# 單晶片C語言程式應用研習

承辦機構: 秉華科技有限公司
講　　師: 丁榮助及技術團隊
網 址 : http://www.be-friend.com.tw
E-MAIL : service@be-friend.com.tw
be.friend@msa.hinet.net

## 感謝主辦,承辦單位,各位老師

- 主辦單位:國立屏東高級工業職業學校
- 承辦單位:國立屏東高工電機科
- 最重要的是各位老師培育國家棟樑的推手
- 協辦單位:秉華科技有限公司

# 課程說明

- 第一天
  - 電腦鼠簡介，C語言編譯器，模擬系統操作說明
  - C語言語法規則：變數，常數，運算子和運算式
  - C語言流程控制，C語言程式庫說明
- 第二天
  - LED I/O埠控制，電路實習，硬體模擬燒錄及實驗
  - 七段顯示器控制電路實習軟，硬體模擬燒錄及實驗
- 第三天
  - Timer/中斷/PWM控制電路實習軟，硬體模擬燒錄及實驗
  - 步進馬達控制電路實習軟，硬體模擬燒錄及實驗

# 電腦鼠簡介

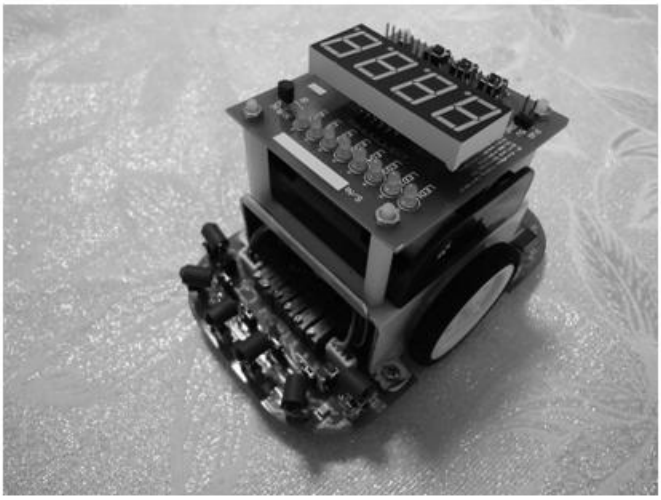
## 國際電腦鼠機械人競賽規則簡介

- 1952年克勞德.夏農在第八屆控制理論研討會首次介紹電腦鼠
- 1977年IEEE Spectrum提出了電腦鼠的概念:
  "電腦鼠是一個小型由微處理器控制的機械人車輛，在複雜的迷宮裡具有解碼和導航的功能"
  並在1979年舉辦第一次電腦鼠比賽。
- 最新的規則是由IEEE制定，詳細請參閱
  http://ieee.ucsd.edu/micromouse/files/micromouse_rules2010.pdf

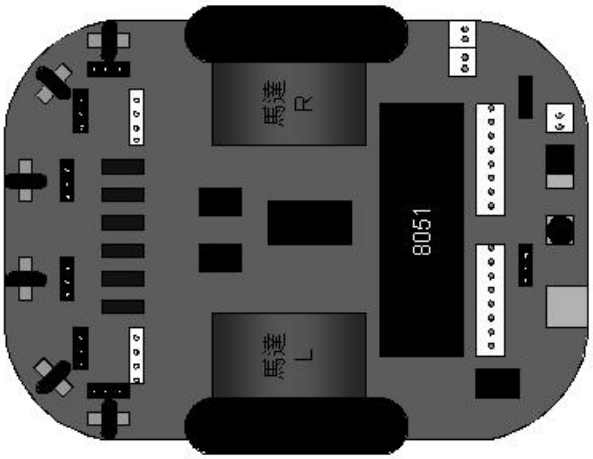
## 電腦鼠走迷宮標準套件

- 8*8傳統迷宮，爲電腦鼠國際賽四分之一大小，適合出其調適與學習使用。
- 16*16比賽迷宮，由四塊8*8迷宮拼成，完全符合IEEE國際標準。
  http://www.be-friend.com.tw/tw/product_detail.php?id=304
- 半尺寸迷宮
- 電腦鼠機器人(SCAR)爲秉華科技所製造的8051步進馬達電腦鼠，容易與學生8051課程銜接，很適合初學者使用。
- http://www.be-friend.com.tw/tw/product_detail.php?id=304

## 創意電腦鼠機器人簡介



## 車體元件分佈配置概略圖

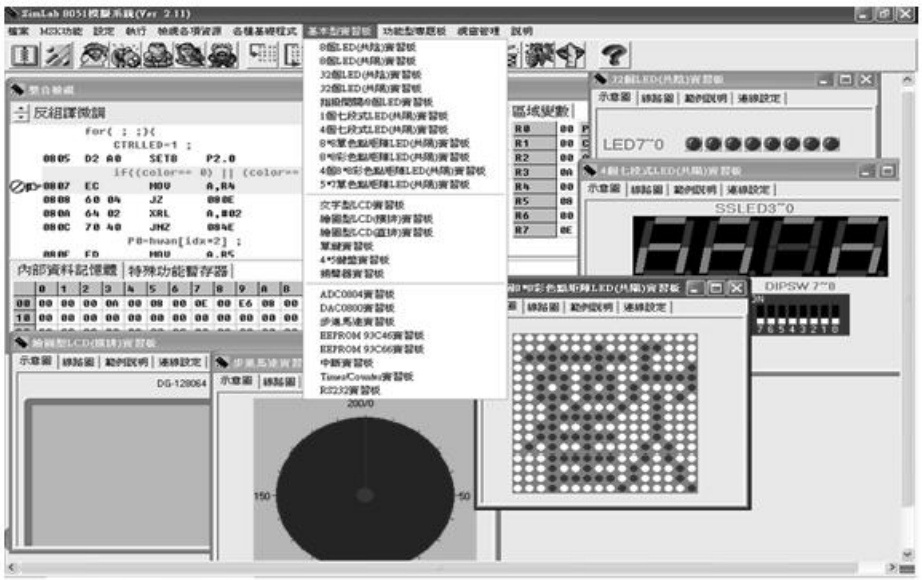


# C語言編譯器，模擬系統操作說明

## All-in-one 全功能開發系統

- 1.**模擬器**(軟體)-可設中斷點、單步除錯，觀察、修改記憶體及內部特殊暫存器內容等功能
- 2.**實驗器**(軟體)-內建30幾種常用專題製作實習模組，從基礎到專題系統整合
- 3.**組譯器**8K(Cross Assembler)
- 4.**編輯器**-PE2,記事本Notepad, Wordpad
- 5.**C編譯器**-加掛C51 Compiler (Franklin, Keil,ARM)
- 6.**模擬器**(硬體)-搭配MSK8051/51燒錄模擬器 可設中斷點、單步除錯等功能
- 7.**燒錄器**-可燒錄89S51/52,89C51/52 積體電路
- 8.**中文(繁體)**、英文版本

## SIMLAB-8051軟體模擬實驗器(I)



## SIMLAB 8051軟體模擬實驗器(II)

- **模擬器**(軟體)-可設中斷點、單步除錯，觀察、修改記憶體及內部特殊暫存器內容等功能
- **實驗器**(軟體)-內建30幾種常用專題製作實習模組，從基礎到專題系統整合
- **組譯器**8K(Cross Assembler
- **編輯器**-PE2,記事本Notepad, Wordpad
- **C編譯器**-加掛C51 Compiler
  (Franklin, Keil,ARM)

## 三十幾種專題製作實習模組

- 30幾個專題製作實習模組從基礎到專題系統整合
- 8個LED(共陰)實習板、8個LED(共陽)實習板
- 32個LED(共陰)實習板、32個LED(共陽)實習板
- 指撥開關/8個LED實習板、1個七段式LED實習板
- 4個七段式LED實習板、8*8單色點矩陣LED實習板
- 8*8彩色點矩陣LED實習板、4個8*8彩色點矩陣LED實習板
- 5*7單色點矩陣LED實習板、文字型LCD實習板
- 繪圖型LCD(橫排)實習板、繪圖型LCD(直排)實習板
- 單鍵實習板、4*5鍵盤實習板、揚聲器實習板
- ADC0804實習板、DAC0800實習板、步進馬達實習板
- EEPROM 93C46實習板、EEPROM 93C66實習板
- 中斷實習板、Timer/Counter實習板、RS232實習板
- 反組譯程式實習板、自動測試反組繹程式實習板
- 自動測試行組繹程式、邏輯閘測試器、紅綠燈控制實習板
- 計時計數器實習板、示波器字幕控制實習板模組

# SIMLAB-8051

## 單晶片微電腦發展系統

(參考資料：**SIMLAB-8051**使用手冊，秉華科技有限公司)

## SimLab-8051特色

- 完全軟體模擬，不需要任何硬體接線
- 具一般模擬器的單步除錯功能
- 可隨時觀察、修改記憶體及內部特殊暫存器內容
- 支援高階**C**語言程式發展系統
- 內建程式組譯器或可選擇其他組譯器、編輯器
- 可以建立專案規劃
- 程式和硬體線路可列印
- 內建三十幾種常用專題製作實習模組，從基礎到整合
- 馬上安裝、馬上學習、快速方便

## SimLab-8051安裝

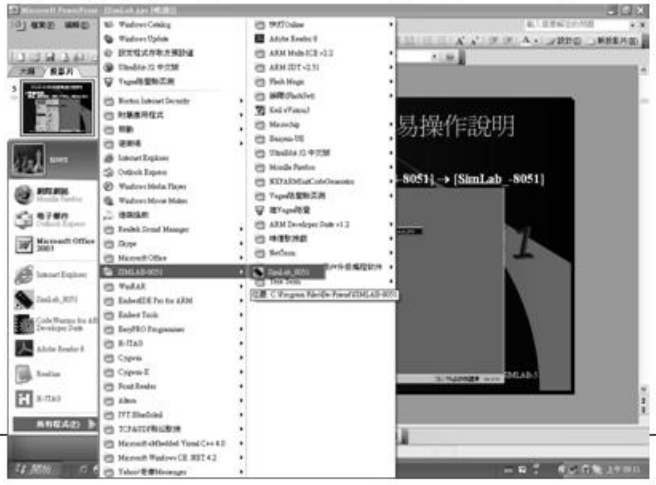
- 硬體環境需求：
  - **586** 以上相容性電腦一部
  - 至少一部光碟機
  - 最少 **16M RAM**
  - **WINDOW 7(32BIT)/XP/2000/ME/98/95** 操作環境

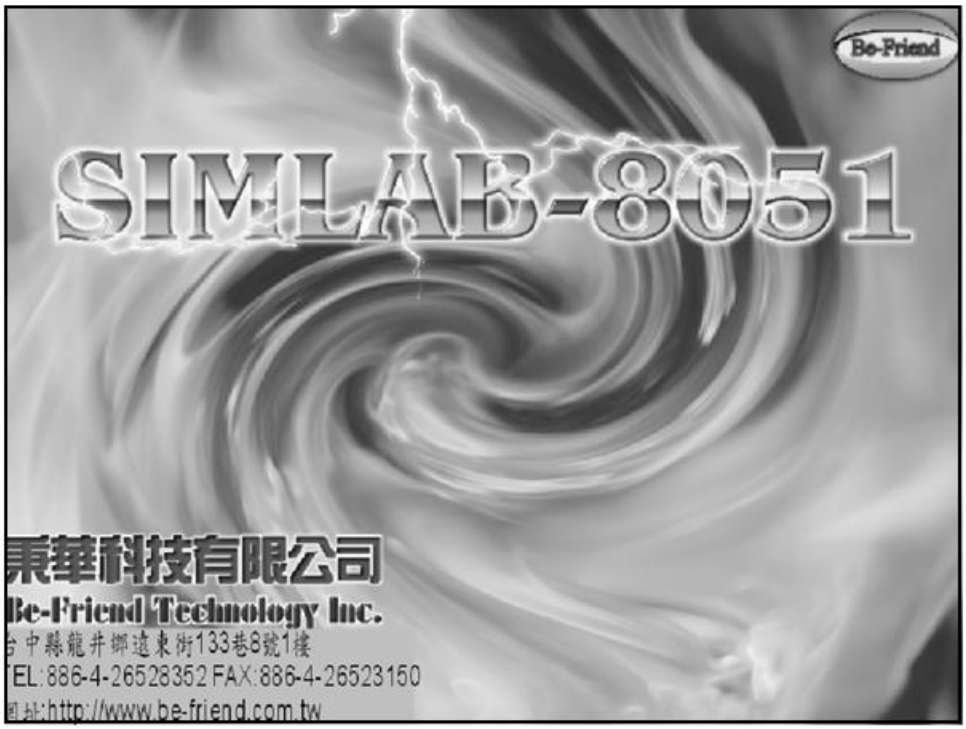
- 軟體安裝:
  - 把**SimLab-8051**的**keypro軟體保護鎖**接在印表機埠上，然後開機進入**WINDOWS**環境。
  - 安裝**SimLab-8051**時，執行光碟片**DISK1**目錄下**setup**安裝程式，然後依照提示按確認即可。

