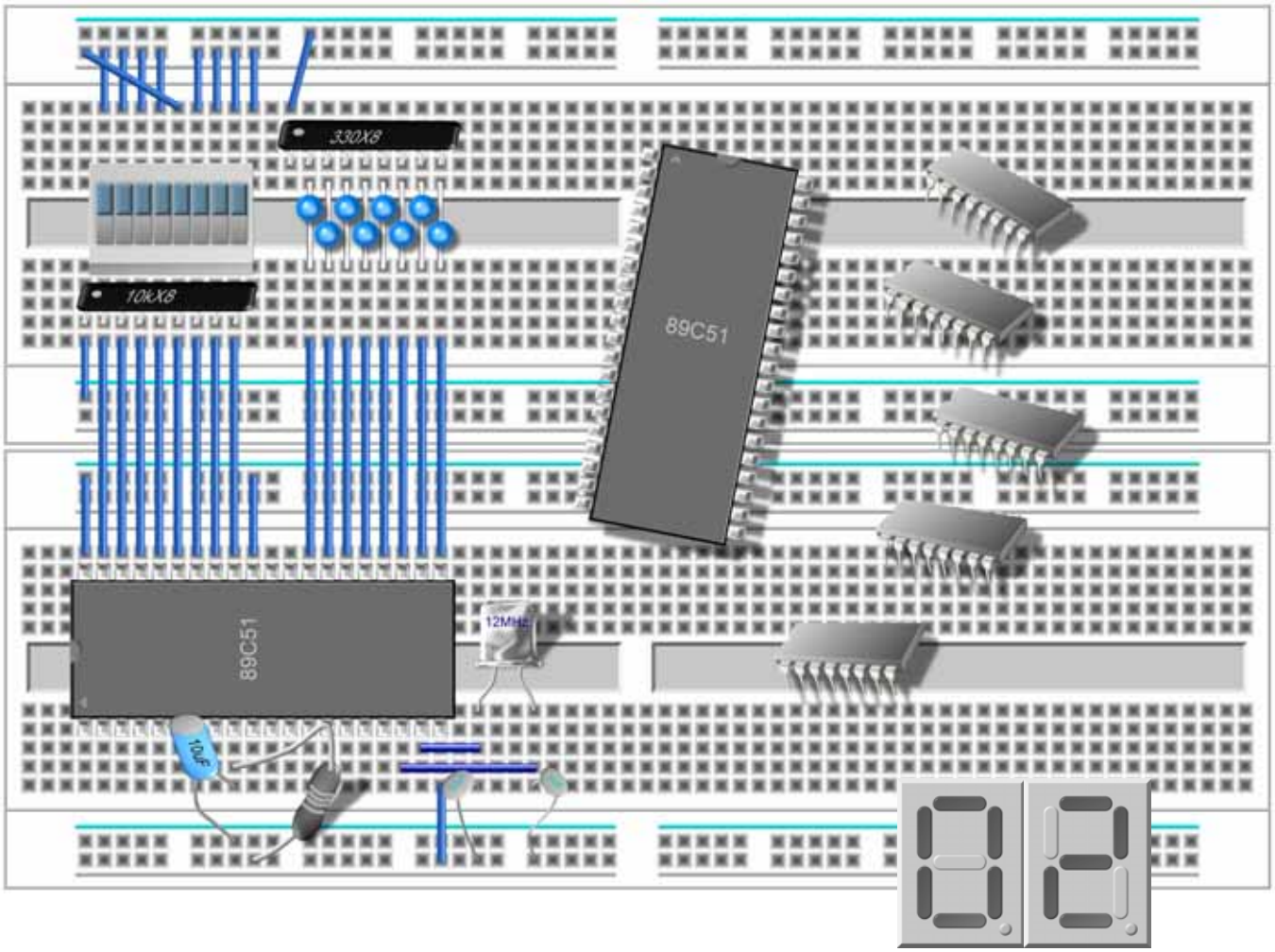


輸出埠之應用

本章內容豐富，主要包括三部分：

- 硬體部分：

  介紹了 8051 的記憶體架構、輸出入埠、輸出電路的設計等。

- 指令部分：

  介紹了指令格式、定址模式，以及資料搬移指令等。

- 程式與實作部分：

  單燈左移、霹靂燈，以及延遲副程式的計算。

2-1 認識 MCS-51 的記憶體結構

除了無 ROM 型的 8031 及 8032 外，MCS-51 之記憶體包括程式記憶體(ROM)與資料記憶體(RAM)兩部分，基本上這兩部分是獨立的個體。標準的 8x51 系列具有 4K 程式記憶體、128Bytes 資料記憶體，而標準的 8032 及 8x52 系列具有 8K、256Bytes 資料記憶體，剛好是 8x51 系列的兩倍。不管是 8x51、8031、8032 或 8x52，其外部擴充的程式記憶體或資料記憶體，最多爲 64K Bytes，如下表所示：

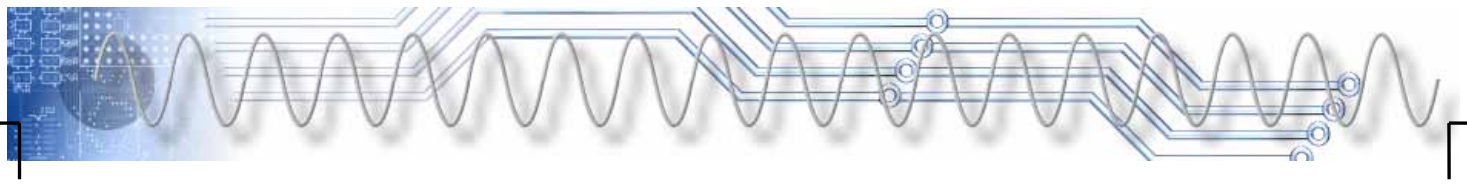
	8031		8x51		8032		8x52	
	內部	外部	內部	外部	內部	外部	內部	外部
程式記憶體	**0**	**64Kbytes**	**4 Kbytes**	**64Kbytes**	**0**	**64Kbytes**	**8 Kbytes**	**64Kbytes**
資料記憶體	**128Bytes**	**64Kbytes**	**128Bytes**	**64Kbytes**	**256Bytes**	**64Kbytes**	**256Bytes**	**64Kbytes**

近年來許多半導體廠商所推出的 MCS-51 相容單晶片，都增大其內部程式記憶體與資料記憶體，例如 Atmel 半導體公司的 TS83C51RB2，其內部 16KBytes 程式記憶體、256Bytes 資料記憶體；TS83C51RC2，其內部 32KBytes 程式記憶體、256Bytes 資料記憶體；TS83C51RD2，其內部 64KBytes 程式記憶體、768Bytes 資料記憶體。

2-1-1 程式記憶體

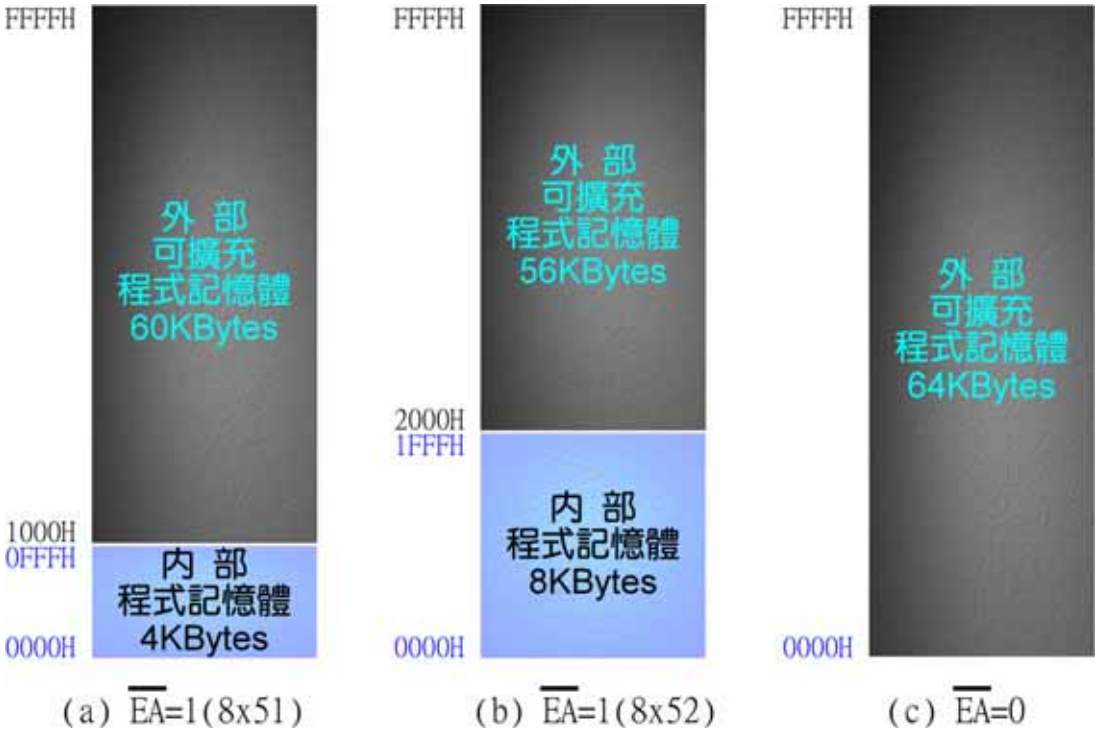
顧名思義，程式記憶體(ROM)是存放程式的位置，而 CPU 將自動從程式記憶體讀取所要執行的指令碼。而 MCS-51 可選擇使用內部程式記憶體或外部程式記憶體，如下說明：

- 若使用 8031 或 8032，由於內部沒有程式記憶體，一定要使用外部程式記憶體，所以其 $\overline{EA}$ 接腳必須接地。
- 當 $\overline{EA}$ 接腳接高準位時，CPU 將使用內部程式記憶體，若程式超過 4K Bytes(8x51)或 8K Bytes(8x52)時，則 CPU 會自動從外部程式記憶體裡，讀取超過部分的程式碼。
- 當 $\overline{EA}$ 接腳接地時，CPU 將自外部程式記憶體讀取所要執行的指令碼，而 CPU 內部的程式記憶體形同虛設。



PS:1.　4Kbytes 的程式記憶體，對於初學者而言，已是足足有餘。

2. 壞掉的 8X51/8X52，很可能是其中的程式記憶體壞掉，則可將其 $\overline{EA}$ 接腳接地，改外接程式記憶體，即可當作 8031/8032 使用。

