顯示器顯示 9、再按一下 PB2，則七節顯示器顯示 8...，以此類推。

參考程式

依功能需求與電路結構得知，只要 P0.0 到 P0.3 輸出 BCD 碼，七節顯示器即可正確地顯示 0 到 9 的數字。在程式的開始，先將七節顯示器設為 0，若讀取到 P2.0 為 0，則進行加數的操作；若讀取到 P2.1 為 0，則進行減數的操作。



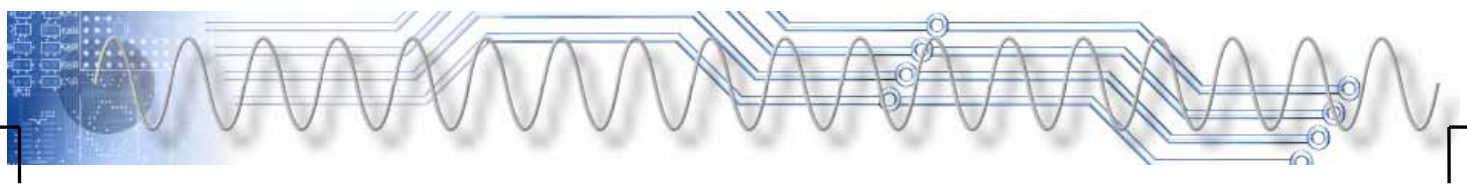
主程式

```
                ORG     0               ;程式從 0 位址開始
START:          MOV     A, #0           ;設定顯示器的初始值
                SETB    P2.0            ;將 P2.0 設定為輸入功能
                SETB    P2.1            ;將 P2.1 設定為輸入功能
LOOP:           MOV     P0, A           ;顯示 A 的內容
                JNB     P2.0, INCR      ;若 PB1 on，則跳至 INCR
                JNB     P2.1, DECR      ;若 PB1 on，則跳至 DECR
                JMP     LOOP            ;跳至 LOOP 形成一個迴圈
;=======================================================
INCR:           JNB     P2.0, INCR      ;放開按鍵？
                INC     A               ;A 加 1
                DA      A               ;將 A 的內容進行 BCD 調整
                JMP     LOOP            ;跳至 LOOP
;=======================================================
DECR:           JNB     P2.1, INCR      ;放開按鍵？
                JZ      ZERO            ;判斷 A 的內容是否為 0
                DEC     A               ;A 減 1
                JMP     LOOP            ;跳至 LOOP
ZERO:           MOV     A, #9           ;0 減 1 等於 9
                JMP     LOOP            ;跳至 LOOP
;=======================================================
                END
```

指撥開關實驗(ch3-3.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. 請利用 AVSIM51 之類的模擬軟體，模擬其功能。若有非預期的狀況，則檢視原始程式，看看哪裡出問題？並將它記錄在實驗報告裡。
3. 請按圖 12 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。
4. 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？
5. 撰寫實驗報告。