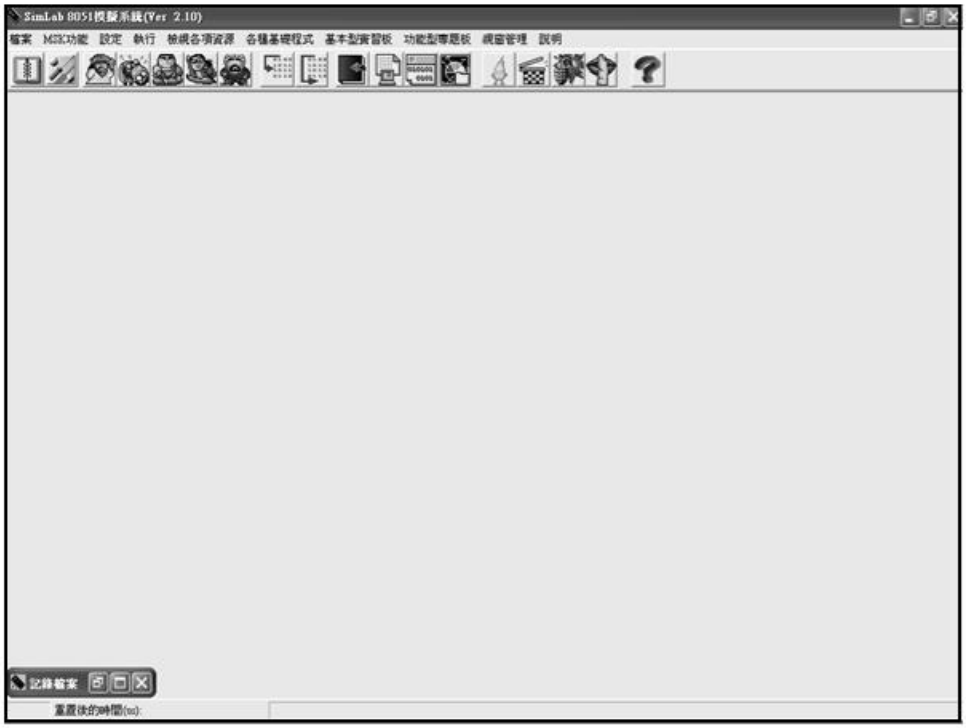
# SimLab-8051簡易操作說明

- 用滑鼠點選：

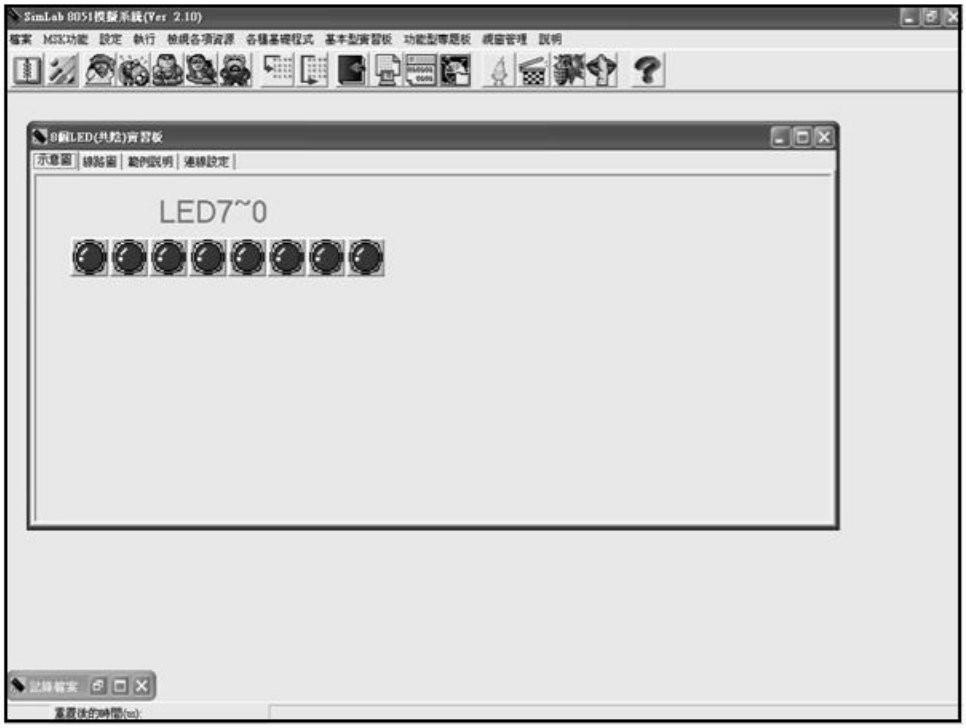
  **[開始]→ [程式集(P)] → [SIMLAB-8051] → [SimLab_-8051]**

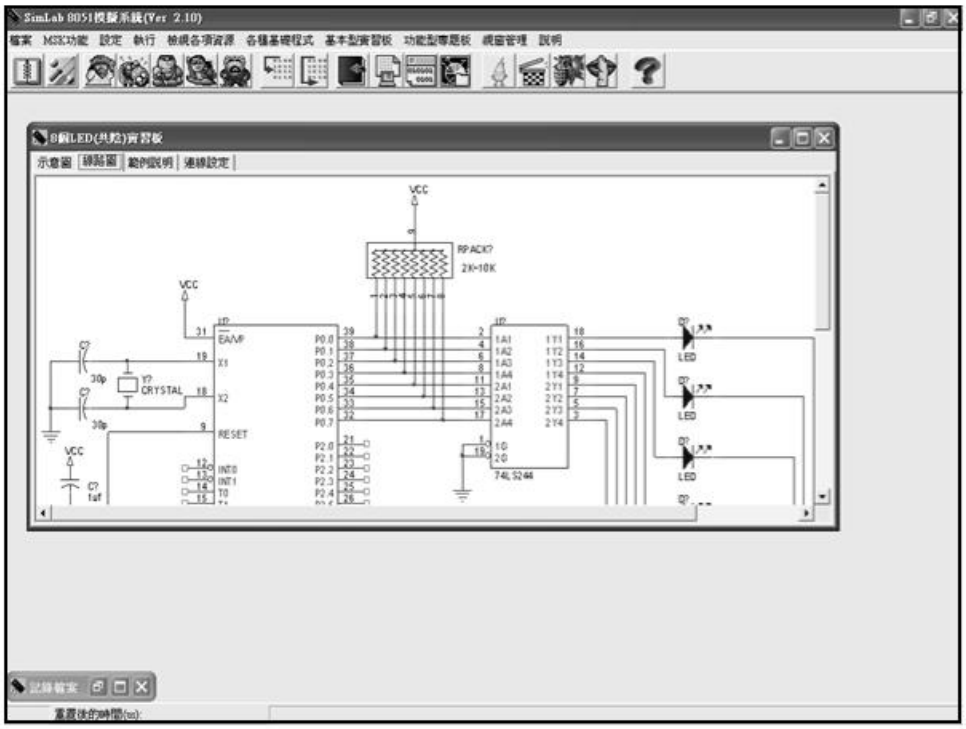



Be-Friend
SIMLAB-8051
秉華科技有限公司
Be-Friend Technology Inc.
台中縣龍井鄉遠東街133巷8號1樓
TEL:886-4-26528352 FAX:886-4-26523150
網址:http://www.be-friend.com.tw

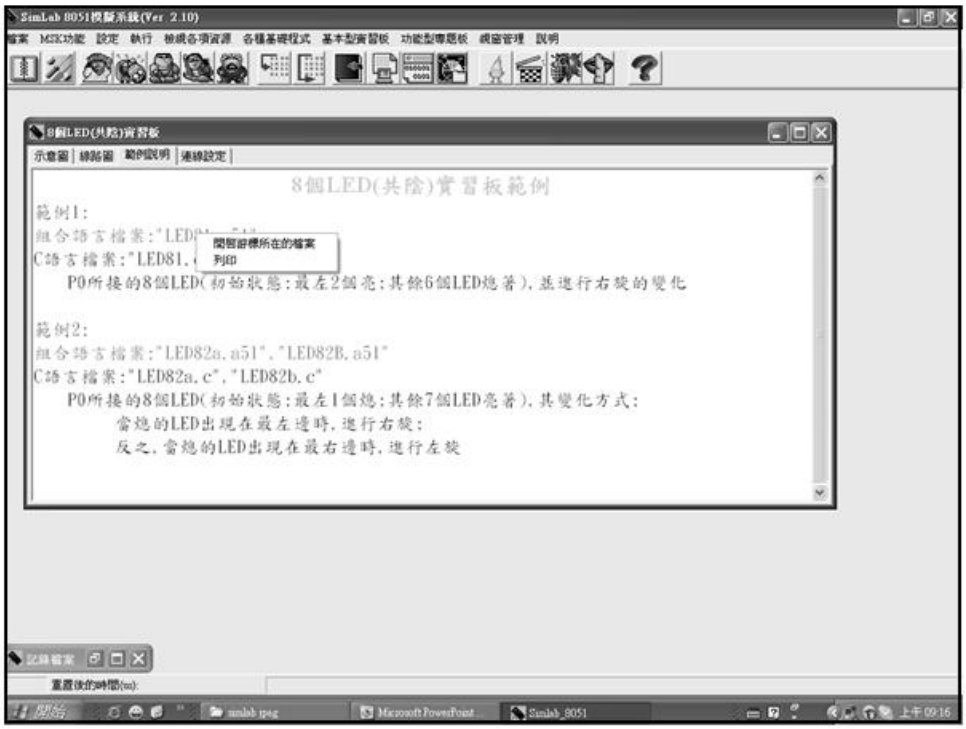

SimLab 8051模擬系統(Ver 2.10)
檔案 MCK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
記錄檔案
重置後的時間(us):


SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8個LED(共陰)實習板
8個LED(共陽)實習板
32個LED(共陰)實習板
32個LED(共陽)實習板
指撥開關8個LED實習板
1個七段式LED(共陽)實習板
4個七段式LED(共陽)實習板
8*8單色點矩陣LED(共陽)實習板
8*8彩色點矩陣LED(共陽)實習板
4個8*8彩色點矩陣LED(共陽)實習板
5*7單色點矩陣LED(共陽)實習板
文字型LCD實習板
繪圖型LCD(橫排)實習板
繪圖型LCD(直排)實習板
單鍵實習板
4*5鍵盤實習板
揚聲器實習板
ADC0804實習板
DAC0800實習板
步進馬達實習板
EEPROM 93C46實習板
EEPROM 93C66實習板
中斷實習板
Timer/Counter實習板
RS232實習板
記錄檔案
重置依存時間(ns):


SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8個LED(共陰)實習板
示意圖 線路圖 範例說明 連線設定
LED7~0
記錄檔案
重置依存時間(ns):

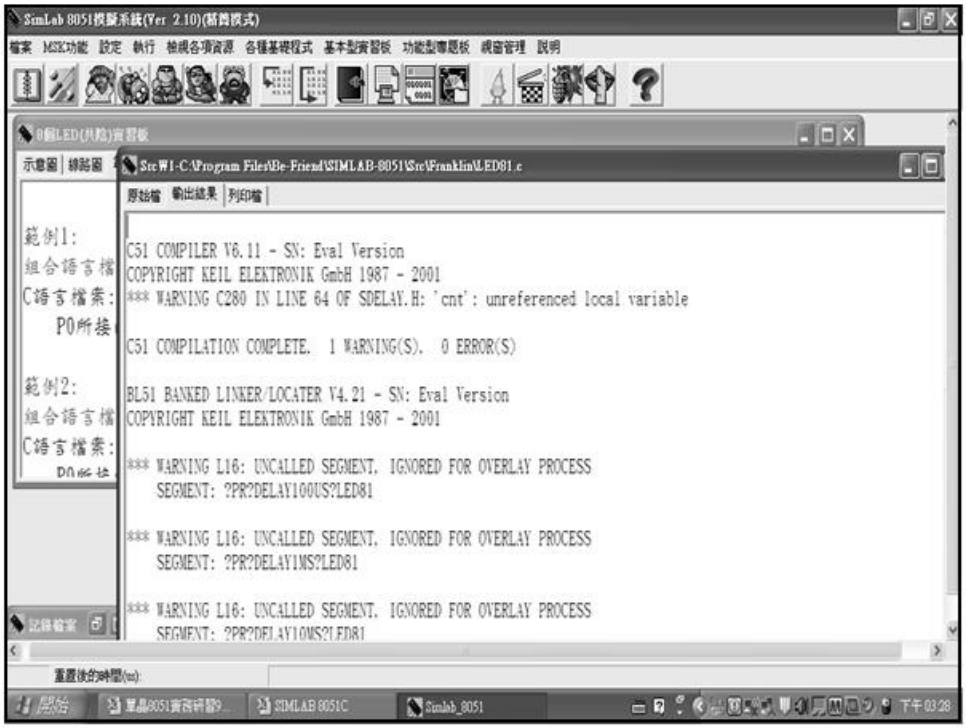

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8個LED(共陰)實習板
示意圖 線路圖 範例說明 連線設定
VCC
RPACK7
2K-10K
EA/VP
X1
X2
CRYSTAL
RESET
INT0
INT1
T0
74LS244
LED
記錄檔案
重置後的時間(ns):

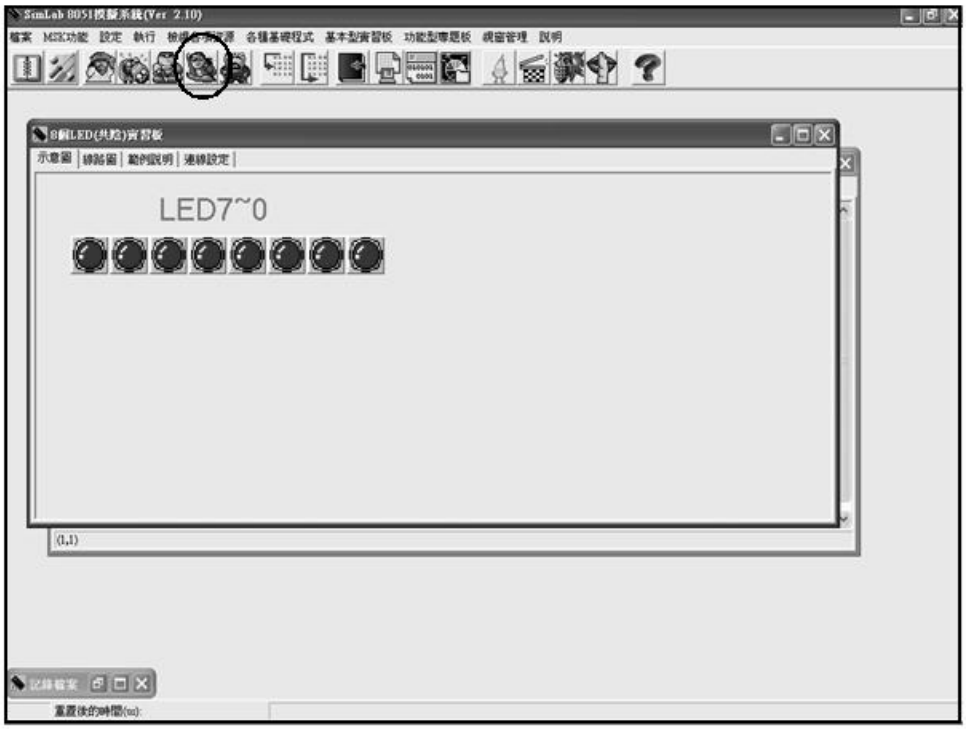

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8個LED(共陰)實習板
示意圖 線路圖 範例說明 連線設定
8個LED(共陰)實習板範例
範例1:
組合語言檔案:"LED81.a51"
C語言檔案:"LED81.c"
P0所接的8個LED(初始狀態:最左2個亮;其餘6個LED熄著),並進行右旋的變化
範例2:
組合語言檔案:"LED82a.a51","LED82B.a51"
C語言檔案:"LED82a.c","LED82b.c"
P0所接的8個LED(初始狀態:最左1個熄;其餘7個LED亮著),其變化方式:
當熄的LED出現在最左邊時,進行右旋;
反之,當熄的LED出現在最右邊時,進行左旋
記錄檔案
重置後的時間(ns):

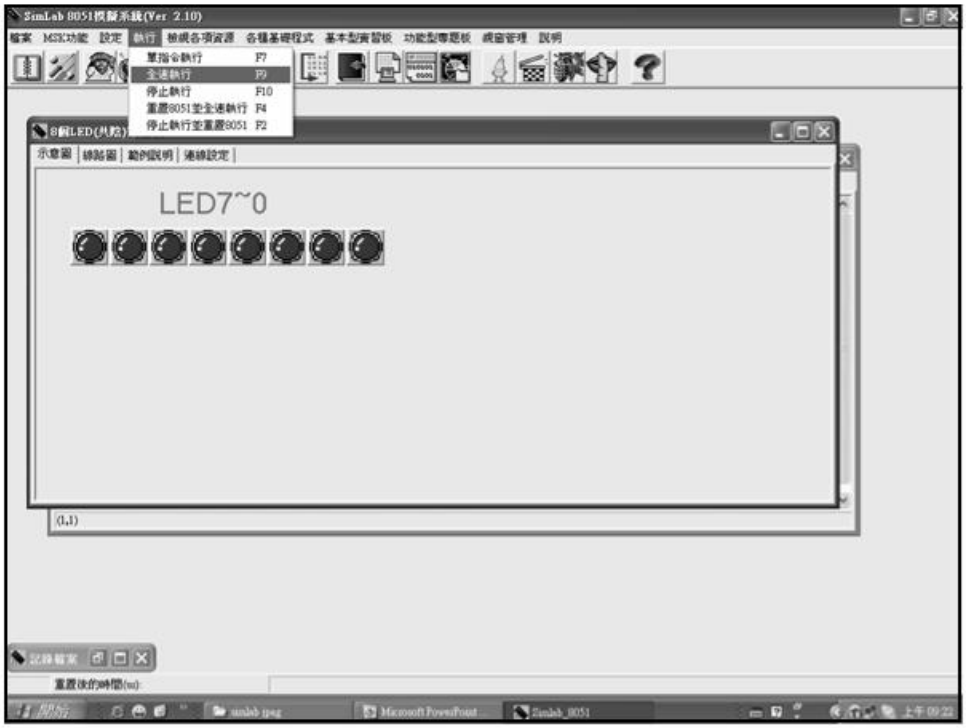

SimLab 8051模擬系統(Ver 2.10)
8個LED(共陰)實習板
8個LED(共陰)實習板範例
範例1:
組合語言檔案:"LED
C語言檔案:"LED81.
P0所接的8個LED(初始狀態:最左2個亮;其餘6個LED熄著),並進行右旋的變化
範例2:
組合語言檔案:"LED82a.a51","LED82B.a51"
C語言檔案:"LED82a.c","LED82b.c"
P0所接的8個LED(初始狀態:最左1個熄;其餘7個LED亮著),其變化方式:
當熄的LED出現在最左邊時,進行右旋;
反之,當熄的LED出現在最右邊時,進行左旋

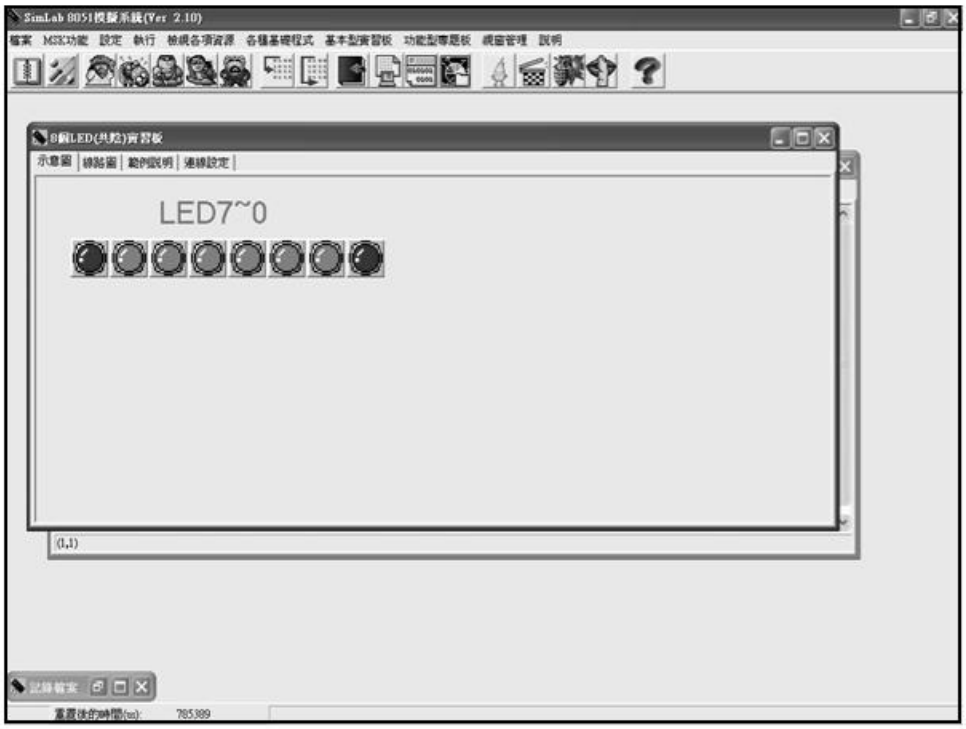

SimLab 8051模擬系統(Ver 2.10)(精簡模式)
8個LED(共陰)實習板
Src#1-C:\Program Files\Be-Friend\SIMLAB-8051\Src\Franklin\LED81.c
原始檔
輸出結果
列印檔
//
// 配合"8個LED(共陰)實習板"
// P0所接的8個LED(初始狀態:最左2個亮;其餘6個LED熄著),並進行右旋的變化
//
#pragma oe db pw(80) SM SB CD
#include <reg51.h>
#include "Delay.h"
main()
{
P0 =0xc0;
for(;;) {
Delay100ms() ;
if(P0&0x01){ //P0右旋1位元
P0 >>=1 ;
P0 |=0x80 ;
}
(10,12)