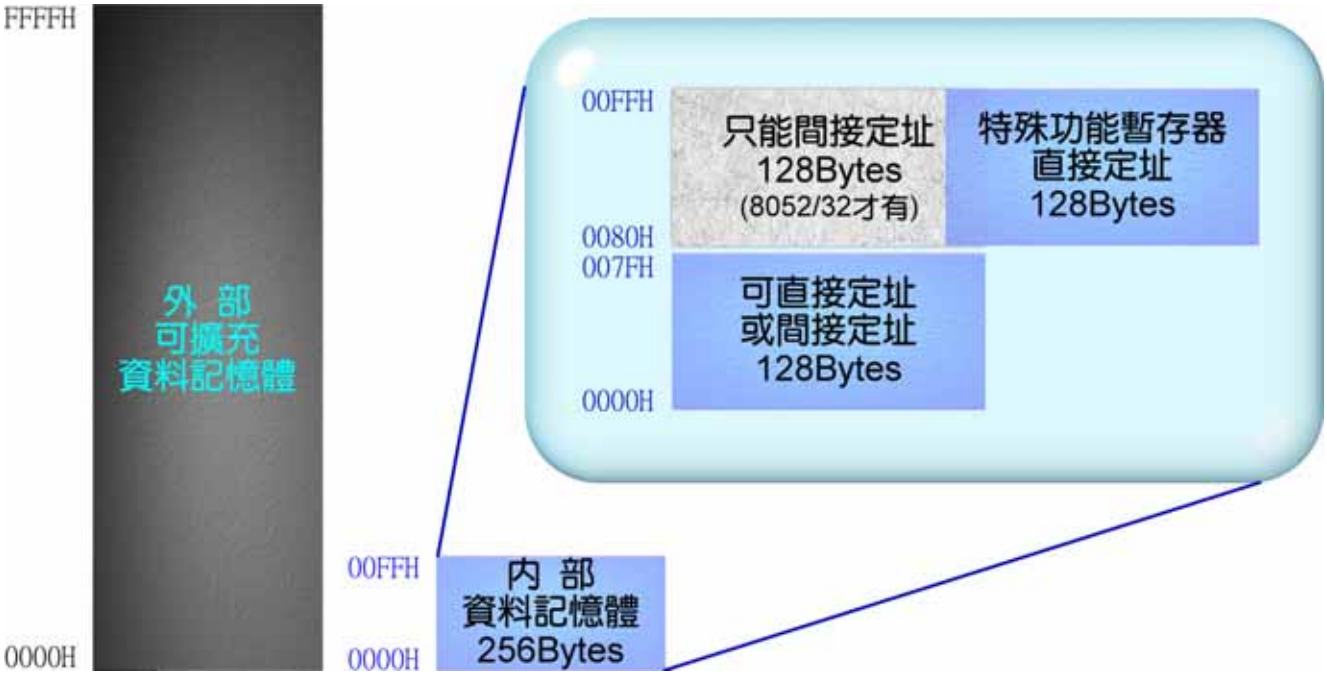


（圖1） MCS-51 之程式記憶體結構

當 CPU 重置後，程式將從程式記憶體 0000H 位置開始執行，如沒有遇到跳躍指令，則沿程式記憶體順序執行。當然，程式記憶體前面幾個位置還有一些玄機，留待第五章介紹中斷時，再詳細說明。

2-1-2 資料記憶體

MCS-51 的程式記憶體與資料記憶體是分開的獨立區塊，所以存取資料記憶體時，所使用的位址並不會與程式記憶體衝突。相對於程式記憶體，資料記憶體就*比較不單純*，如下圖所示：



（圖2） MCS-51 之資料記憶體結構

如上圖所示，在 8051 裡的資料記憶體，除內部資料記憶體外，還可擴充外部資料記憶體，這兩部分的資料記憶體可以並存。不過，存取資料記憶體時，所採用的指令並不一樣，例如存取內部資料記憶體時，可用 MOV 指令，但存取外部資料記憶體時，則須使 MOVX 指令。

另外，內部資料記憶體也有點文章！如下說明：

- 從 0000H 到 007FH 之間的 128Bytes 爲可直接定址或間接定址的記憶體，至於「直接定址」與「間接定址」，留待 2-5 節再詳細說明。在這一區裡的資料記憶體又可分成三部分，如下說明：

  - 暫存器庫區

    0000H 到 001FH 的 32 個位址爲暫存器庫(Register Bank)區，如下說明：

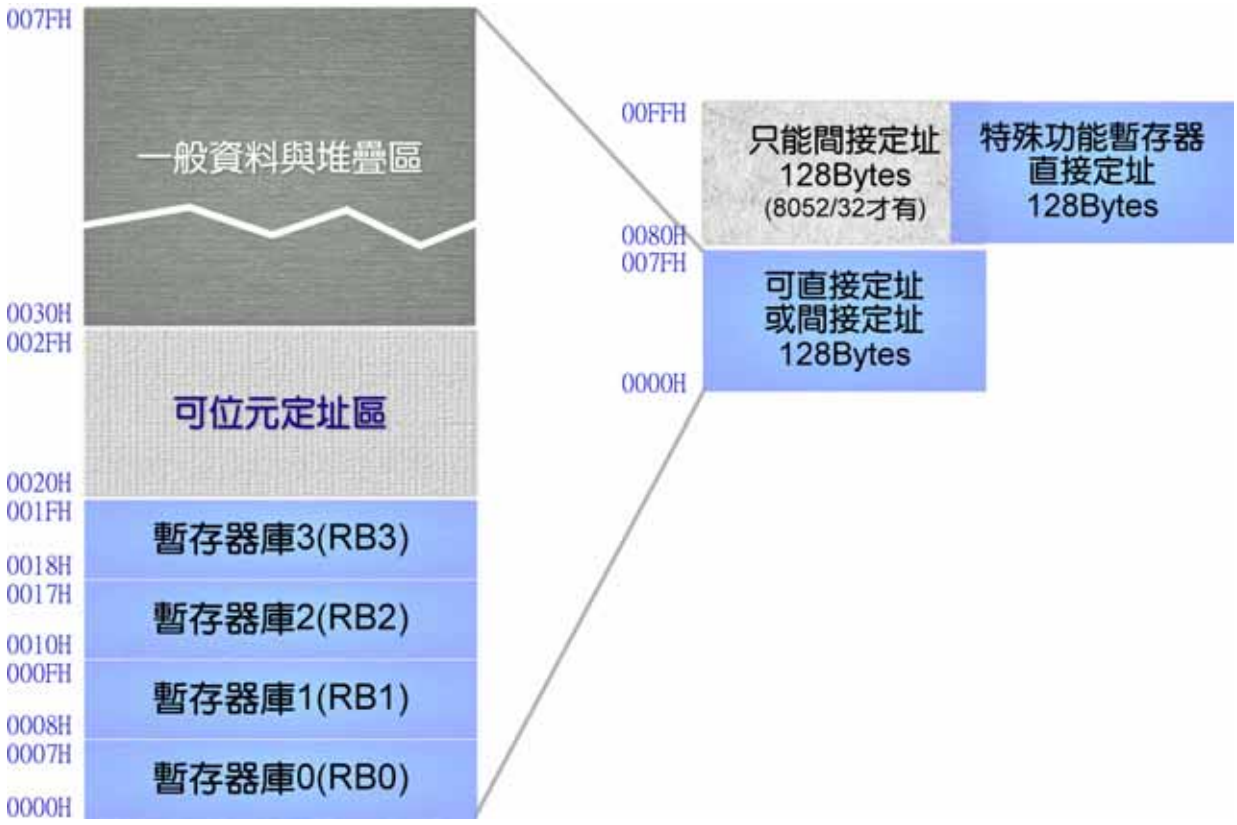
    1. 0000H 到 0007H 爲暫存器庫 0(即 **RB0**)、0008H 到 000FH 爲暫存器庫 1(即 **RB1**)、0010H 到 0017H 爲暫存器庫 2(即 **RB2**)、0018H 到 001FH 爲暫存器庫 3(即 **RB3**)。
    2. 每個暫存器庫都包含 R0、R1…R7 等 8 個暫存器，而任何一

個時間，只能使用其中一個暫存器庫。

3. 暫存器庫的切換，可以**程式狀態字組**(**P**rogram **S**tatus **W**ord，簡稱 **PSW**)中的 **RS1** 與 **RS0** 來決定，如下表所示：

RS1	RS0	暫存器庫	位　　址
0	0	RB0	0000H～0007H
0	1	RB1	0008H～000FH
1	0	RB2	0010H～0017H
1	1	RB3	0018H～001FH

4. 當 CPU 重置時，系統的堆疊指標(SP)指向 07H 位址，所以資料存入堆疊時，將從 08H 開始，也就是 RB1 裡的 R0 位址。爲避免衝突或不必要的錯誤，通常會把堆疊指標移到 30H 以後的位址。



（圖3）　**內部資料記憶體**

- 可位元定址區

  0020H 到 002FH 共 16 個位元組的記憶體區爲可位元定址區，通常存取記憶體是以一個位元組(Byte)爲單位，所謂「**可位元定址**」是指可以指定存取一個位元(bit)，在 8051 裡，只要使用布林運算指令，即可進行位元操作，例如要把 20H 記憶體位址的 bit 5 設定爲 1，則可使用下列指令：

```
SETB 20H.5
```

  另外，從 0020H 到 002FH 的 16 個位元組，總共 128 個位元(16×8)，也可以直接指定爲 0 到 127，以剛才的 20H 記憶體位址的 bit 5 而言，可指定爲 5，如下：

```
SETB 5
```

  同理，若要將 25H 記憶體位址的 bit 2 清除爲 0，則可使用下列指令：

```
CLR 25H.2
```

  或(5×8+2=42)

```
CLR 42
```

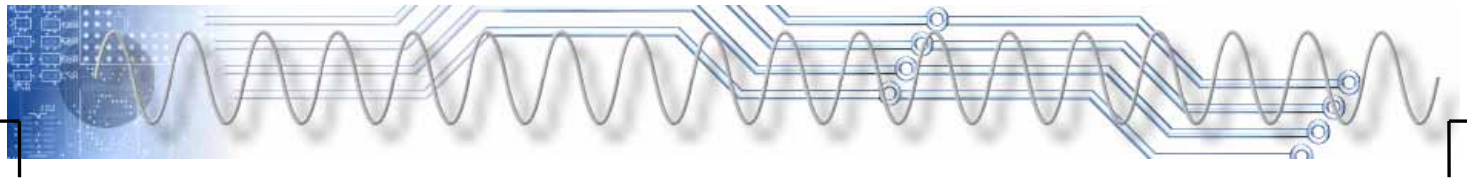
- 一般資料與堆疊區

  0030H 到 007FH 的 80 個位元組位址爲一般資料存取及堆疊區。由於 CPU 重置後，堆疊指標指向 07H 位置，爲了確保資料的安全與程式執行的正確，如果在程式之中，有使用 PUSH、POP 命令，最好能把堆疊指標改至本區，例如要將堆疊指標移至 0030H 位址，則在程式開始處即使用如下命令：

```
MOV SP, #30H
```

- 從 0080H 到 00FFH 之間的 128Bytes 爲特殊功能暫存器(**S**pecial **F**unction **R**egister 簡稱 **SFR**)，或可直接定址的記憶體，至於「**特殊功能暫存器**」，稍後再詳細說明。

- 如果是 8052/8032，則 0080H 到 00FFH 之間的 128Bytes，除了是特殊功能暫存器或可直接定址的記憶體外，另外也可以間接定址的方式，



存取與這特殊功能暫存器位置重疊，但爲獨立的記憶體。

2-1-3 特殊功能暫存器

在 MCS-51 裡，暫存器只是 CPU 裡特定位址的資料記憶體而已。而在 0080H 到 00FFH 之間的 128Bytes，正是特殊功能暫存器所在位置，什麼是「**特殊功能暫存器**」呢？首先看看這 128 個位址：

	8	9	A	B	C	D	E	F	
F8									FF
F0	**B**								F7
E8									EF
E0	**ACC**								E7
D8									DF
D0	**PSW**								D7
C8	**T2CON**		**RCAP2L**	**RCAP2H**	**TL2**	**TH2**			CF
C0									C7
B8	**IP**								BF
B0	**P3**								B7
A8	**IE**								AF
A0	**P2**								A7
98	**SCON**	**SBUF**							9F
90	**P1**								97
88	**TCON**	**TMOD**	**TL0**	**TL1**	**TH0**	**TH1**			8F
80	**P0**	**SP**	**DPL**	**DPH**				**PCON**	87
	0	1	2	3	4	5	6	7	