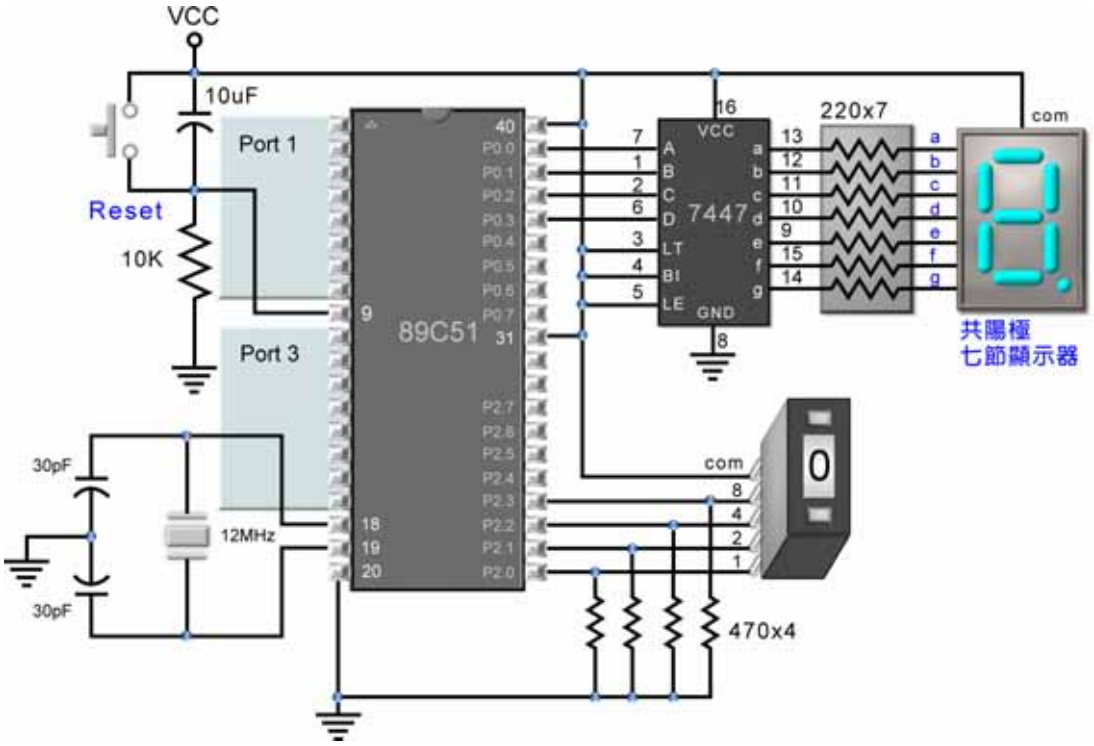


思考一下

1. 在本實驗裡，哪一些指令具有防彈跳的功能？若取消會有發生什麼事？

2. 若把本單元的電路改成 P0 連接兩個 7447 及兩個七節顯示器，其中高四位元爲十位數、低四位元爲個位數，試撰寫一個 00 到 99 的計數器。同樣地，按 PB1 增數、按 PB2 減數？

3-4-4 BCD 指撥開關實例演練



（圖13）**電路圖**

功能說明

如圖 13 所示，P0 的低四位元連接到 7447。P2 的低四位元連接到 BCD 指撥開關。讓 BCD 指撥開關上的數字，顯示在七節顯示裡。

參考程式

依功能需求與電路結構得知，只要 P0.0 到 P0.3 輸出 BCD 碼，七節顯示器即可正確地顯示 0 到 9 的數字。而由 P2 的低四位元輸入，即可得到 BCD 碼。因此，本實驗的程式只是讀取 P2，再將它送到 P0

而已，與 3-4-1 節的實驗類似。


```
                ORG     0               ;程式從 0 位址開始
START:          MOV     P2, #FFH        ;將 P2 規劃為輸入功能
LOOP:           MOV     A, P2           ;讀入指撥開關狀況
                MOV     P0, A           ;將開關狀況反應到 P0
                JMP     LOOP            ;跳至 LOOP 形成一個迴圈
                END
```

BCD 指撥開關實驗(ch3-4.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. 請利用 AVSIM51 之類的模擬軟體，模擬其功能。若有非預期的狀況，則檢視原始程式，看看哪裡出問題？並將它記錄在實驗報告裡。
3. 請按圖 13 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。
4. 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？
5. 撰寫實驗報告。

思考一下

1. 在本實驗裡，有沒有「彈跳」的困擾？
2. 若把本單元的電路改成 P0 連接兩個 7447 及兩個七節顯示器，其

中高四位元爲十位數、低四位元爲個位數；P2 連接兩個 BCD 指撥開關，其中高四位元爲十位數、低四位元爲個位數。讓 BCD 指撥開關反應在兩個七節顯示器裡，可否使用 CH3-1.ASM 或 CH3-4.ASM 程式？

3. 以圖 13 爲例，請設計一個下數計數器，以 BCD 指撥開關所設定的數字爲起始值，每 0.1 秒減 1，直到 0；然後再重新讀取 BCD 指撥開關，重新下數。

和自己賽跑！

3-5 即時練習

在本章裡探討 8051 的時序、與輸入相關的主題，以及改變程式流程的跳躍指令，可說是 8051 系統中，相當重要的一部分。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. 在 12MHz 的 8051 系統裡，一個機械週期包括多少個狀態週期？而一個狀態週期是由幾個時鐘脈波所組成？
2. 在 8051 的指令裡，大多數指令能在 1 或 2 個機械週期執行完畢，而哪些指令必須四個機械週期才能完成？
3. 當 8051 系統重置時，ACC 的內容爲何？堆疊指標指向哪個位置？而 P0 的內容爲何？
4. 若要將 8051 的輸出入埠規劃成輸入功能，應如何處理？
5. 常用的開關可分爲按鈕開關及單刀開關兩種，若要取得脈波信號，應使用哪一種開關？若要取得準位信號，應使用哪一種開關？而指撥開關屬於哪一種開關？
6. 試述如何使用 BCD 指撥開關？其輸出信號爲何？
7. 何謂「彈跳」？試繪製一個防彈跳電路？
8. 試寫一段防彈跳副程式？
9. 試說明「JB」與「JBC」指令之異同？
10. 試說明「CJNE」指令之功能與語法？