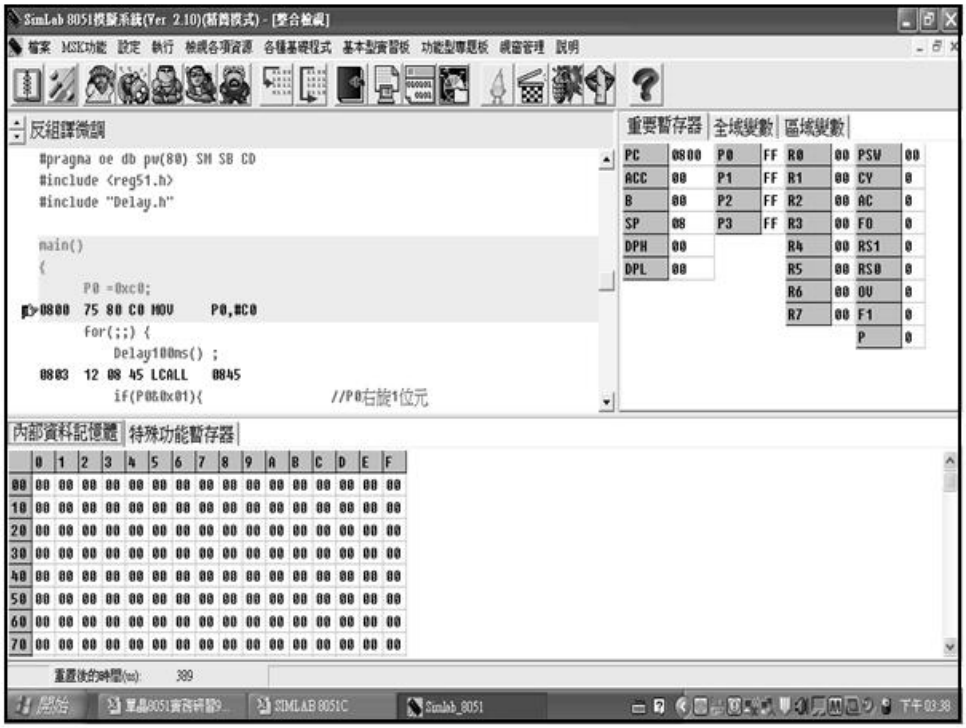

SimLab 8051模擬系統(Ver 2.10)(精簡模式)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8顆LED(共陰)實習板
示意圖 線路圖
Src\W1-C:\Program Files\Be-Friend\SIMLAB-8051\Src\Franklin\LED01.c
原始檔 輸出結果 列印檔
範例1:
組合語言檔
C語言檔案:
P0所接
範例2:
組合語言檔
C語言檔案:
//
// 配合"8個LED(共陰)實習板"
// P0所接的8個LED(初始狀態:最左2個亮;其餘6個LED熄滅),並進行右旋的變化
//
#pragma oe db pw(80) SM SB CD
#include <reg51.h>
#include "Delay.h"
main()
{
P0 =0xc0;
for(;;) {
Delay100ms() ;
if(P0&0x01){ //P0右旋1位元
P0 >>=1 ;
P0 |=0x80 ;
}
儲存、組譯及載入(慢速延遲時間)(S)
儲存、組譯及載入(快速延遲時間)(T)
剪下(U)
複製(Y)
貼上(W)
儲存檔案(A)
另存新檔案(T)
列印(Z)
(7,11)
記錄檔案
重置後的時間(ms):
開始
單晶8051實務研習9...
SIMLAB 8051C
Simlab_8051
下午 03:27


SimLab 8051模擬系統(Ver 2.10)(精簡模式)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8顆LED(共陰)實習板
示意圖 線路圖
Src\W1-C:\Program Files\Be-Friend\SIMLAB-8051\Src\Franklin\LED01.c
原始檔 輸出結果 列印檔
範例1:
組合語言檔
C語言檔案:
範例2:
組合語言檔
C語言檔案:
C51 COMPILER V6.11 - SN: Eval Version
COPYRIGHT KEIL ELEKTRONIK GmbH 1987 - 2001
*** WARNING C280 IN LINE 64 OF SDELAY.H: 'cnt': unreferenced local variable
C51 COMPILATION COMPLETE. 1 WARNING(S), 0 ERROR(S)
BL51 BANKED LINKER/LOCATER V4.21 - SN: Eval Version
COPYRIGHT KEIL ELEKTRONIK GmbH 1987 - 2001
*** WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS
SEGMENT: ?PR?DELAY100US?LED01
*** WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS
SEGMENT: ?PR?DELAY1MS?LED01
*** WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS
SEGMENT: ?PR?DELAY10MS?LED01
記錄檔案
重置後的時間(ms):
開始
單晶8051實務研習9...
SIMLAB 8051C
Simlab_8051
下午 03:28

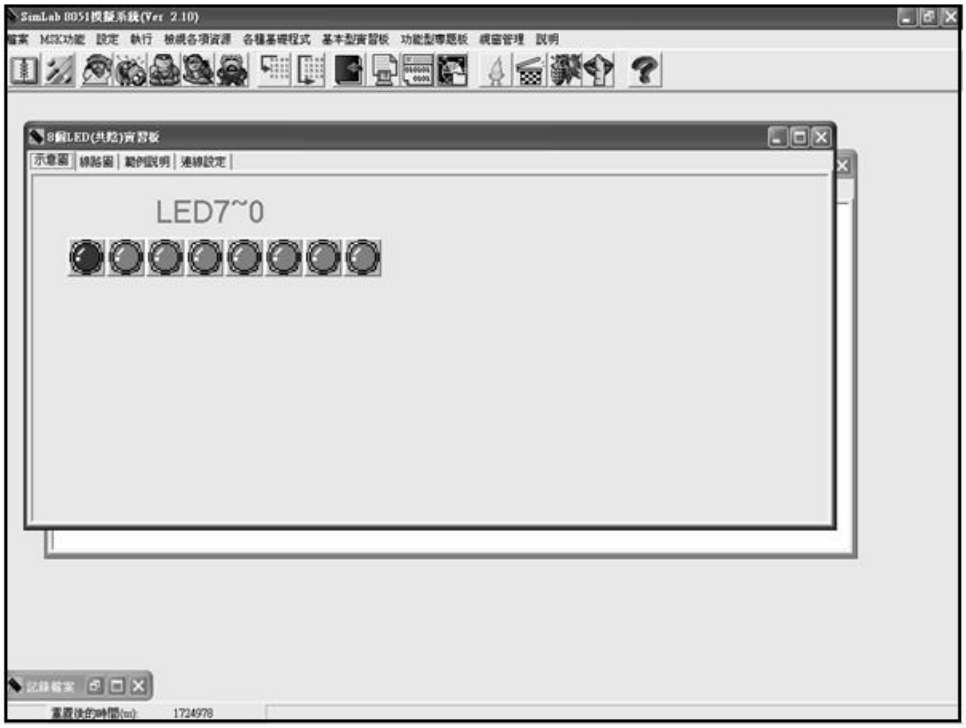

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8個LED(共陰)實習板
示意圖 線路圖 範例說明 連線設定
LED7~0
(1,1)
記錄檔案
重置後的時間(ns):

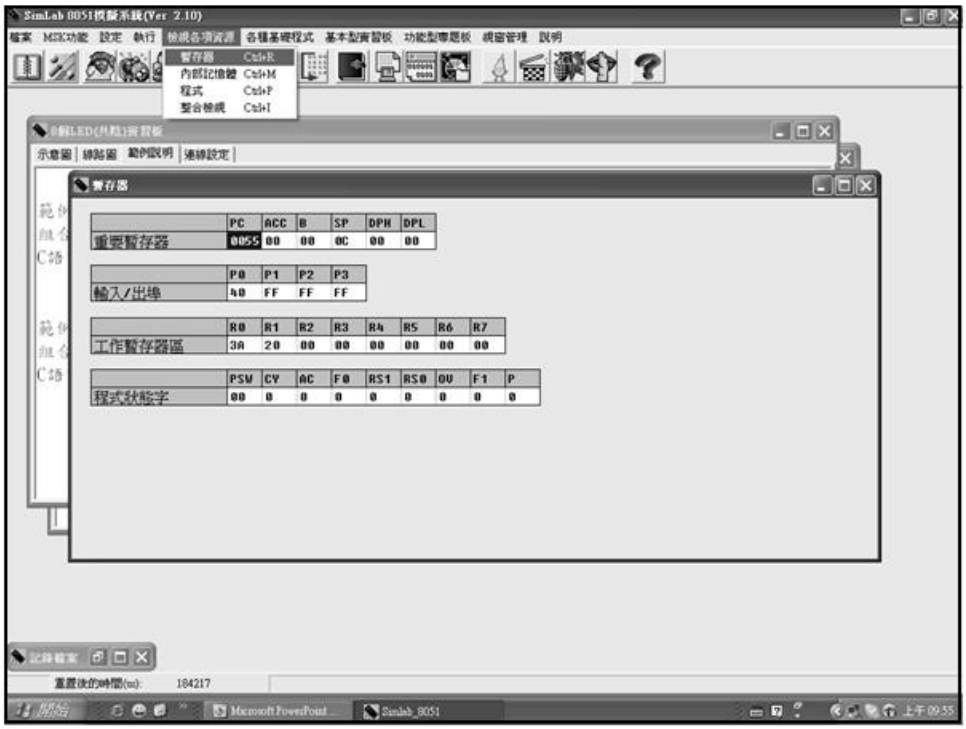

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
單指令執行 F7
全速執行 F9
停止執行 F10
重置8051並全速執行 F4
停止執行並重置8051 F2
8個LED(共陰)
示意圖 線路圖 範例說明 連線設定
LED7~0
(1,1)
記錄檔案
重置後的時間(ns):

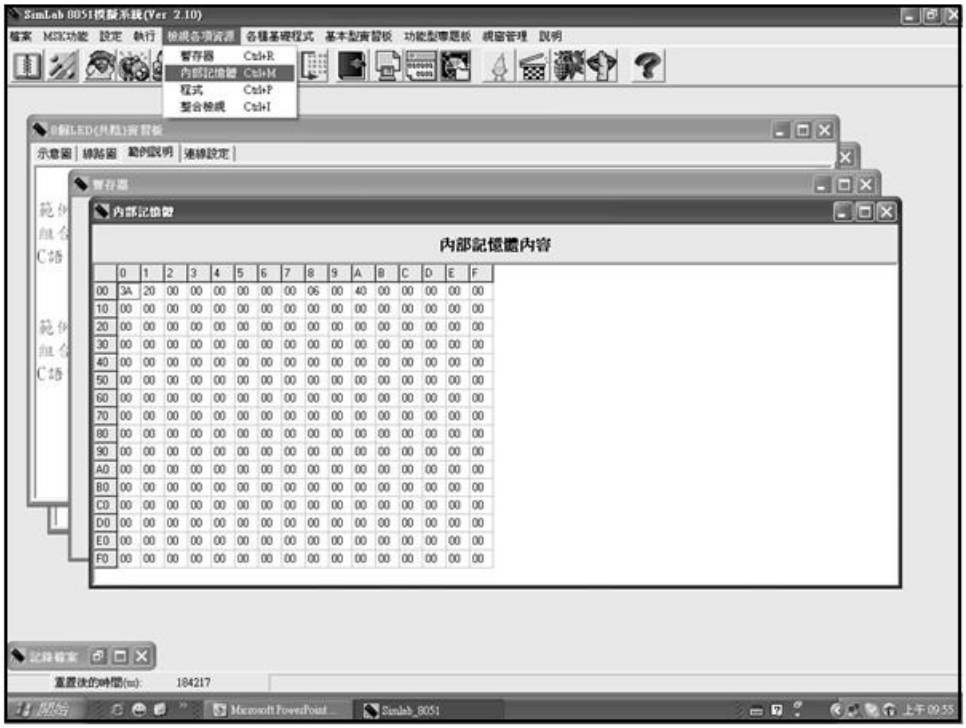

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8個LED(共陰)實習板
示意圖 線路圖 範例說明 連線設定
LED7~0
(1,1)
記錄檔案
重置後的時間(us): 785389