* 如上表所示，藍色字部分爲 8052/8032 才有的暫存器，而藍色網底的部分爲可位元定址的暫存器。

P0、P1、P2、P3

P0～P3 爲 MCS-51 的 4 個輸出入埠，其位址分別爲 80H、90H、A0H 及 B0H，稍後再詳細介紹這 4 個輸出入埠。

SP

SP 爲堆疊指標暫存器(**S**tack **P**ointer)，其位址爲 81H。堆疊是一種特殊的資料儲存方式，其資料的操作順序是先進後出(**F**irst **I**n **L**ast

Out，簡稱爲 **FILO)**，當資料以 PUSH 命令送入堆疊時，SP 自動加 1；若以 POP 命令從堆疊取出資料時，SP 自動減 1。

- DPL、DPH

DPL 與 DPH 各爲 8 位元的**資料指標暫存器(D**ata **P**ointer)，其位址分別爲 8AH、8BH，若以 DPL 爲低 8 位元、DPH 爲高 8 位元，即可組成一個 16 位元的資料指標暫存器，簡稱爲 **DPTR**，如此將可定址到 64Kbytes 的資料位址。

- PCON

PCON 爲電源控制暫存器，其位址爲 87H，其功能是設定 CPU 的電源模式，待後續關於電源部分，再行說明。

- TCON

TCON 爲計時/計數器控制暫存器，其位址爲 88H，其功能是設定計時/計數器的啓動、記錄計時/計數溢位，以及外部中斷的型式等，待第六章關於計時/計數器部分，再行說明。

- TMOD

TMOD 爲計時/計數模式控制暫存器，其位址爲 89H，其功能是設定計時/計數的模式，留待第六章關於計時計數器部分再詳細說明。

- TL0、TL1、TH0、TH1

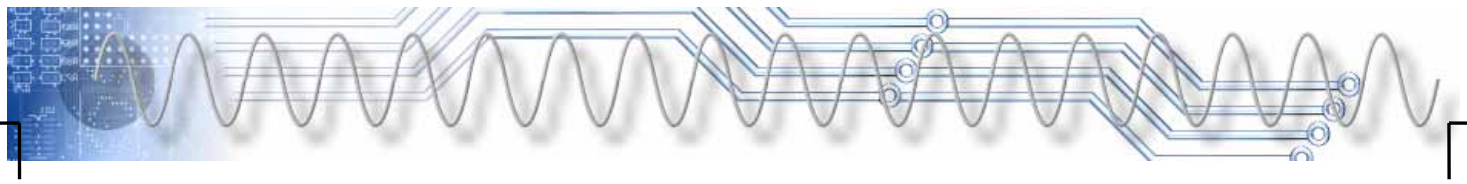
TL0、TH0 爲第一組計時/計數器(Timer0)的計量暫存器，其位址爲 8AH、8CH，將 TH0 與 TL0 組合即可進行 16 位元的計時/計數。TL1、TH1 爲第二組計時/計數器(Timer1)的計量暫存器，其位址爲 8BH、8DH，將 TH1 與 TL1 組合即可進行 16 位元的計時/計數，留待第六章關於計時計數器部分再詳細說明。

- SCON

SCON 爲串列埠控制暫存器，其位址爲 98H，其功能是設定串列埠工作模式與旗標，留待第七章關於串列埠部分再詳細說明。

- SBUF

SBUF 爲串列埠緩衝器，其位址爲 99H，這是由使用同一個位址的兩個暫存器所構成，其中一個暫存器做爲傳出資料用的緩衝器，另



一個暫存器做爲接收資料用的緩衝器。至於如何分辨同一個位址的兩個暫存器，則視指令而定，若是資料傳出的指令，則自動定位到傳出資料用的緩衝器；若是接收資料的指令，則自動定位到接收資料用的緩衝器，留待第七章關於串列埠部分再詳細說明。

IE

IE 爲中斷致能暫存器(**I**nterrupt **E**nable Register)，其位址爲 A8H，其功能是啓用中斷功能，待第五章關於中斷部分，再行說明。

IP

IP 爲中斷優先等級暫存器(**I**nterrupt **P**riority Register)，其功能是設定中斷的優先等級，待第五章關於中斷部分，再行說明。

T2CON

T2CON 爲 Timer 2 的計時/計數器控制暫存器，其位址爲 C8H，其功能是設定 Timer 2 的啓動、記錄計時/計數溢位，以及外部中斷的型式等，而 Timer 2 只有在 8052/8032 才有。

RCAP2L、RCAP2H

RCAP2L、RCAP2H 爲捕捉暫存器(Capture Register)，其位址爲 CAH、CBH。當 Timer 2 在捕捉模式時，若 T2EX(P1.1)接腳上的輸入信號由高態轉爲低態，TL2 與 TH2 的內容將被載入 RCAP2L 與 RCAP2H 裡，就像是把 Timer 2 的內容捉進 RCAP 暫存器一樣，而 Timer 2 只有在 8052/8032 才有。

TL2、TH2

TL2、TH2 爲第三組計時/計數器(Timer2)的計量暫存器，其位址爲 CCH、CDH，將 TH2 與 TL2 組合即可進行 16 位元的計時/計數，而 Timer 2 只有在 8052/8032 才有。

PSW

PSW 爲 CPU 的程式狀態字組暫存器(**P**rogram **S**tatus **W**ord)，其位址爲 D0H，其內容如下說明：

	7	6	5	4	3	2	1	0
PSW	CY	AC	F0	RS1	RS0	OV		P

- **PSW.7**：本位元爲進位位元(**CY**)，進行加法(減法)運算時，若最左邊位元(MSB，即 bit 7)產生進位(借位)時，則本位元將自動設定爲 1，即 CY=1；否則 CY=0。
- **PSW.6**：本位元爲輔助進位位元(**AC**)，進行加法(減法)運算時，若 bit 3 產生進位(借位)時，則本位元將自動設定爲 1，即 AC=1；否則 AC=0。
- **PSW.5**：本位元爲使用者旗標(**F0**)，可由使用者自行設定的位元。
- **PSW.4** 與 **PSW.3**：這兩個位元爲暫存器庫選擇位元(**RS1**、**RS0**)，其功能如下表所示：

RS1	RS0	暫存器庫
0	0	RB0
0	1	RB1
1	0	RB2
1	1	RB3

- **PSW.2**：本位元爲溢位旗標(**OV**)，當進行算術運算時，若發生溢位，則 OV=1；否則 OV=0。
- **PSW.1**：本位元爲保留位元，沒有提供服務。
- **PSW.0**：本位元爲同位位元(**P**)，8051 採偶同位，若 ACC 裡有奇數個 1，則 P=1；若 ACC 裡有偶數個 1，則 P=0。

ACC

ACC 累積器(Accumulator)又稱爲 A 暫存器，其位址爲 E0H，這個暫存器提供 CPU 主要運作的位置，可說是最常用的暫存器。

B

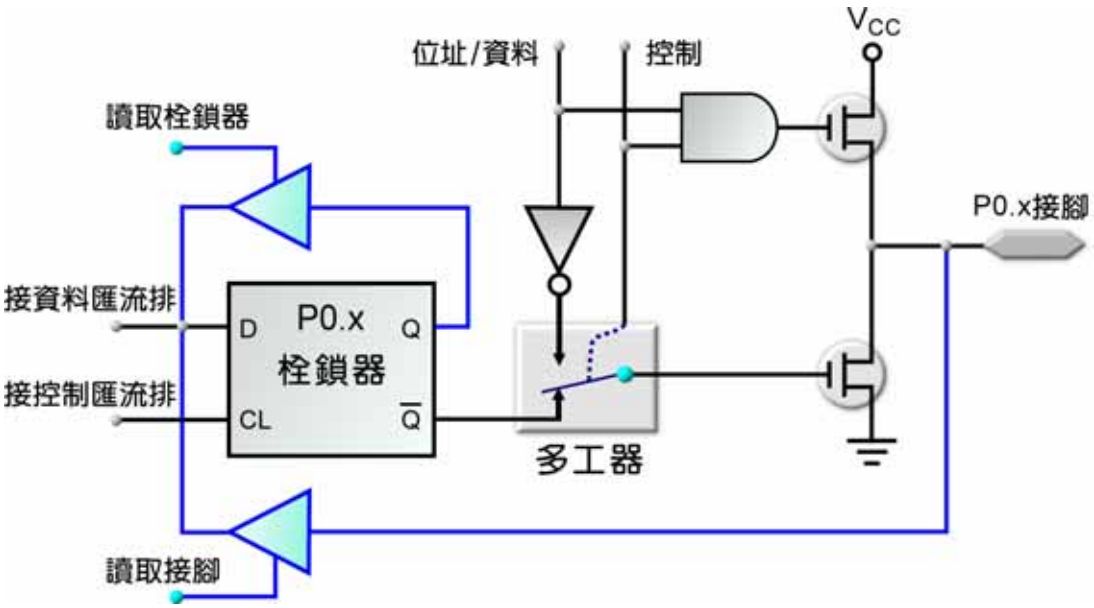
B 暫存器的位址爲 F0H，主要功能是搭配 A 暫存器進行乘法或除法運算，進行乘法運算時，乘數放在 B 暫存器，而運算的結果，高八位元放在 B 暫存器；進行除法運算時，除數放在 B 暫存器，而運算的結果，餘數放在 B 暫存器。若不進行乘/除法運算，B 暫存器也可當成一般暫存器使用。

2-2 8051 的輸出入埠

MCS-51 迷人的地方之一，就在其四個輸出入埠！這四個看似相似的輸出入埠，還是有點差異，如下說明：

PORT 0

PORT 0 爲 8 位元、可位元定址的輸出入埠，以針腳式包裝的 8051 爲例，P0.0 爲 39 腳、P0.1 爲 38 腳、...P0.7 爲 32 腳，如下圖所示爲其內部結構：



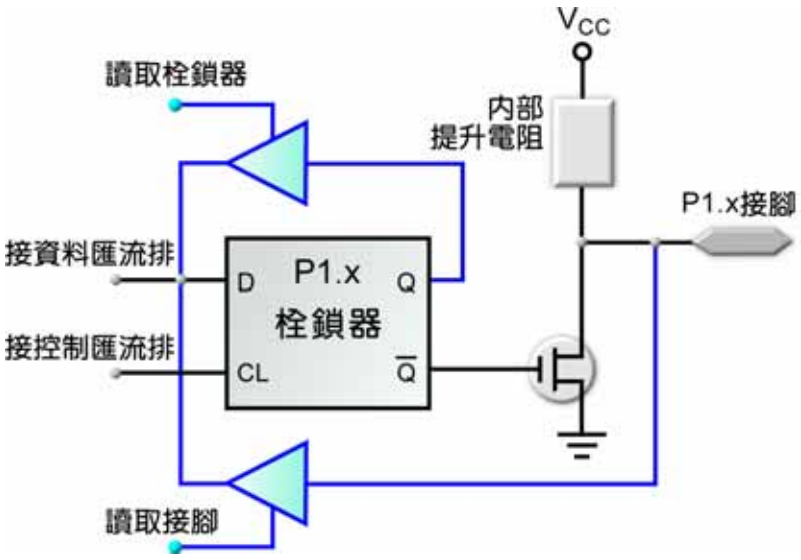
（圖4）　PORT 0 內部電路結構(1 個位元)

PORT 0 的特色說明如下：

- PORT 0 的 8 位元皆爲開洩極式輸出(open drain，簡稱 OD)，千萬不要誤解爲圖騰式輸出，而每隻接腳可驅動 8 個 LS 型 TTL 負載。
- PORT 0 內部無提升電阻，執行輸出功能時，外部必須接提升電阻(10K 歐姆即可)。
- 若要執行輸入功能，必須先輸出高準位(1)，方能讀取該埠所連接的外部資料。
- 若系統連接外部記憶體，則 PORT 0 可做爲位址匯流排(A0～A7)及資料匯流排(D0～D7)之多工接腳。

PORT 1

PORT 1 爲 8 位元、可位元定址的輸出入埠，以針腳式包裝的 8051 爲例，P1.0 爲 1 腳、P1.1 爲 2 腳、...P1.7 爲 8 腳，如下圖所示爲其內部結構：



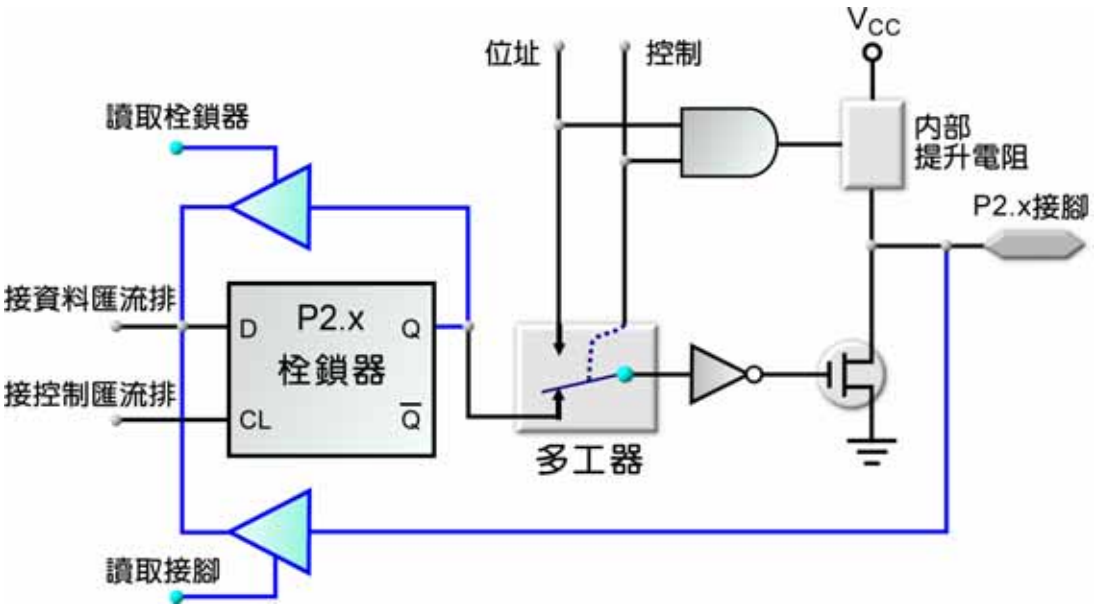
（圖5） PORT 1 內部電路結構(1 個位元)

PORT 1 的特色說明如下：

- PORT 1 內部具備 30KΩ提升電阻，執行輸出功能時，不須連接外部提升電阻。
- PORT 1 的 8 位元皆爲開洩極式輸出(OD)，每隻接腳可驅動 4 個 LS 型 TTL 負載。
- 若要執行輸入功能，必須先輸出高準位(1)，方能讀取該埠所連接的外部資料。
- 若是 8052/8032，則 P1.0 兼具有 Timer 2 的外部脈波輸入功能(即 T2)、P1.1 兼具有 Timer 2 的捕捉/重新載入之觸發輸入功能(即 T2EX)。

PORT 2

PORT 2 爲 8 位元、可位元定址的輸出入埠，以針腳式包裝的 8051 爲例，P2.0 爲 21 腳、P2.1 爲 22 腳、...P2.7 爲 28 腳，如下圖所示爲其內部結構：



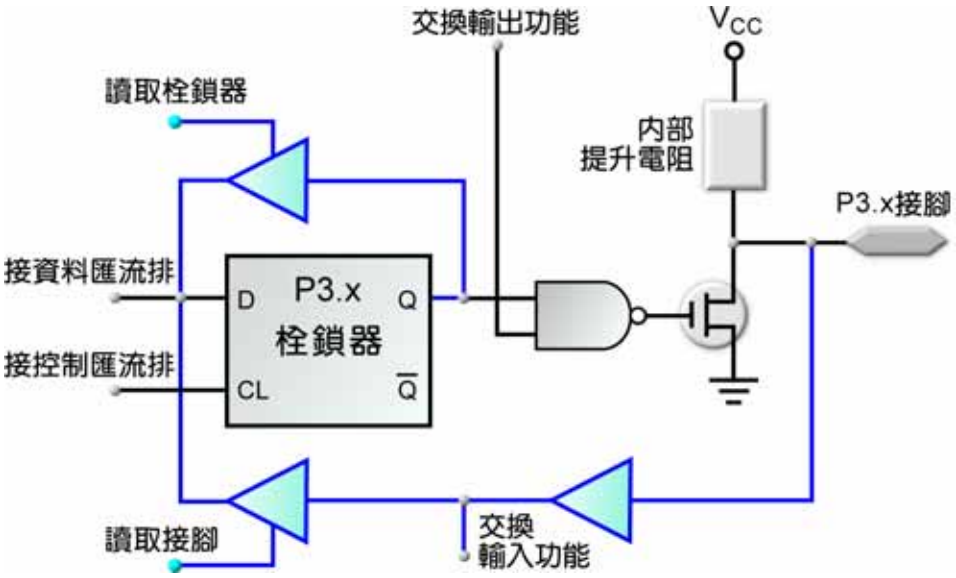
（圖6）　PORT 2 內部電路結構(1 個位元)

PORT 2 的特色說明如下：

- PORT 2 內部具備 30KΩ提升電阻，執行輸出功能時，不須連接外部提升電阻。
- PORT 2 的 8 位元皆爲開洩極式輸出(OD)，每隻接腳可驅動 4 個 LS 型 TTL 負載。
- 若要執行輸入功能，必須先輸出高準位(1)，方能讀取該埠所連接的外部資料。
- 若系統連接外部記憶體，而外部記憶體的位址線超過 8 條時，則 PORT 0 可做爲位址匯流排(A8～A15)接腳。

PORT 3

PORT 3 爲 8 位元、可位元定址的輸出入埠，以針腳式包裝的 8051 爲例，P3.0 爲 10 腳、P3.1 爲 11 腳、...P3.7 爲 17 腳，如下圖所示爲其內部結構：



（圖7） PORT 3 內部電路結構(1 個位元)

PORT 3 的特色說明如下：

- PORT 3 內部具備 30KΩ提升電阻，執行輸出功能時，不須連接外部提升電阻。
- PORT 3 的 8 位元皆爲開洩極式輸出(OD)，每隻接腳可驅動 4 個 LS 型 TTL 負載。
- 若要執行輸入功能，必須先輸出高準位(1)，方能讀取該埠所連接的外部資料。
- PORT 3 的 8 隻接腳各有其它功能，如下表所示：

PORT 3	其它功能	說　　明
P3.0	**RXD**	串列埠的接收接腳
P3.1	**TXD**	串列埠的傳送接腳
P3.2	**INT0**	INT0 中斷輸入
P3.3	**INT1**	INT1 中斷輸入
P3.4	**T0**	Timer 0 輸入
P3.5	**T1**	Timer 1 輸入
P3.6	**WR**	寫入外部記憶體控制接腳
P3.7	**RD**	讀取外部記憶體控制接腳

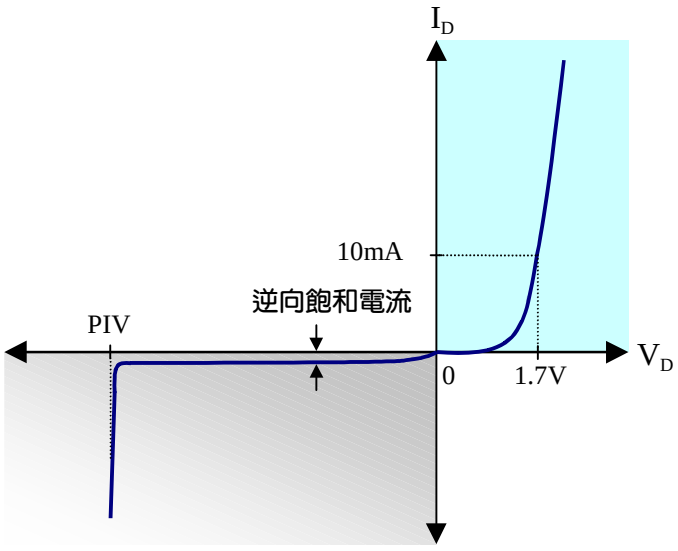
2-3 輸出電路設計

8051 的輸出埠可直接連接數位電路，也可用來驅動 LED、繼電器或喇叭等負載，以下就來探討如何與這些負載過招。

2-3-1 驅動 LED

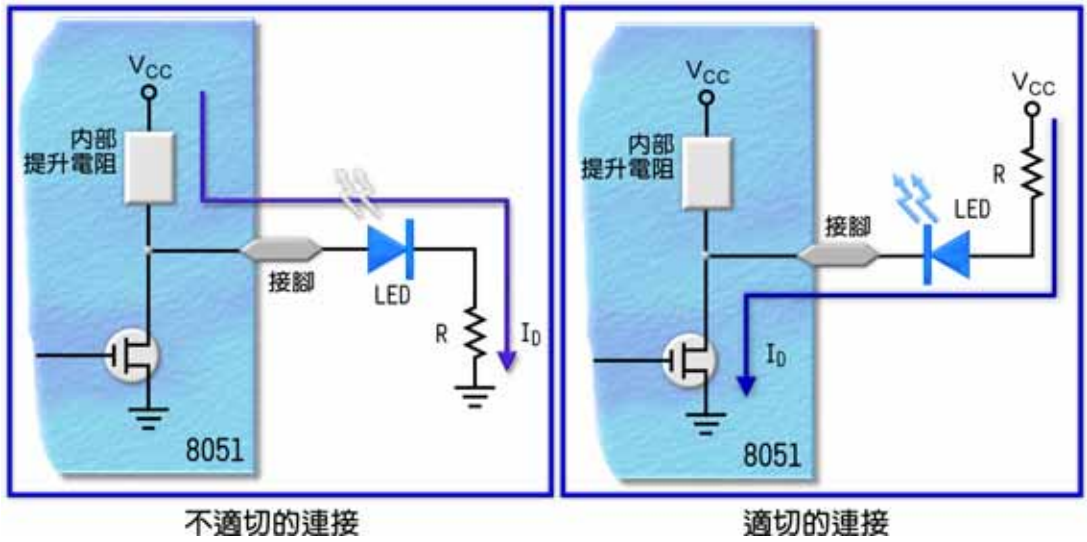
LED 爲發光二極體的簡稱，其體積小、低耗電量，常被用爲微電腦與數位電路的輸出裝置，以指示信號狀態。近年來 LED 的技術大爲精進，除了紅色、綠色、黃色外，還出現了藍色與白色，而高亮度的 LED 更取代傳統燈泡，成爲交通號誌(紅綠燈)的發光元件；就連汽車的尾燈，也開始流行使用 LED 車燈。