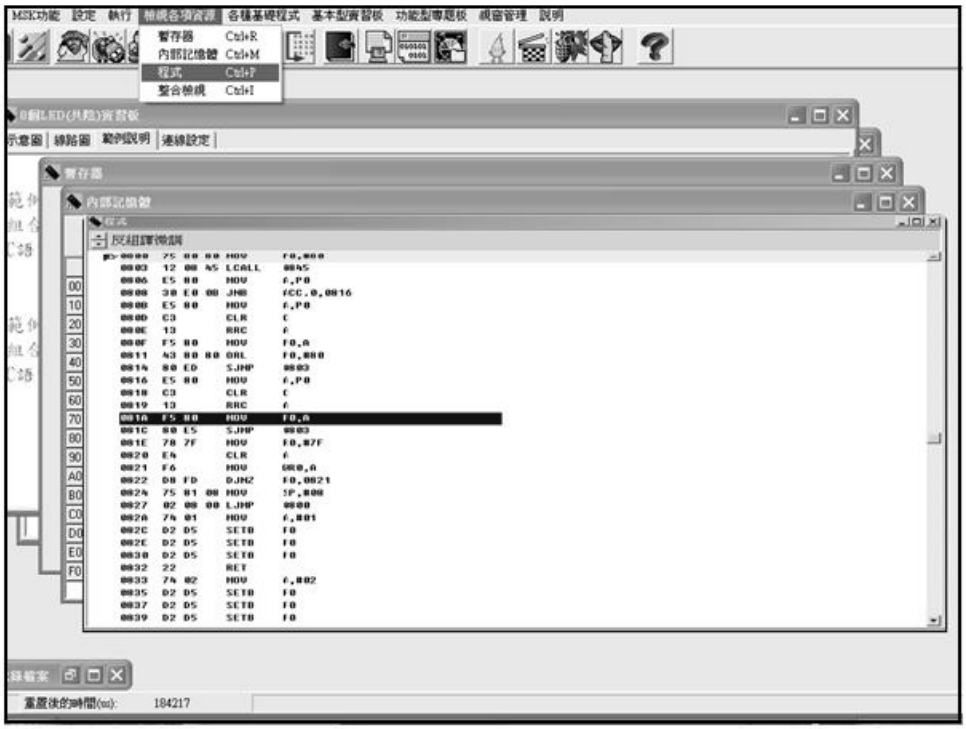

SimLab 8051模擬系統(Ver 2.10)(精簡模式) - [整合檢視]
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
反組譯微調
#pragma oe db pw(80) SM SB CD
#include <reg51.h>
#include "Delay.h"
main()
{
P0 =0xc0;
0800 75 80 C0 MOV P0,#C0
for(;;) {
Delay100ms() ;
0803 12 08 45 LCALL 0845
if(P0&0x01){ //P0右旋1位元
重要暫存器 全域變數 區域變數
PC 0800 P0 FF R0 00 PSW 00
ACC 00 P1 FF R1 00 CY 0
B 00 P2 FF R2 00 AC 0
SP 08 P3 FF R3 00 F0 0
DPH 00 R4 00 RS1 0
DPL 00 R5 00 RS0 0
R6 00 OV 0
R7 00 F1 0
P 0
內部資料記憶體 特殊功能暫存器
0 1 2 3 4 5 6 7 8 9 A B C D E F
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
重置後的時間(us): 399
開始
SIMLAB 8051C
Simlab_8051
下午 03:38

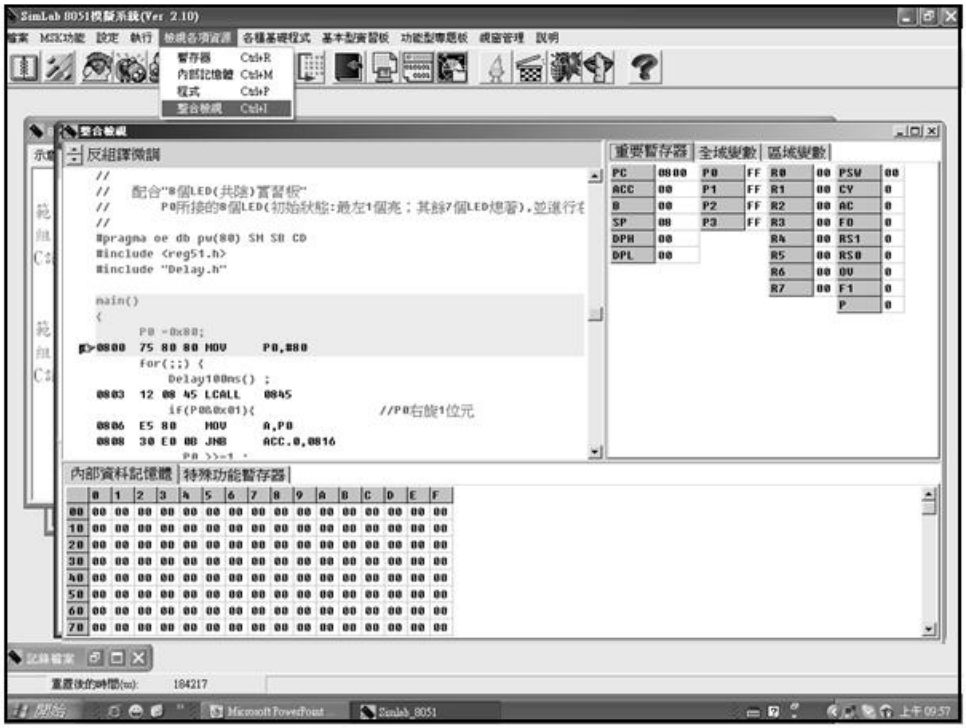

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
Src W1-C:\Program Files\Be-Friend\SIMLAB-8051\Src\Franklin\LED81.c
原始檔 | 輸出結果 | 列印檔
//
// 配合"8個LED(共陰)實習板"
// P0所接的8個LED(初始狀態:最左1個亮;其餘7個熄滅),並進行右移的變化
//
#pragma oe db pw(80) SM SB CD
#include <reg51.h>
#include "Delay.h"
main()
{
P0 =0x80;
for(;;) {
Delay100ms() ;
if(P0&0x01){ //P0右移1位元
P0 >>=1 ;
P0 |=0x80 ;
}
else{
P0 >>= 1 ;
}
}
}
重置後的時間(ms): 285813970
開始
上午 07:51

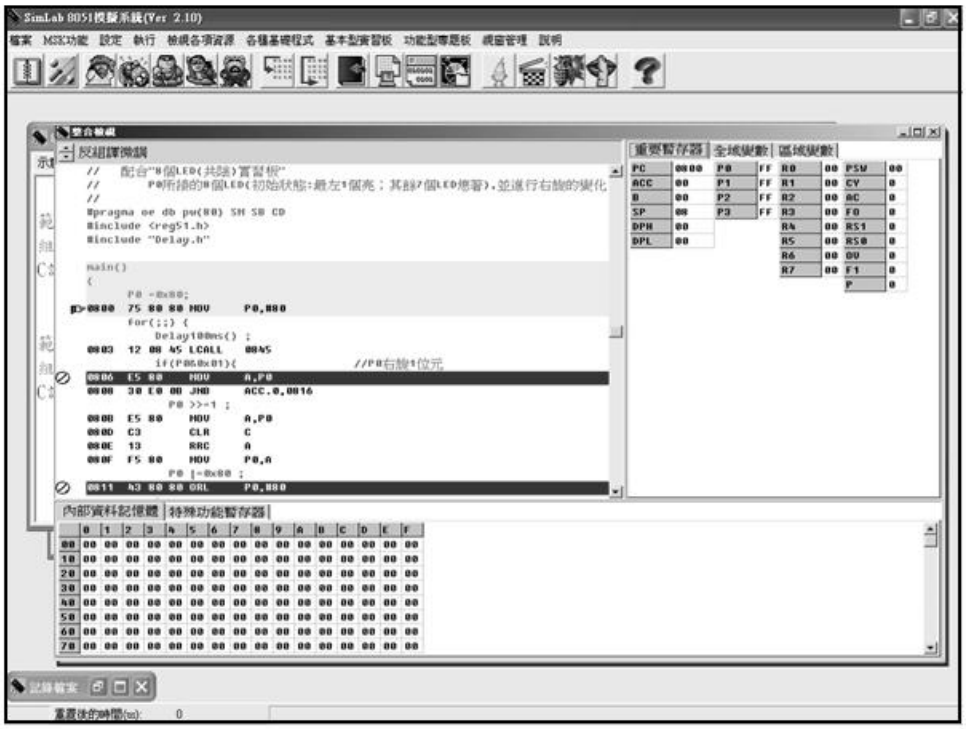

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
8個LED(共陰)實習板
示意圖 線路圖 範例說明 連線設定
LED7~0
記錄檔案
重置後的時間(ms): 1724978

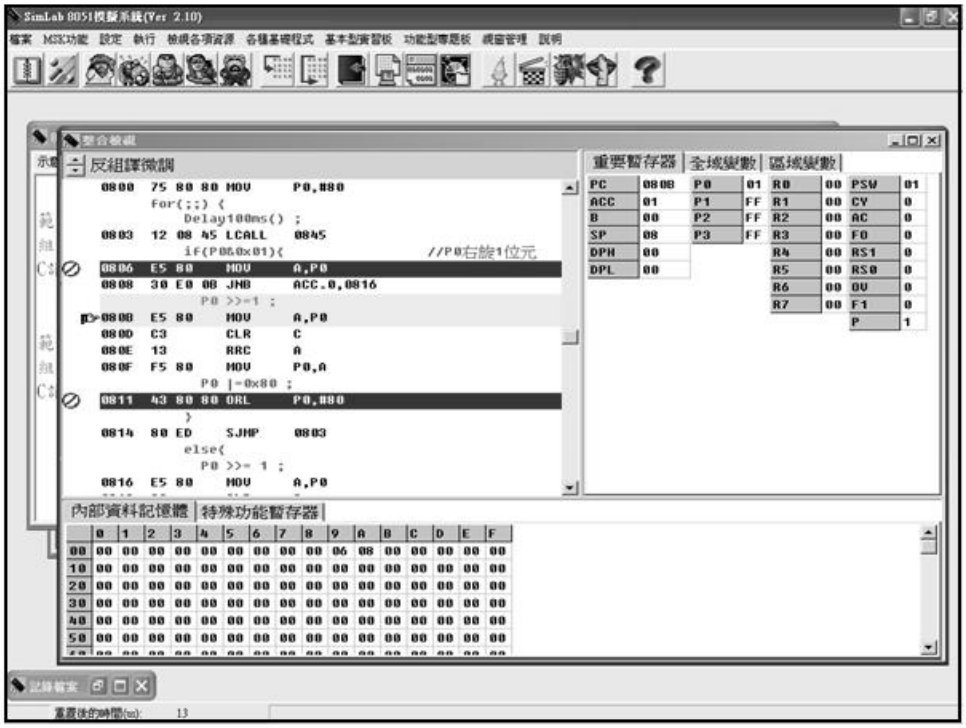

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
暫存器 Ctrl+R
內部記憶體 Ctrl+M
程式 Ctrl+P
整合檢視 Ctrl+I
示意圖 線路圖 範例說明 連線設定
暫存器
PC ACC B SP DPH DPL
重要暫存器 0055 00 00 0C 00 00
P0 P1 P2 P3
輸入/出埠 40 FF FF FF
R0 R1 R2 R3 R4 R5 R6 R7
工作暫存器區 3A 20 00 00 00 00 00 00
PSW CY AC F0 RS1 RS0 OV F1 P
程式狀態字 00 0 0 0 0 0 0 0 0
重置後的時間(ns): 184217
Microsoft PowerPoint ...
Simlab_8051
上午 09:55