基本上，LED 具有二極體的特色，逆向偏壓時，LED 將不發光；順向偏壓時，LED 將發光，而 LED 兩端約有 1.7V 左右的壓降(比二極體大)，如下圖所示爲其特性曲線：



（圖8） LED 特性曲線

隨著通過 LED 順向電流的增加，LED 將更亮，而 LED 的壽命也將縮短，而以 10m 到 20m 安培爲宜。8051 的輸出入埠都是開洩極式的輸出，其中的 P1、P2 與 P3 內建 30KΩ提升電阻，因此想從 P1、P2 或 P3 流出 10m 到 20m 安培，恐怕有困難。如果從外面流入 8051 的 Port，那電流就可以大一點，如下圖所示：



（圖9） 輸出 LED 之連接

如上右圖所示，當輸出低態時，輸出端的FET將導通，輸出端電壓接近 0V，若LED順向時，兩端電壓V_D爲 1.7V，則限流電阻*R*兩端將存在3.3V(即 5-1.7)。如果希望流過LED的電流I_D限制爲 10m安培，則此限流電阻R爲：

$$R = \frac{5-1.7}{10m} = 330\,\Omega$$

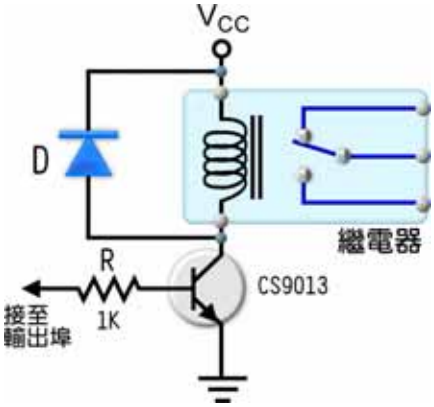
若想要LED亮一點，可使I_D提高至 15 m安培，則限流電阻*R*改爲：

$$R = \frac{5-1.7}{15m} = 220\,\Omega$$

對於 TTL 準位的數位電路或微電腦電路，其中 LED 所串接的限流電阻，大多在 200 到 330Ω之間，限流電阻越小，LED 越亮。

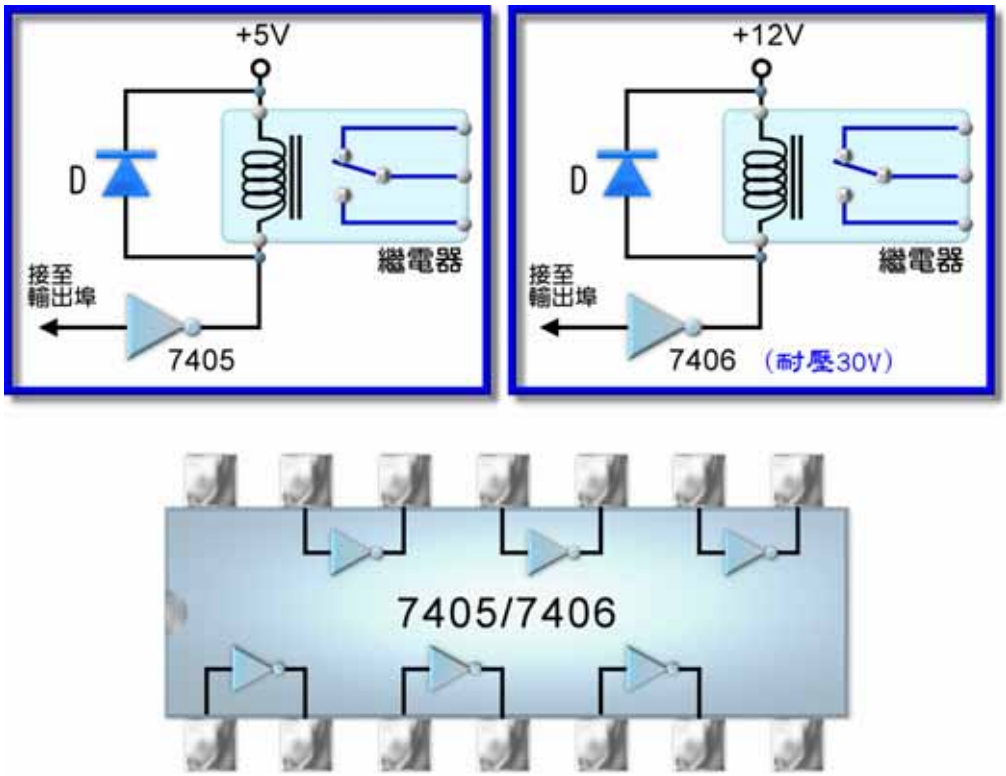
2-3-2 驅動繼電器

如果要利用 8051 來控制不同電壓或較大電流的負載時，則可透過繼電器(RELAY)來轉達控制的意圖。若要驅動繼電器，光靠 8051 輸出埠的電流恐怕不夠，況且驅動繼電器線圈這種電感性負載，還要有些保護才行！我們可使得電晶體來控制繼電器，以一個 12V 繼電器爲例，如下圖所示：



（圖10）使用電晶體驅動繼電器

在此這個電晶體是當成開關來用，也就是 8051 輸出高態時，電晶體工作於飽和狀態、8051 輸出低態時，電晶體工作於截止狀態。其中的二極體D提供繼電器線圈電流的放電路徑，以保護電晶體。由於線圈屬於電感性負載，當電晶體截止時，i_c=0，而原本線圈上的電流i_L不可能瞬間爲 0，所以二極體D就提供一個i_L的放電路徑，使線圈不會產生高的應電勢，就不會破壞電晶體了。

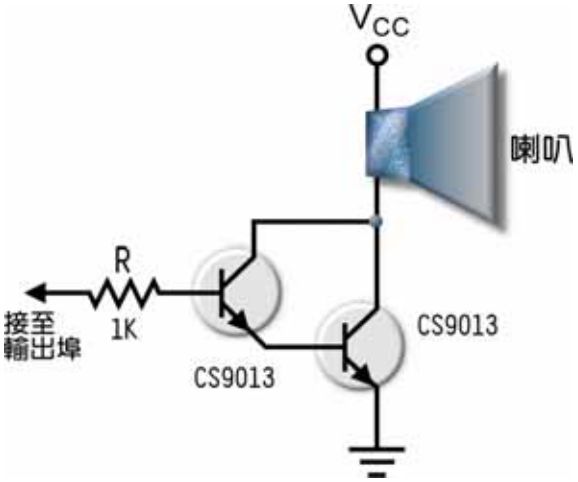


（圖11）使用 7405/7406 驅動繼電器

如果要同時有多個輸出埠要驅動繼電器，則可使用開集極式(OC)輸出的反相閘，如 7405(驅動 5V 的繼電器)或 7406(驅動較高電壓的繼電器，最高 30V)，如圖 11 所示。

2-3-3 驅動喇叭

基本上，喇叭(speaker)是一種電感性的負載，而 8051 驅動喇叭的信號爲各種頻率的脈波。因此，最簡單的喇叭驅動方式就是利用達靈頓電晶體，或以兩個常用的小電晶體(如 CS9013)連接成達靈頓架構，如下圖所示：



（圖12）**使用電晶體驅動喇叭**

其中的電阻器 R 爲限流電阻，在此利用電晶體的高電流增益(若單個電晶體的β超過 35，則整體電流增益將超過 1000 倍)，以達到電路快速飽和的目的。

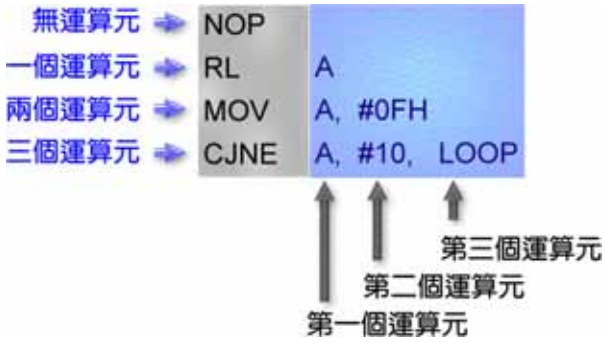
2-4 指令格式

8051 原始程式的指令格式包括四個欄位，最左邊欄位爲標籤欄位(label)，第二個欄位是指令助憶碼欄位(mnemonic)，第三個欄位是運算元欄位(operand)，第四個欄位是註解欄位(comment)，如下所示：

標籤欄位	助憶碼欄位	運算元欄位	註解欄位
	ORG	0	
START:	MOV	A, #0FH	;00001111B
L1:	MOV	P1, A	;由PORT 1輸出
	CALL	DELAY	;呼叫延遲副程式
	CPL	A	;對A取補數
	JMP	L1	;跳至L1
DELAY:	MOV	R7, #200	;延遲副程式
D1:	MOV	R6, #100	
	DJNZ	R6, $	
	DJNZ	R7, D1	
	RET		
	END		

標籤欄位的功能是放置標籤，以爲副程式的起始標記或跳躍指令的參考位置。**若不放置標籤，則標籤欄位必須空著**。而第二個欄位就是放置指令助憶碼，如 MOV、ADDC 等。

在第三個欄位裡放置運算元，隨著指令的不同，就有不同個數的運算元，某些指令沒有運算元(如 NOP)、某些指令只有一個運算元(如 INC)、某些指令有兩個運算元(如 MOV、ANL)、某些指令有三個運算元(如 CJNE)，若運算元爲兩個以上，則在兩個運算元之間以逗點分隔。



註解是給人看的，並不會被組譯，我們可在第四個欄位裡放置說明文字。當然，如果在 Windwos 下使用**記事本**編輯原始程式的話，則可在此欄位中使用中文。在註解之前必須放置分號(;)，分號之右的文字將被組譯程式給忽略掉，除了可在第四個欄位中放置說明文字外，也可在其它位置放置說明文字，只要在其左邊放置分號即可。

不管是使用 PE2 還是**記事本**，只要按 Tab 鍵即可跳到下個欄位。

2-5 定址模式

所謂「**定址**」，簡單地講，就是找到運算元位址的方法。8051 提供 5 種定址模式，如下說明：

直接定址

直接定址(direct addressing)是直接在運算元欄位裡，指定運算元所在位置的位址，包括特殊功能暫存器(如 P1、P2、PSW 等)，例如：

```
ADD     A, 40H
```

將記憶體 40H 位址的內容，加到 ACC 裡。又如：

```
MOV     30H, A
```

將 ACC 的內容，複製到記憶體 30H 位址裡。

間接定址

間接定址(indirect addressing)是利用索引暫存器(R1 或R0，標示爲 R_i)、基底暫存器(SP或DPTR)，間接指示運算元所在位置的位址，而在索引暫存器或基底暫存器之左要加上「@」符號，例如：

```
INC     @R0
```

以 R0 的內容爲位址，將記憶體該位址的內容加 1。又如：

```
MOVX  @DPTR, A
```

以 DPTR 的內容爲位址，將 ACC 的內容，複製到外部記憶體該位址裡。

暫存器定址

暫存器定址(Register addressing)是以暫存器(即 R0 到 R7)的內容爲運算元，例如：

```
ANL     A, R5
```

將 R5 的內容與 ACC 的內容進行 AND 運算，其結果放入 ACC 裡。又如：

```
DJNZ    R7, LOOP
```

將 R7 的內容減 1，若其結果不等於 0，則跳到 LOOP 處；若其結果等於 0，則執行下一列指令。

立即定址

立即定址(immediate addressing)是直接在運算元欄位裡放置運算元，而在運算元左邊必須放置一個「#」符號，例如：

```
ORL     A, #0FH
```

將 0FH 與 ACC 的內容進行 OR 運算，其結果放入 ACC 裡。又如：

```
CJNE    A, #10, LOOP
```

若 ACC 的內容不等於 10，則跳到 LOOP 處；若 ACC 的內容等於 10，則執行下一列指令。

索引定址

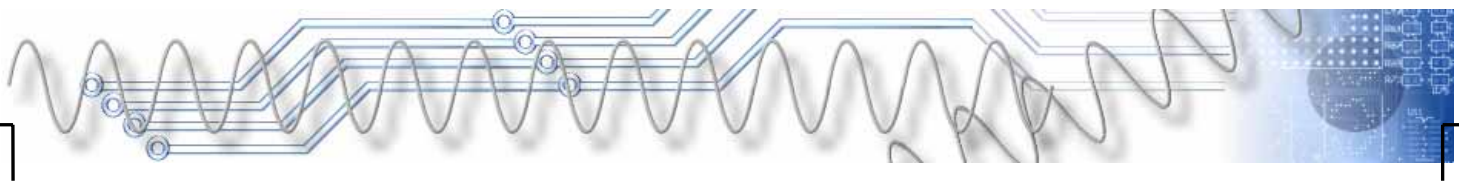
索引定址(index addressing)與間接定址有點類似，不過，索引定址同時使用索引暫存器與基底暫存器，而改以 ACC 爲索引暫存器，以儲存差距(offset)、PC 或 DPTR 爲基底暫存器。將基底與差距，將基底與差距相加後，才是運算元所在位置的位址，例如：

```
MOVC  A, @A+PC
```

先將 PC 的內容與 ACC 的內容相加，以其結果爲位址，再將該位址的內容，複製到 ACC 裡。又如：

```
JMP     @A+DPTR
```

先將 DPTR 的內容與 ACC 的內容相加，以結果爲位址，再跳到該位址。



2-6 資料轉移指令

資料轉移指令的功能是將來源運算元的資料，複製到目的運算元裡；或將指定的運算元內容交換。資料轉移指令屬於 8051 指令裡的最大族群，包括 28 個指令，在此將它們分爲 5 大類來介紹：

資料複製指令

資料複製指令的功能是將來源運算元的資料複製到目的運算元，如下圖所示：

其中的來源運算元可爲記憶體(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*、ACC 的內容等；目的運算元可爲記憶體(RAM)位址 *direct*、暫存器 R*n*、以索引暫存器 R*i* 內容爲地址(@R*i*)、ACC、資料指標暫存器 DPTR 等。

- MOV A, *direct*
  - **說明**：將記憶體(RAM)位址(*direct*)的內容複製到 ACC 裡，即 (*direct*) ⟶ ACC。
  - **組譯後大小**：2 bytes　　**執行時間**：12 個時鐘脈波
  - **範例**：

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

- MOV A, R*n*
  - **說明**：將暫存器 R*n* 的內容複製到 ACC 裡，即 R*n* ⟶ ACC。
  - **組譯後大小**：1 byte　　**執行時間**：12 個時鐘脈波

▶範例：

```
指令：MOV    A, R1
若執行前：ACC=12H、R1=27H
執行後：ACC=27H、R1=27H
```

- MOV　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複製到 ACC 裡，即(R*i*) ⟶ ACC。
  - ▶組譯後大小：1 byte　　執行時間：12 個時鐘脈波
  - ▶範例：

```
指令：MOV    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=A3H、R0=21H、記憶體(21H)位址的內容爲 A3H
```

- MOV　A, #*data*
  - ▶說明：將立即值 *data* 複製到 ACC 裡，即 *data* ⟶ ACC。
  - ▶組譯後大小：2 bytes　　執行時間：12 個時鐘脈波
  - ▶範例：

```
指令：MOV    A, #125
若執行前：ACC=A2H
執行後：ACC=125
```

- MOV　R*n*, A
  - ▶說明：將 ACC 的內容複製到暫存器 R*n* 裡，即 ACC ⟶ R*n*。
  - ▶組譯後大小：1 byte　　執行時間：12 個時鐘脈波
  - ▶範例：

```
指令：MOV    R5, A
若執行前：ACC=ABH、R5=30H
執行後：ACC=ABH、R5=ABH
```

- MOV Rn, #*data*
  - **說明**：將立即值 *data* 複製到暫存器 Rn 裡，即 *data* → Rn。
  - **組譯後大小**：2 bytes **執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：MOV  R7, #200
若執行前：R7=00H
執行後：R7=200
```

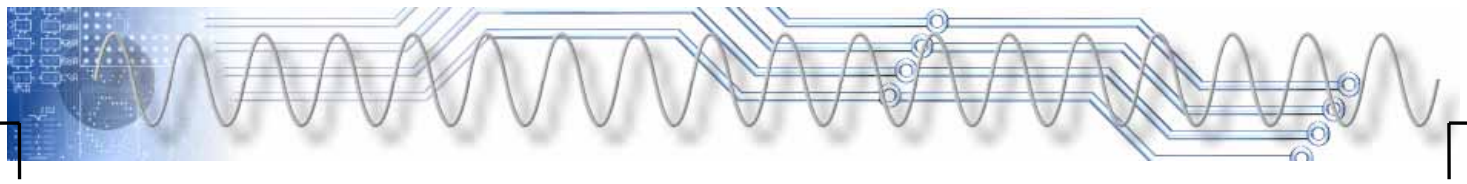
- MOV Rn, *direct*
  - **說明**：將記憶體位址(*direct*)的內容複製到暫存器 Rn 裡，即 (*direct*) → Rn。
  - **組譯後大小**：2 bytes **執行時間**：24 個時鐘脈波
  - **範例**：

```
指令：MOV  R2, 20H
若執行前：R2=01H、記憶體(20H)位址的內容爲 30H
執行後：R2=30H
```

- MOV *direct*, A
  - **說明**：將 ACC 的內容複製到記憶體位址(*direct*)，即 ACC → (*direct*)。
  - **組譯後大小**：2 bytes **執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：MOV  20H, A
若執行前：ACC=AAH、記憶體(20H)位址的內容爲 30H
執行後：ACC=AAH、記憶體(20H)位址的內容爲 AAH
```

- MOV *direct*, Rn
  - **說明**：將暫存器 Rn 裡的內容複製到記憶體位址(*direct*)，即 Rn → (*direct*)。
  - **組譯後大小**：2 bytes **執行時間**：24 個時鐘脈波
  - **範例**：



指令：**MOV　R0, 21H**

若執行前：**R0=32H、記憶體(21H)位址的內容爲 ACH**

執行後：**R0=ACH、記憶體(21H)位址的內容爲 ACH**

- MOV　*direct1*, *direct2*
  - **說明**：將記憶體位址(*direct2*)的內容複製到記憶體位址(*direct1*)，即(*direct2*) → (*direct1*)。
  - **組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令：**MOV　20H, 30H**

若執行前：**記憶體(20H)位址的內容爲 00H、(30H)位址的內容爲 12H**

執行後：**記憶體(20H)位址的內容爲 12H、(30H)位址的內容爲 12H**

- MOV　*direct*, @R*i*
  - **說明**：以索引暫存器 R*i* 的內容爲地址，將記憶體該位址的內容複製到記憶體位址(*direct*)，即(@R*i*) → (*direct*)。
  - **組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令：**MOV　20H, @R0**

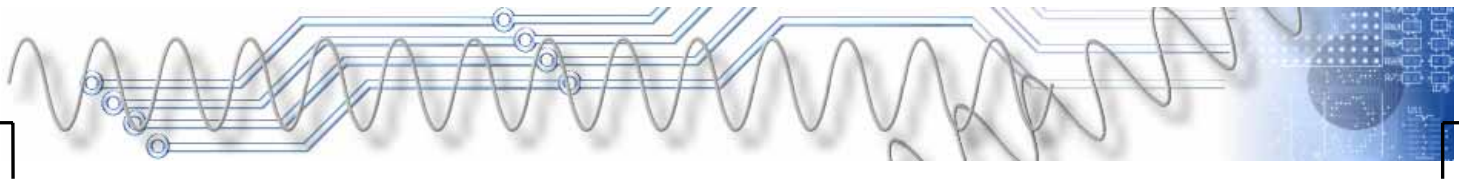
若執行前：**記憶體(20H)位址的內容爲 00H、R0=30H、(30H)位址的內容爲 34H**

執行後：**記憶體(20H)位址的內容爲 34H、R0=30H、(30H)位址的內容爲 34H**

- MOV　*direct*, #*data*
  - **說明**：將 8 位元立即值 *data* 複製到記憶體位址(*direct*)，即 *data* → (*direct*)。
  - **組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令：**MOV　30H, #FFH**

若執行前：**記憶體(30H)位址的內容爲 00H**



執行後：記憶體(30H)位址的內容爲 FFH

- MOV @R*i*, A
  - **說明**：將 ACC 的內容複製到以索引暫存器 R*i* 內容爲地址的記憶體裡，即 ACC ⟶ (R*i*)。
  - **組譯後大小**：1 byte **執行時間**：12 個時鐘脈波
  - **範例**：

指令：**MOV @R0, A**

若執行前：**ACC=12H、R0=22H、記憶體(22H)位址的內容爲 56H**

執行後：**ACC=12H、R0=22H、記憶體(22H)位址的內容爲 12H**

- MOV @R*i*, *direct*
  - **說明**：將(*direct*)記憶體位址的內容複製到以索引暫存器 R*i* 內容爲地址的記憶體裡，即(*direct*) ⟶ (R*i*)。
  - **組譯後大小**：2 bytes **執行時間**：24 個時鐘脈波
  - **範例**：

指令：**MOV @R1, 20H**

若執行前：**R1=21H、記憶體(21H)位址的內容爲 56H、(20H)位址的內容爲 ABH**

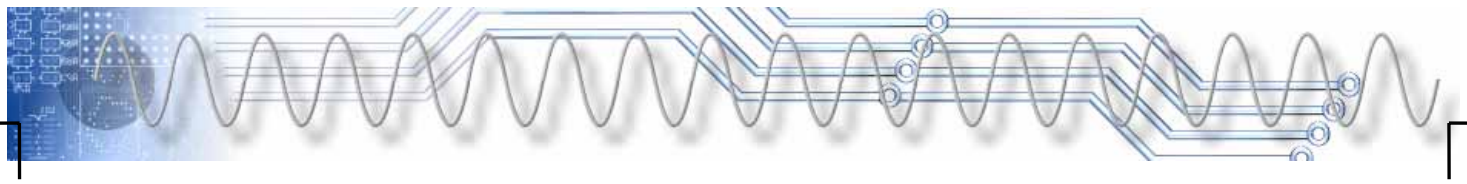
執行後：**R1=21H、記憶體(21H)位址的內容爲 ABH、(20H)位址的內容爲 ABH**

- MOV @R*i*, #*data*
  - **說明**：將 8 位元立即值 *data* 複製到以索引暫存器 R*i* 內容爲地址的記憶體裡，即 *data* ⟶ (R*i*)。
  - **組譯後大小**：2 bytes **執行時間**：12 個時鐘脈波
  - **範例**：