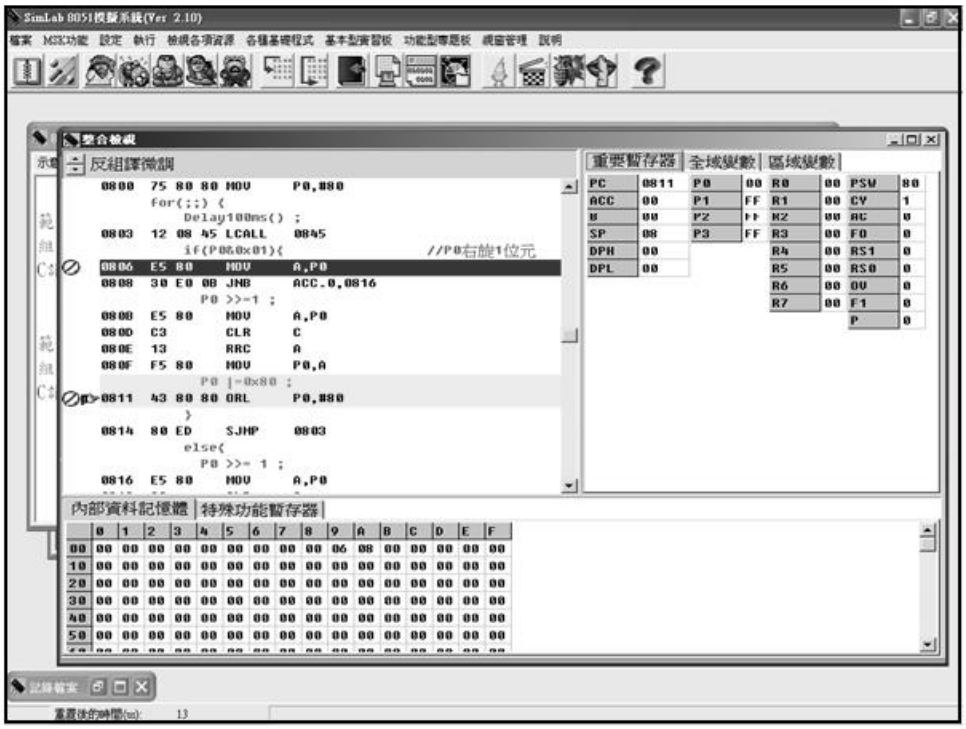

SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
暫存器 Ctrl+R
內部記憶體 Ctrl+M
程式 Ctrl+P
整合檢視 Ctrl+I
示意圖 線路圖 範例說明 連線設定
暫存器
內部記憶體
內部記憶體內容
0 1 2 3 4 5 6 7 8 9 A B C D E F
00 3A 20 00 00 00 00 00 00 06 00 40 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
重置後的時間(ns): 184217
Microsoft PowerPoint ...
Simlab_8051
上午 09:55

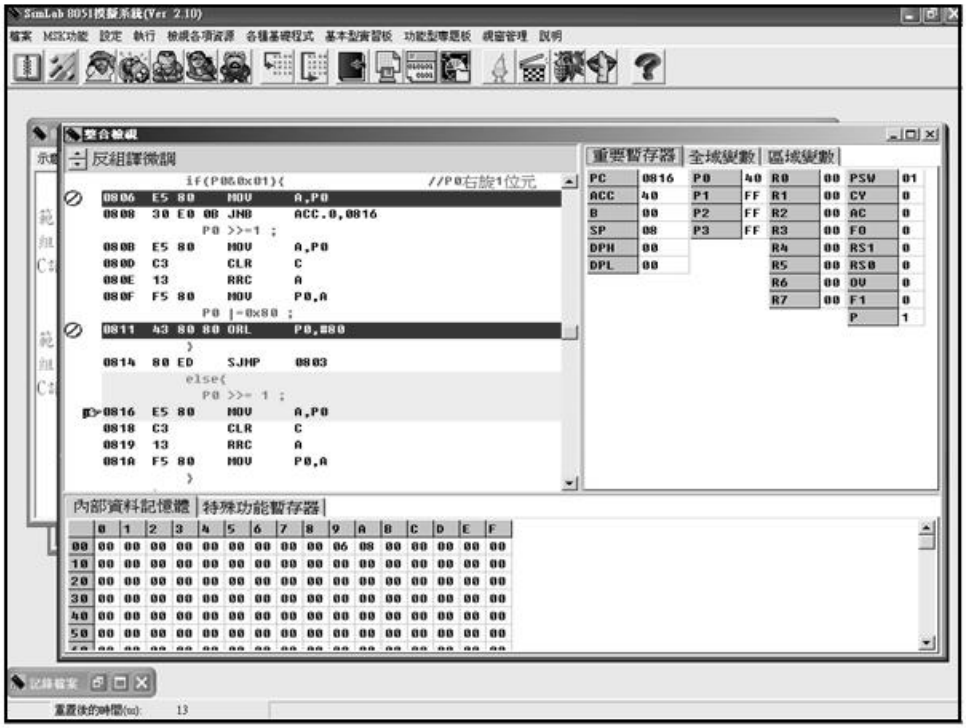

暫存器 Ctrl+R
內部記憶體 Ctrl+M
程式 Ctrl+P
整合檢視 Ctrl+I
反組譯微調
重置後的時間(ns): 184217

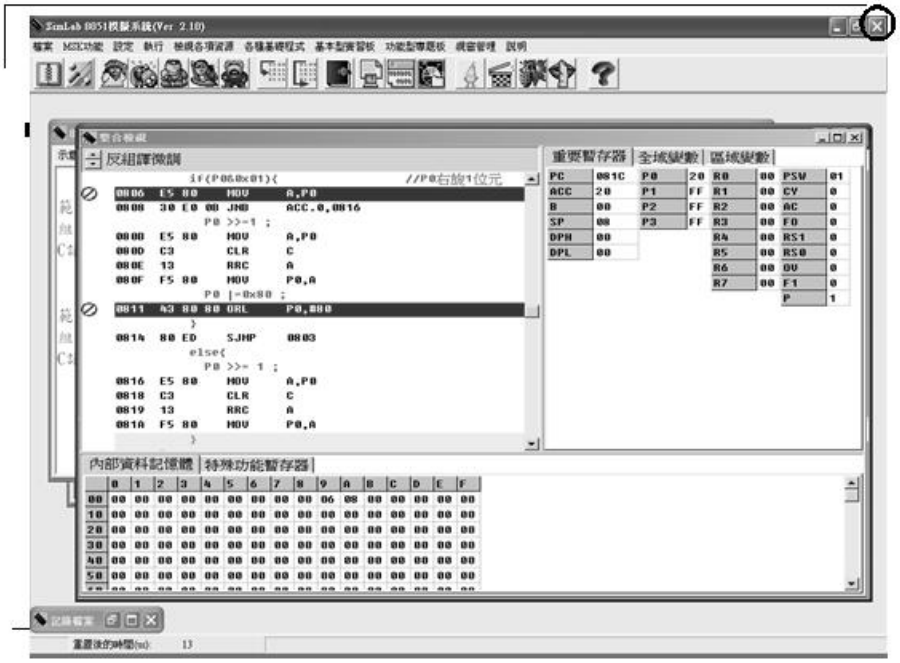

SimLab 8051模擬系統(Ver 2.10)
暫存器 Ctrl+R
內部記憶體 Ctrl+M
程式 Ctrl+P
整合檢視 Ctrl+I
反組譯微調
重置後的時間(ns): 184217


SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
整合檢視
反組譯微調
// 配合"8個LED(共陰)實習板"
// P0所插的8個LED(初始狀態:最左1個亮；其餘7個LED熄滅),並進行右旋的變化
//
#pragma oe db pw(80) SM SB CD
#include <reg51.h>
#include "Delay.h"
main()
{
P0 =0x80;
0800 75 80 80 MOV P0,#80
for(;;) {
Delay100ms() ;
0803 12 08 45 LCALL 0845
if(P0&0x01){ //P0右旋1位元
0806 E5 80 MOV A,P0
0808 30 E0 08 JNB ACC.0,0816
P0 >>=1 ;
080B E5 80 MOV A,P0
080D C3 CLR C
080E 13 RRC A
080F F5 80 MOV P0,A
P0 |=0x80 ;
0811 43 80 80 ORL P0,#80
重要暫存器
全域變數
區域變數
內部資料記憶體
特殊功能暫存器
記錄檔案
重置後的時間(ms): 0


SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
整合檢視
反組譯微調
0800 75 80 80 MOV P0,#80
for(;;) {
Delay100ms() ;
0803 12 08 45 LCALL 0845
if(P0&0x01){ //P0右旋1位元
0806 E5 80 MOV A,P0
0808 30 E0 08 JNB ACC.0,0816
P0 >>=1 ;
080B E5 80 MOV A,P0
080D C3 CLR C
080E 13 RRC A
080F F5 80 MOV P0,A
P0 |=0x80 ;
0811 43 80 80 ORL P0,#80
}
0814 80 ED SJMP 0803
else{
P0 >>= 1 ;
0816 E5 80 MOV A,P0
重要暫存器
全域變數
區域變數
內部資料記憶體
特殊功能暫存器
記錄檔案
重置後的時間(ms): 13


SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
整合檢視
反組譯微調
0800 75 80 80 MOV P0,#80
for(;;) {
Delay100ms() ;
0803 12 08 45 LCALL 0845
if(P0&0x01){ //P0右旋1位元
0806 E5 80 MOV A,P0
0808 30 E0 08 JNB ACC.0,0816
P0 >>=1 ;
080B E5 80 MOV A,P0
080D C3 CLR C
080E 13 RRC A
080F F5 80 MOV P0,A
P0 |=0x80 ;
0811 43 80 80 ORL P0,#80
}
0814 80 ED SJMP 0803
else{
P0 >>= 1 ;
0816 E5 80 MOV A,P0
重要暫存器 全域變數 區域變數
PC 0811 P0 00 R0 00 PSW 80
ACC 00 P1 FF R1 00 CY 1
B 00 P2 FF R2 00 AC 0
SP 08 P3 FF R3 00 F0 0
DPH 00 R4 00 RS1 0
DPL 00 R5 00 RS0 0
R6 00 OV 0
R7 00 F1 0
P 0
內部資料記憶體 特殊功能暫存器
0 1 2 3 4 5 6 7 8 9 A B C D E F
00 00 00 00 00 00 00 00 00 00 06 08 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
記錄檔案
重置後的時間(us): 13


SimLab 8051模擬系統(Ver 2.10)
檔案 MSK功能 設定 執行 檢視各項資源 各種基礎程式 基本型實習板 功能型專題板 視窗管理 說明
整合檢視
反組譯微調
if(P0&0x01){ //P0右旋1位元
0806 E5 80 MOV A,P0
0808 30 E0 08 JNB ACC.0,0816
P0 >>=1 ;
080B E5 80 MOV A,P0
080D C3 CLR C
080E 13 RRC A
080F F5 80 MOV P0,A
P0 |=0x80 ;
0811 43 80 80 ORL P0,#80
}
0814 80 ED SJMP 0803
else{
P0 >>= 1 ;
0816 E5 80 MOV A,P0
0818 C3 CLR C
0819 13 RRC A
081A F5 80 MOV P0,A
重要暫存器 全域變數 區域變數
PC 0816 P0 40 R0 00 PSW 01
ACC 40 P1 FF R1 00 CY 0
B 00 P2 FF R2 00 AC 0
SP 08 P3 FF R3 00 F0 0
DPH 00 R4 00 RS1 0
DPL 00 R5 00 RS0 0
R6 00 OV 0
R7 00 F1 0
P 1
內部資料記憶體 特殊功能暫存器
0 1 2 3 4 5 6 7 8 9 A B C D E F
00 00 00 00 00 00 00 00 00 00 06 08 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
記錄檔案
重置後的時間(us): 13




# 軟體操作

- 安裝SIMLAB 8051軟體
- 將Franklin C Demo (2k C51編譯)解壓縮到硬碟C槽根目錄
- 將電腦鼠USB線接到電腦插槽，已安裝硬體FT232R驅動程式
- 安裝Nuvoton 8051以燒錄華邦電的8051 IC W78E052D

# SIMLAB 8051 設定、編譯、除錯

- 開啓SIMLAB 8051


# SIMLAB 8051 設定、編譯、除錯

- 掛載編譯器並設定石英震盪器頻率

# SIMLAB 8051 設定、編譯、除錯

- 開新C語言檔案，打完程式並另存新檔到自建的資料夾(副檔名要打.C)


# SIMLAB 8051 設定、編譯、除錯

- 複製光碟內SDelay.h檔(25M(Hz)專用)到剛建立的資料夾裡面

# SIMLAB 8051 設定、編譯、除錯

- 再回SIMLAB 80511編譯我們的測試程式，(實際延遲)


# SIMLAB 8051 設定、編譯、除錯

- 產生輸出結果(顯示0個警告0個錯誤)，假設有錯誤或警告 可以在看輸出結果旁列印檔

# SIMLAB 8051 設定、編譯、除錯

- 假設有錯誤該如何抓出來呢(除錯方式)?我把剛剛的程式幾個地方改掉(改成錯誤的)我們再來學習抓錯誤的方法。以下是常見的幾種錯誤
  - 大小寫問題 (C語言是大小寫敏感的)
  - 少打或多打”;”或”{””}”或是打錯符號
  - 數值打錯(可能並不會出現錯誤或警告，但程式執行結果不正確。)

# SIMLAB 8051 設定、編譯、除錯

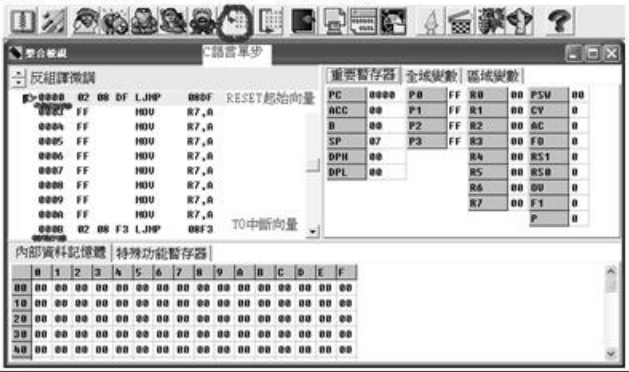
- 第一第二種錯誤，我們利用編譯後產生的列印檔看問體在哪，在判斷問題而修改。

## SIMLAB 8051 設定、編譯、除錯

- 第三種錯誤由於”程式語法”並沒有問題，我們並看不出錯誤或警告，我們利用燒出來的程式行爲判斷(經驗)，或是利用軟體編譯器執行單步除錯。

## SIMLAB 8051 設定、編譯、除錯

- 我們在程式編譯好後，上面功能列上開啓檢視各項資源→整合檢視

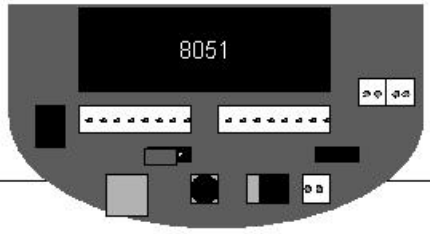


# SIMLAB 8051 設定、編譯、除錯

- 我們有設定產生高階組譯檔，故可同時看到C語言與組合語言，在們們要檢視的地方設定中斷點，並利用快速執行執行致中斷點，判斷是否有達成所需動作來判斷程式可能是哪邊出了問題。


# 燒錄方式

- 當我們程式編譯完後，將在程式資料夾內，產生.Hex檔(燒錄檔)，我們可以利用此晶片的燒錄軟體，透過USB介面燒錄之IC內執行，我們先將電腦鼠電源關閉，把JP1的短路器短路至靠近USB接頭的那端，並接上USB線到電腦，之後開啓Nuvoton 8051 ISP燒錄軟體。




# 燒錄方式

- 依照圖順序設定並燒錄，即可完成燒錄


# C51程式語言

# C語言大綱

- C51語言簡介
- 變數宣告與資料形態
- 算術與邏輯運算
- 迴圈與流程控制
- 函式
- 前置處理器

# C語言的特性

- 可以像組合語言一樣進行硬體之直接存取.
- 語法規則簡單,清楚,容易使用之結構化語言.
- 必要時可以與組合語言連結.
- 可攜性(Portability)極佳,跨平台的能力強

# C51程式語言的基本結構

```
#include <reg51.h>      ----> 前置處理器
main( )                 ----> 主程式
{
      int i=0 ;         /* 變數宣告 */
      i=i+1;            /* 算數運算 */----> 程式主體
      P1=i ;            /* 輸出至 Port 1 */
      ..
      ..
}
subroutine( )           ----> 副程式
..
..
subroutine( )           ----> 副程式
..
..
```

# ANSI & C51 關鍵字

| auto | break | case | char | Const | default |
|---|---|---|---|---|---|
| do | double | else | enum | extern | float |
| for | goto | int | long | register | return |
| short | signed | sizeof | Static | struct | switch |
| typedef | union | unsigned | void | volatile | while |

| _at_ | alien | bdata | Bit | code |
|---|---|---|---|---|
| compact | data | idata | interrupt | large |
| pdata | _priority_ | sbit | sfr | sfr16 |
| small | _task_ | using | xdata | |

# C51程式語言的敘述

- 資料宣告：
  - 利用常數、變數與各種結構的資料宣告，替代組合語言中繁瑣的資料搬移與定址方式。
- 算術邏輯運算：
  - 利用簡單的運算符號，提供加、減、乘、除、與AND、OR、NOT、XOR...等運算。
- 程式流程控制：
  - 提供更簡便的迴圈、條件判斷..等的程式流程控制功能。
- 函式呼叫：
  - 過函式呼叫功能，提供具有結構化的程式。

# C51的基本資料型態

| 符號 | 位元組 | 說明 |
|---|---|---|
| char | 1 | 字元型態,帶正負號字元. 範圍 -128~127 |
| Unsigned char | 1 | 字元型態,無正負號字元. 範圍 0~255 |
| Short | 1 | 短整數型態,範圍-32768~32767 |
| Unsigned short | 1 | 無號短整數型態,範圍0~65535 |
| Int | 2 | 整數型態,範圍-32768~32767 |
| Unsigned int | 2 | 無號整數型態,範圍0~65535 |
| long | 4 | unsigned long,float..... 爲4個bytes組成 |

# C51資料宣告範例

```
#include <reg51.h>
void main()
{
    /* 無號數的宣告   */
    unsigned   char      usg_var1=1;    //  宣告爲無號數字元資料型態
    unsigned   int       usg_var2=2;    //  宣告爲無號數整數資料型態
    unsigned   short     usg_var3=3;    //  宣告爲無號數短整數資料型態
    unsigned   long      usg_var4=4;    //  宣告爲無號數長整數資料型態

    /* 有號數的宣告   */
    char       sg_var1=-1;       //  宣告爲有號數字元資料型態
    int        sg_vars2=-2;      //  宣告爲有號數整數資料型態
    short      sg_vars3=-3;      //  宣告爲有號數短整數資料型態
    long       sg_vars4=-4;      //  宣告爲有號數長整數資料型態
}
```

# reg51.h表頭檔

- /* BYTE Register */
- sfr P0  = 0x80;
- sfr P1  = 0x90;
- sfr P2  = 0xA0;
- sfr P3  = 0xB0;
- sfr PSW = 0xD0;
- sfr ACC = 0xE0;
- sfr B   = 0xF0;
- sfr SP  = 0x81;
- sfr DPL = 0x82;
- sfr DPH = 0x83;
- sfr PCON = 0x87;
- sfr TCON = 0x88;
- sfr TMOD = 0x89;

# 變數的有效範圍

- **區域變數**:在函式的大括號內”{}”宣告的變數，有效範圍僅止於括號內部，離開此函式，變數失去作用，佔有記憶體會被釋放。不同括號內的程式碼也不允許存取
- **靜態變數**：在宣告前加入static關鍵字，靜態變數宣告讓系統保留變數內容
- **全域變數**：定義在所有大括號之外，通常在程式最前面，特點是能在任何函式中被使用，一直要到程式結束後才會被釋放
- **外部變數**：使用 extern關鍵字來連接這個變數，extern宣告時並不會建立一個變數，它只會到其它模組去找尋同名的變數以連結

# 靜態變數

靜態變數與自動變數一樣，是某特定函數內的區域性變數，但靜態變數的值不會因函數的執行結束而消失

**靜態變數的宣告如下所示：**

```
{
 static int a;
 static int b=54321;
 static char c;
 static float d=54.32;
          .
          .
}
```

**Example 1:**

```
main()
{
      increment();
      increment();
      increment();
}
increment()
{
    int x=0;
    x=x+1;
    P0=x;
}
```

Result P0= 1

**Example 2:**

```
main()
{
   increment();
   increment();
   increment();
}
increment()
{
   static int x=0;
   x=x+1;
   P0=x;
}
```

Result P0= 1,2,3

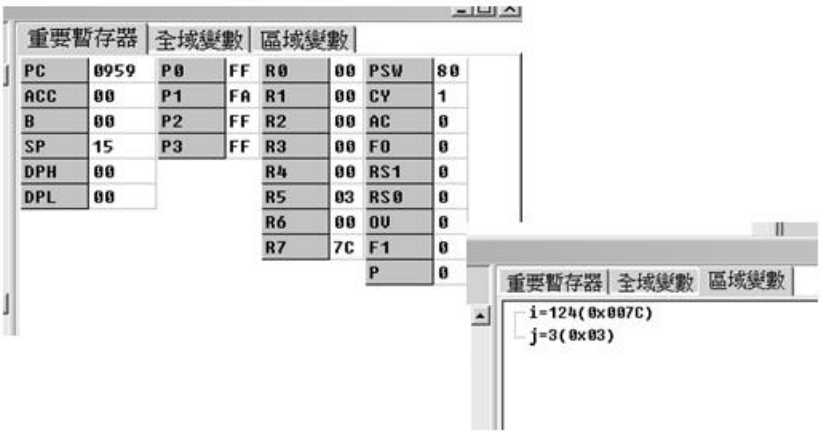
# 資料宣告指定記憶體種類

| 記憶體型態 | 範圍 | 說明 |
|---|---|---|
| **register** | R0~R7 | 暫存器 |
| **code** | C:0x0000~C:0xFFFF | 程式記憶體 |
| **data** | D:0x00~D:0x7F | 128Byte 的內部記憶體(可直接定址) |
| **idata** | 0x80~0xFF | 可間接定址記憶體區 |
| **bdata** | 0x20~0x2F | 可位元定址的內部記憶體 |
| **xdata** | <64KB | 外部記憶體 |
| **pdata** | <256 Byte | 頁外記憶體 |

# 資料宣告指定記憶體種類

| 符號 | 位元(bit) | 說明 |
|---|---|---|
| Bit | 1 | 1位元(0,1)的變數型態 |
| Sbit | 1 | 存取1位元可位元定址區(20H~2FH)或sfr |
| Sfr | 8 | 一次存取8位元特殊暫存器 |
| Sfr16 | 16 | 一次存取16位元特殊暫存器 |