指令：**MOV @R1, #20**

若執行前：**R1=33H、記憶體(33H)位址的內容爲 1CH**

執行後：**R1=33H、記憶體(33H)位址的內容爲 20**



- MOV　DPTR, #*data16*
  - **說明**：將 16 位元立即值 *data16* 複製到資料指標暫存器 DPTR 裡，即 *data16* ⟶ DPTR。
  - **組譯後大小**：3 bytes　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令：MOV　DPTR, #1234H
若執行前：DPTR=0000H
執行後：DPTR=1234H

查表法指令

查表指令的功能是先將基底暫存器的內容加上 ACC 的內容，再以其和爲地址，將該記憶體地址的內容複製到 ACC，如下圖所示：

其中的基底暫存器爲 16 位元的暫存器，如 DPTR、PC，因此可以定址到 64K 的範圍。

- MOVC　A, @A+DPTR
  - **說明**：先將 ACC 的內容與 DPTR 的內容相加裡，再以其結果爲地址(16 位元)，將該記憶體位址的內容複製到 ACC 裡，如此將可定址到 64K 個位置，即(ACC+DPTR) ⟶ ACC。
  - **組譯後大小**：1 byte　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令：MOVC　A, @A+DPTR
若執行前：ACC=02H、DPTR=0010H、記憶體(0012H)位址的內容爲 3DH
執行後：ACC=3DH、DPTR=0010H、記憶體(0012H)位址的內容爲 3DH

- MOVC A, @A+PC
  - **說明**：先將 ACC 的內容與程式計數器 PC 的內容相加裡，再以其結果爲地址(16 位元)，將該記憶體位址的內容複製到 ACC 裡，如此將可定址到 64K 個位置，即(ACC+PC) ⟶ ACC。
  - **組譯後大小**：1 byte **執行時間**：24 個時鐘脈波
  - **範例**：

指令：MOVC A, @A+PC
若執行前：ACC=0、PC=0020H、記憶體(0020H)位址的內容爲 52H
執行後：ACC=52H、PC=0020H、記憶體(0020H)位址的內容爲 52H

外部資料存取指令

外部資料存取指令的功能是存取外部擴充記憶體(RAM)的資料，如下圖所示：

其中的來源運算元可爲以索引暫存器 R*i* 內容爲地址(@R*i*)的資料、以資料暫存器 DPTR 內容爲地址(@DPTR)的資料、ACC 的內容等；目的運算元可爲 ACC、以索引暫存器 R*i* 內容爲地址的位置(@R*i*)、以資料暫存器 DPTR 內容爲地址的位置(@DPTR)等。

- MOVX A, @R*i*
  - **說明**：以索引暫存器 R*i* 的內容爲地址(8 位元)，將外部記憶體該位址的內容複製到 ACC 裡，即(R*i*) ⟶ ACC。
  - **組譯後大小**：1 byte **執行時間**：24 個時鐘脈波
  - **範例**：

指令：MOVX A, @R1
若執行前：ACC=0、R1=60H、外部記憶體(60H)位址的內容爲 79H
執行後：ACC=79H、R1=60H、外部記憶體(60H)位址的內容爲 79H

- MOVX　A, @DPTR
  - **說明**：以資料指標暫存器 DPTR 的內容爲地址(16 位元)，將外部記憶體該位址的內容複製到 ACC 裡，如此將可定址到 64K 個位置，即(DPTR) ⟶ ACC。
  - **組譯後大小**：1 byte　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令：MOVX　A, @DPTR
若執行前：ACC=0、DPTR=1234H、外部記憶體(1234H)位址的內容爲 54H
執行後：ACC=54H、DPTR=1234H、外部記憶體(1234H)位址的內容爲 54H

- MOVX　@R*i*, A
  - **說明**：以索引暫存器 R*i* 的內容爲地址(8 位元)，將 ACC 的內容複製到外部記憶體該位址裡，即 ACC ⟶ (R*i*)。
  - **組譯後大小**：1 byte　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令：MOVX　@R0, A
若執行前：ACC=0、R0=20H、外部記憶體(20H)位址的內容爲 79H
執行後：ACC=0、R0=20H、外部記憶體(20H)位址的內容爲 0

- MOVX　@DPTR, A
  - **說明**：以資料指標暫存器 DPTR 的內容爲地址(16 位元)，將 ACC 的內容複製到外部記憶體該位址裡，如此將可定址到 64K 個位置，即 ACC ⟶ (DPTR)。
  - **組譯後大小**：1 byte　　**執行時間**：24 個時鐘脈波
  - **範例**：

指令：MOVX　@DPTR, A
若執行前：ACC=EFH、DPTR=0012H、外部記憶體(0012H)位址的內容爲 AAH
執行後：ACC=EFH、DPTR=0012H、外部記憶體(0012H)位址的內容爲 EFH

堆疊存取指令

堆疊存取指令的功能是按堆疊指標存取堆疊資料，堆疊記憶體具有先進後出(FILO)的特性，適用於堆疊操作的運算元爲記憶體(RAM)位址 *direct* 的資料，當然，所有特殊功能暫存器都屬於 RAM 的一部分，所以，可以堆疊存取指令來操作。

PUSH *direct*

- **說明**：將堆疊指標暫存器 SP 的內容加 1，然後將(*direct*)位址的內容複製到(SP)位址的記憶體裡，即(*direct*) → (SP+1)。
- **組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波
- **範例**：

指令：PUSH　A
若執行前：ACC=FFH、SP=60H、記憶體(61H)位址的內容爲 00H
執行後：ACC=FFH、SP=61H、記憶體(61H)位址的內容爲 FFH

POP *direct*

- **說明**：以堆疊指標暫存器 SP 爲地址，將記憶體中該位址的內容複製到(*direct*)位址，然後將 SP 減 1，即(SP) → (*direct*)。
- **組譯後大小**：2 bytes　　**執行時間**：24 個時鐘脈波
- **範例**：

指令：POP　A
若執行前：ACC=00H、SP=61H、記憶體(61H)位址的內容爲 12H
執行後：ACC=12H、SP=60H、記憶體(61H)位址的內容爲 12H

資料互換指令

資料互換指令的功能是將來源運算元的資料與目的運算元的資料，如下圖所示：

其中的來源運算元可爲記憶體(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*)的資料等；目的運算元則只能爲 ACC。

- XCH　A, *direct*
  - **說明**：將 ACC 的內容與(*direct*)位址的內容互換，即 A ⟷ (*direct*)。
  - **組譯後大小**：2 bytes　　**執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：XCH   A, P0
若執行前：ACC=12H、P2=34H
執行後：ACC=34H、P2=12H
```

- XCH　A, R*n*
  - **說明**：將 ACC 的內容，與 R*n* 的內容互換，即 A ⟷ R*n*。
  - **組譯後大小**：1 byte　　**執行時間**：12 個時鐘脈波
  - **範例**：

```
指令：XCH   A, R2
若執行前：ACC=05H、R2=123
執行後：ACC=123、R2=05H
```

- XCH　A, @R*i*
  - **說明**：將 ACC 的內容，與記憶體(R*i*)位址的內容互換，即 ⟷ (R*i*)。
  - **組譯後大小**：1 byte　　**執行時間**：12 個時鐘脈波

▶**範例**：

指令：XCH A, @R0
若執行前：ACC=05H、R0=20H、記憶體(20H)位址的內容爲 32H
執行後：ACC=32H、R0=20H、記憶體(20H)位址的內容爲 05H

XCHD A, @R*i*

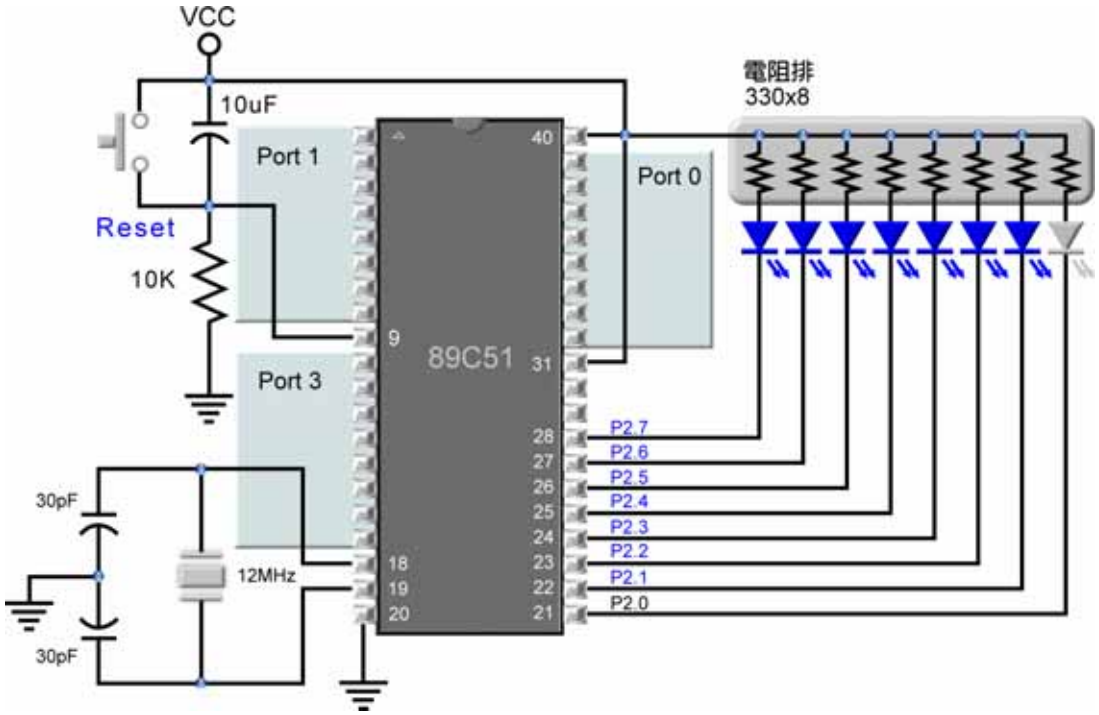
▶**說明**：將ACC的低四位元，與記憶體(R*i*)位址內的低四位元互換，即$A_{0\text{-}3}$ ⟷ $(Ri)_{0\text{-}3}$。

▶**組譯後大小**：1 byte　　**執行時間**：12 個時鐘脈波

▶**範例**：

指令：XCHD A, @R0
若執行前：ACC=05H、R0=20H、記憶體(20H)位址的內容爲 32H
執行後：ACC=02H、R0=20H、記憶體(20H)位址的內容爲 35H

2-7 實例演練

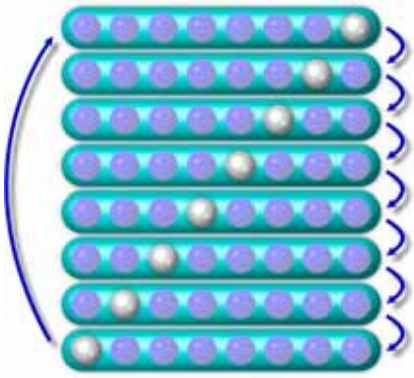


（圖13）**實驗電路**

如上圖所示爲本單元的實驗電路，在此將進行單燈左移、雙燈左移，以及霹靂燈的實驗。

2-7-1 單燈左移

在此的「單燈左移」是從 Port 2 輸出的 8 個 LED 中，從 P2.0 所連接的 LED 開始亮，任何時間只有一個 LED 亮，如下圖所示：

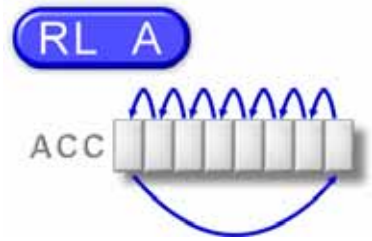


（圖14）**單燈左移動作示意圖**

設計要點

1. 如圖 13 所示，若要 LED 亮，則 P2 輸出 0；因此，剛開始時輸出 11111110，即可使最右邊的 LED 亮，其餘不亮。

2. 若要從左邊第一個 LED 亮，轉變成第二個 LED 亮，可用左轉指令，即



3. 在此要介紹 8051 的「計次迴圈」的格式，如下所示：

```
LOOP:  MOV   Rn, 次數
       指令區塊
       DJNZ  Rn, LOOP
```

其中 R*n* 爲紀錄次數的暫存器，例如要重複執行指令區塊 5 次，則

```
        MOV   R7, #5
LOOP:   [ 指令區塊 ]
        DJNZ  R7, LOOP
```

4. 每個 LED 亮的時間約 0.1 秒，因此需要一個延遲副程式，在此將應用前述的計次迴圈，如下所示：

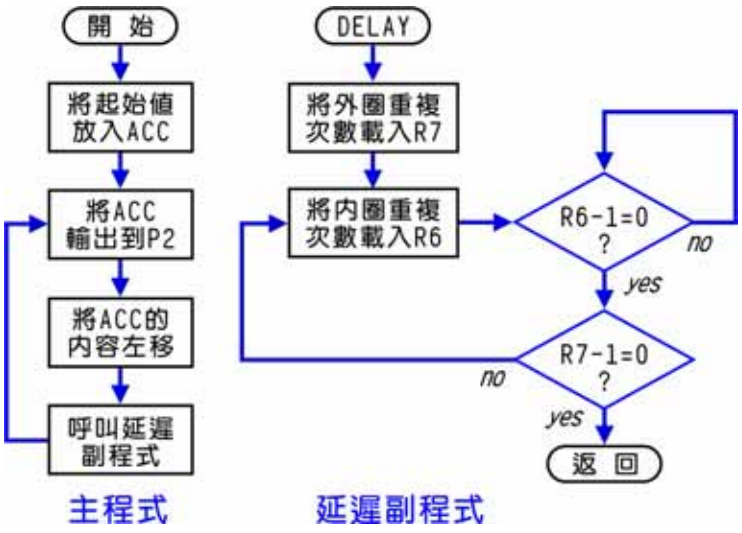
```
DELAY:
        MOV  R7, #200     ;12個週期
D1:     MOV  R6, #250     ;12個週期
        DJNZ R6, $        ;24個週期
        DJNZ R7, D1       ;24個週期
        RET               ;24個週期
執行R6次
執行R7次
```

其中的「**DJNZ　R6, $**」指令是對本身那一列指令執行 R6 次。對於 12MHz 時鐘脈波的 8051 系統而言，12 個週期(1 個機械週期)剛好是 1 微秒，整個副程式的延遲時間 T 爲：

$$T=1+(1+2\times R6+2)\times R7+2$$
$$=3+3\times R7+2\times R6\times R7$$
$$\cong 2\times R6\times R7 \text{ 微秒}$$

R6=250、R7=200，則延遲時間 T≅0.1 秒。

程式設計



```
                ORG     0               ;程式從 0 位址開始
START:          MOV     A, #FEH         ;讓 ACC 的內容為 11111110
LOOP:           MOV     P2, A           ;從 Port 2 輸出 ACC 的內容
                RL      A               ;將 ACC 的內容左移
                CALL    DELAY           ;呼叫延遲副程式
                JMP     LOOP            ;跳到 LOOP 處執行
;=====================================================
DELAY:                                  ;延遲副程式(0.1 秒)
                MOV     R7, #200        ;R7 暫存器載入 200 次數
D1:             MOV     R6, #250        ;R6 暫存器載入 250 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
                END                     ;結束程式
```

Ch2-1.asm

軟體模擬

經過組譯與連結後，即可以 AVSIM51 進行軟體模擬，但要注意下列事項：

1. 軟體模擬並不能正確反應延遲副程式的延遲時間，若以 R7=200、R6=250 模擬，恐怕要等非常久；最好能改短一點，例如 R7=20、R6=25 即可。以本延遲副程式爲例，通常 R6 的值盡量大一點，R7 的值盡量小一點，時間誤差比較小。

2. 若以視窗模式執行 AVSIM51 程式，可能模擬起來會有點怪異，最好是按 Alt + Enter 鍵，讓 AVSIM51 程式以全螢幕執行。模擬完畢後，再按 Alt + Enter 鍵即可恢復爲視窗模式。

實體模擬

連接好圖 13 的電路，再連接好實體模擬器，然後載入經過組譯與連結後的*.HEX 檔，以進行實體模擬。*嘿嘿*，對於延遲副程式而言，實體模擬可是真正的時間，記得把 R7 與 R6 恢復正常。

舉一反三

1.認識單燈左移的程式之後，請修改該程式，讓它變成單燈右移的功能。

2.試設計一個雙燈左移的程式？

3.在本單元裡採用一個 0.1 秒的延遲副程式，若要得到 0.01 秒的延遲，要如何修改？1 秒的延遲副程式又如何設計？

2-7-2 霹靂燈

「霹靂燈」是只有一個 LED 亮，由右邊移至左邊、再移回右邊...，如此不斷地來回移動。在此將以圖 13 的電路，設計一個霹靂燈程式。

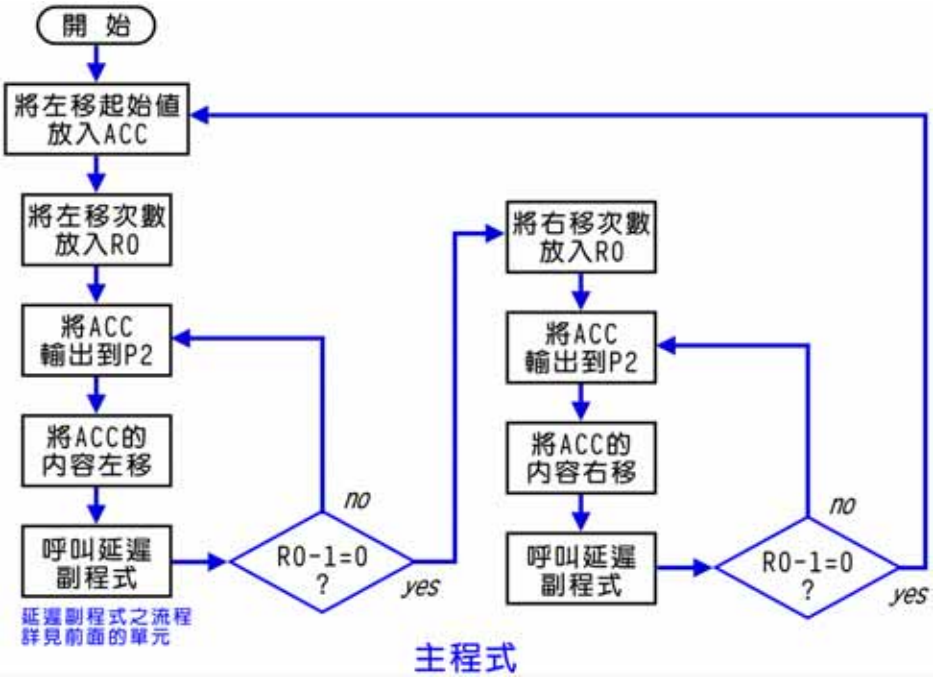
設計要點

1. 基本上，霹靂燈程式是單燈左移與單燈右移的組合。

2. 若一開始是單燈左移，則其初始值爲「11111110」，即最右邊 LED 亮，其它不亮。

3. 左移 7 次後，將變成最左邊 LED 亮，其它不亮。緊接著是單燈右移，右移 7 次後，又恢復爲最右邊 LED 亮，其它不亮。如此週而復使，即爲霹靂燈。

程式設計




```
                ORG     0               ;程式從 0 位址開始
;單燈左移=============================================
START:          MOV     A, #FEH         ;讓 ACC 的內容為 11111110
LOOP:           MOV     R0, #7          ;以 R0 為左移的計次計數器
LOOPL:          MOV     P2, A           ;從 Port 2 輸出 ACC 的內容
                RL      A               ;將 ACC 的內容左移
                CALL    DELAY           ;呼叫延遲副程式
                DJNZ    R0, LOOPL       ;LOOPL 迴圈執行 R0 次
;單燈右移=============================================
                MOV     R0, #7          ;以 R0 為右移的計次計數器
LOOPR:          MOV     P2, A           ;從 Port 2 輸出 ACC 的內容
                RR      A               ;將 ACC 的內容右移
                CALL    DELAY           ;呼叫延遲副程式
                DJNZ    R0, LOOPR       ;LOOPR 迴圈執行 R0 次
                JMP     LOOP            ;從頭開始
;延遲副程式===========================================
DELAY:                                  ;延遲副程式(0.1 秒)
                MOV     R7, #200        ;R7 暫存器載入 200 次數
D1:             MOV     R6, #250        ;R6 暫存器載入 250 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
                END                     ;結束程式
```

Ch2-2.asm

軟體模擬

經過組譯與連結後，即可以 AVSIM51 進行軟體模擬，注意事項與 2-7-1 節相同。

實體模擬

連接好圖 13 的電路，再連接實體模擬器(ICE)，即可載入經過組譯與連結後的*.HEX 檔，以進行實體模擬。嘿嘿，對於延遲副程式而言，實體模擬可是真正的時間，記得把 R7 與 R6 恢復正常。

舉一反三

1.認識單燈的霹靂燈程式之後，請修改該程式，讓它變成雙燈的霹靂燈功能。

和自己賽跑！

2-8 即時練習

在本章裡，介紹了 8051 的記憶體架構、輸出入埠、輸出電路的設計等硬體部分；在軟體方面，則介紹了指令格式、定址模式，以及資料搬移指令，這些都是學習 8051 不可或缺的相關知識。在此請試著回答下列問題，以驗證學習成效：

1. 8051 內部的程式記憶體與資料記憶體各多少？而外部擴充的程式記憶體與資料記憶體最多各多少？
2. 在 8051 電路裡，若要使用外部程式記憶體，應如何連接？而存取外部資料記憶體，必須使用哪個指令？
3. 試述 8051 內部有多少個暫存器庫？如何切換？
4. 試簡述 PSW 爲何？並說明其中各位元的功能？
5. 試說明何謂 SFR？其位址在哪裡？
6. 試說明何謂 DPTR？其功能爲何？
7. 試說明「位元定址」？哪裡的記憶體可位元定址？哪些特殊功能暫存器可位元定址？
8. 試述 P0 與 P2 接腳的其它功能？
9. 試述 P3 接腳的其它功能？
10. 試問 7405 與 7406 之異同？
11. 在電晶體驅動繼電器的電路裡，繼電器的線圈兩端並接一個逆向二極體，其功能爲何？
12. 試述在 8051 的原始程式裡，包括哪些欄位？
13. 試述 8051 提供哪幾種定址模式？
14. 試說明 8051 的間接定址與索引定址？
15. 試說明「MOVC　A, @A+DPTR」指令的動作？
16. 試說明「XCH　A, @Ri」指令與「XCHD　A, @Ri」指令的差別？
17. 試說明「PUSH　A」指令與「POP　A」指令的動作？
18. 當我們使用 AVSIM51 進行軟體模擬時，要注意哪些事項？

19. 若要進行實體模擬時，以你手邊的實體模擬器爲例，應如何連接？

20. 試撰寫一個 1 秒鐘的延遲副程式？