# 資料宣告指定記憶體種類-Simlab




# 算術與邏輯運算

| 運算 | 符號 | 表示式 | 說明 |
|---|---|---|---|
| 加 | + | a+b | a+b 的加法運算 |
| 減 | - | a-b | a-b 的減法運算 |
| 乘 | * | a*b | a*b 的乘法運算 |
| 除 | / | a/b | a/b 的除法運算 |
| 餘數 | % | a%b | a 除 b 的餘數法運算 |
| 遞增 | ++ | a++ | a+1 的遞增運算 |
| 遞減 | -- | a-- | a-1 的遞減運算 |

# 算術運算實例

```
#include <reg51.h>
main()
{
        unsigned int a=48,b=0x23;
        unsigned int c,d,e,f,g;
        c=a+b;
        d=a-b;
        e=a*b;
        f=a/b;
        g=a%b;
        a++;
        b--;
}
```

answer

a=48 b=35

c=83

d=13

e=1680

f=1

g=13

a=49

b=34

# 關係運算子

| 運算 | 符號 | 表示式 | 說明 |
|---|---|---|---|
| 小於 | < | a<b | 判斷 a 是否小於 b，成立爲 1 |
| 大於 | > | a>b | 判斷 a 是否大於 b，成立爲 1 |
| 小於等於 | <= | a<=b | 判斷 a 是否小於等於 b，成立爲 1 |
| 大於等於 | >= | a>=b | 判斷 a 是否大於等於 b，成立爲 1 |
| 等於 | == | a==b | 判斷 a 是否等於 b，成立爲 1 |
| 不等於 | != | a!=b | 判斷 a 是否不等於 b，成立爲 1 |

# 關係運算實例

```
#include <reg51.h>

main()
{
        int a=0xc3,b=0x1f;
        int c,d,e,f,g,h;
        c=(a>b);
        d=(a<b);
        e=(a==b);
        f=(a<=b);
        g=(a>=b);
        h=(a!=b);
}
```

Answer

c=1

d=0

e=0

f=0

g=1

h=1

# 邏輯運算子

| 運算 | 符號 | 表示式 | 說明 |
|---|---|---|---|
| AND | && | a&&b | 若 ab 兩數都非零，得出結果 1，否則爲 0 |
| OR | \|\| | a\|\|b | 若 ab 兩數有一數非零，得出結果 1，否則爲 0 |
| NOT | ! | !a | 置於 a 數前方!表示 a 數反相 |

# 邏輯運算實例

```
#include <reg51.h>
void main()
{
    int a=0x42,b=0x51;
    int c,d,e;
    c=a&&b;
    d=a||b;
    e=!c;
}
```

```
0x42 =0100 0010
0x51 =0101 0001
a&&b=0100 0000
A||b  =0101 0011
 e    =1011 1111
```

# 位元運算子

| 運算 | 符號 | 表示式 | 說明 |
|---|---|---|---|
| AND | & | a&b | a 與 b 兩變數的相對位元做 AND 運算 |
| OR | \| | A\|b | a 與 b 兩變數的相對位元做 OR 運算 |
| XOR | ^ | a^b | a 與 b 兩變數的相對位元做 XOR 運算 |
| 補數 | ~ | a~b | a 變數的每一位元做反相運算 |
| 右移 | >> | a>>n | a 變數內位元右移 n 次(n=1,2,3…) |
| 左移 | << | a<<n | a 變數內位元左移 n 次(n=1,2,3…) |

# 位元運算實例

```
#include <reg51.h>

void main()
{
        P0=0x34;            /* 設定 P0=00110100 */
        P0=P0 & 0x81;       /* P0 埠做 AND 位元運算 */
        P0=P0 | 0x50;       /* P0 埠做 OR 位元運算 */
        P0=P0 ^ 0x0F;       /* P0 埠做 XOR 位元運算 */
        P0=~P0 ;            /* P0 埠做反相位元運算 */
        P0=P0 >>2;          /* P0 埠做右移兩個位元運算 */
        P0=P0 >>1;          /* P0 埠做左移一個位元運算 */
}
```

# 指定運算子

| 特殊運算式 | 基本運算式 |
|---|---|
| a += b | a = a + b |
| a -= b | a = a - b |
| a *= b | a = a * b |
| a /= b | a = a / b |
| a %= b | a = a % b |

# 指定運算子的應用實例

```
#include <reg51.h>

main()
{
        int a=35,b=0x30;
        a+=b;
        a-=b;
        a*=b;
        a/=b;
        a%=b;
        a|=b;
        a^=b;
}
```

| | |
|---|---|
| a=35=0x23 | |
| b=48=0x30 | |
| a=a+b | 83 |
| a=a-b | 35 |
| a=a*b | 1680 |
| a=a%b | 35 |
| a= a\|b | 51 |
| a=a^b | 3 |

# 基本運算子優先順序表

| | |
|---|---|
| !, 負號 (-), ++, -- | 優先 |
| 乘 ( * )、除 ( / )、餘數( % ) | |
| 加 ( +)、減 ( - ) | |
| <, <=, >, >= | |
| ==, != | |
| && | |
| \|\| | 最低 |

# C51程式流程控制

## C51提供三類的程式流程控制指令

- 迴圈控制：for、while、do-while。
- 條件分支控制： if-else、switch-case。
- 無條件躍遷： goto。

# 迴路 for 的運用

- **for** 允許使用者在同一行敘述中就設定了迴路的三個部份。
- (1)計數器的初值，(2)關係運算元，(3)增減迴路計數器的值。其語法如下所示：

```
for ( 計數器初值；關係運算元；計數器值更新 )
{
        statement 1 ;
              |
        statement n ;
}
```

# 迴路 while 的運用

- 在 while 的敘述中，當關係運算元之條件為真時，會不斷地重覆執行位於 while 後所列的敘述，直到條件變為否定才停止。

```
while ( 關係運算元 )
{
        statement 1 ;
              |
        statement n ;
}
```

# 迴路 do-while 的運用

- do-while 迴路，由於測試條件在迴路的後面，所以迴路中的敘述至少會被執行一次。

```
do
{
        statement 1 ;
        statement 2 ;
              |
        statement n ;
} while ( 關係運算元 ) ;
```

# 迴圈應用實例

```
void main(){
      int i,sum1=0;              // for      迴圈用變數
      int j=1,sum2=0;           // while  迴圈用變數
      int k=1,sum3=0;          // do while  迴圈用變數
   /* for  迴圈敘述計算  1 加到 10 的總和  */
     for(i=1;i<=10;i++)
      sum1+=i;
   /* while  迴圈敘述計算  1 加到 10 的總和  */
      while(j<=10){
         sum2+=j;
         j++;
      }
  /* do-while  迴圈敘述計算  1 加到 10 的總和  */
      do {
         sum3+=k;
         k++;
      }
      while(k<=10);
}
```

# 條件執行 if 的運用

- **if_then_else** 流程控制。

```
if (關係運算元)
{
        statement 1 ;
        statement 2 ;
            |
        statement n ;
}
else
        statement ;
```

# 簡單條件分歧應用實例

```
#include <reg51.h>
void main()
{
      while(1){
               if(P1==0x01)
                        P0=0xF0;          // 若 P1=0x01 則 P0=0xF0
               else
                        P0=0;             // 若 P1≠0x01 則 P0=0
               }
}
```

# 多重條件-使用if-else if-else

```
#include <reg51.h>
   void main()
   {     /* if - else if 判斷P1輸入狀態,決定P0輸出結果 */
         P1=0x00;
         for(;;) {
                  if(P1&0x01)
                           P0=0x80;
                  else if (P1&0x02)
                           P0=0xC0;
                  else if (P1&0x04)
                           P0=0xE0;
                  else if (P1&0x08)
                           P0=0xF0;
               }
   }
```

# 條件執行 switch 的運用

- 我們在實際的程式寫作時常會遇到多種選擇情況，而使用一連串 if - else 來表示是常發生的，所以C提供了一項特殊的控制結構，讓我們能夠有效且精簡處理程式。

```
switch ( 變數)
        {
                case 常數 1 :     statement 1 ;
                case 常數 2 :     statement 2 ;
                                                  |
                case 常數 n :     statement n ;
                default :         statement ;
        }
```

# 多重條件-使用switch

```
#nclude <reg51.h>
void main()
{       /* switch case 判斷P1輸入狀態,決定P0輸出結果 */
    P1=0x00;
    while(1){       switch(P1){
                            case 0x01 :
                                    P0=0x80; break;
                            case 0x02 :
                                    P0=0xC0; break;
                            case 0x04 :
                                    P0=0xE0; break;
                            case 0x08 :
                                     P0=0xF0; break;
                               }
          }
}
```

# goto 的運用

- 在一個程式中，使用 goto 敘述可以強制改變程式執行的步驟，但也會因此使程式的結構混亂，所以此敘述應儘量不用。其語法如下所述。

**Syntax :**

```
goto label ;

label :
        statement n ;
```

# C51-函式

- 何謂函數?
  - 1. C語言使用的函數可以寫出非常漂亮的程式結構，使程式簡單化，偵錯容易。
  - 2. 將重複之某些指令撰寫成一個函數，可減少編輯程式時間，更可使程式精簡，清晰了解。
  - 3. C語言使用的函數其呼叫方法與數學上使用函數完全相同，。

# 函數的結構(格式)

- **1. 函數的定義 (Function Definition)**
- **2. 呼叫函數 (Function Call)**
- **3. 函數原形宣告 (Function Prototype),**

# 函數的定義 (格式)

資料型別 函數名稱 (形式引數的串列)

{

形式引數的宣告 ;

函數的本體(執行敘述)

}

```
void Line (void)
{
    int j;
    for (j=1;j<30;j++)
    printf("%d\n",j);

}
```

# 函數的呼叫

Extern 的使用方式

Example1: file1.c

```
void func1( )
{
        .
        .
}
void func2( )
{
        .
}
```

Example1: file2.c

```
extern func1( );
void func4( );
void main( )
{
        .
    func1( );
    func4( );
}
void func4( )
{
    .......
}
```

# 函數的宣告

函數的宣告爲使用函數(函數呼叫)前的宣告，主要目的是提供給程式設計者參考及使用，亦可讓編譯系統(compiler)預知該名稱爲一函數名稱，因此可做較快速編譯。

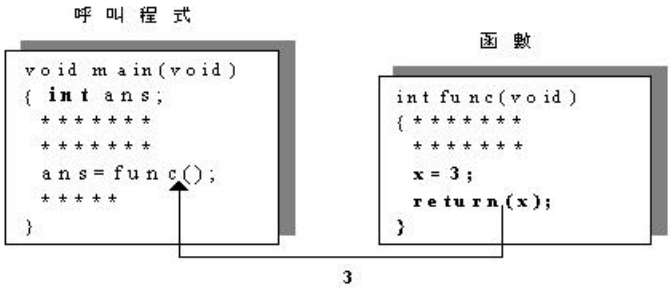
通常函數的宣告都是因該函數的定義不在函數呼叫所在地的檔案之前。但是，如果函數的定義在函數呼叫所在地的檔案之前，那麼我們就不需要函數的宣告。

函數 f 定義在呼叫處 printf(..., f(x) ) 之前，因此在 main() 裡面就不需要宣告函數 f。
其語法如下：
資料型　函數名(資料型參數名[,資料型參數名]);

```
char f1(char);
int  f2(int);                    /* 函數 f( ) 的宣告 */
```

# 函數的傳回值與資料形態

要求函數送回傳回值(函數值),必須在函數本體中利用return敘述進行,例如:




**return()**敘述之目的如下
1. 將控制權傳回給呼叫程式
2. 將return()敘述後括號內之數值傳給呼叫程式之方式

# 值的傳遞方法

(1)傳遞變數值本身(Call by value)
(2)傳遞變數的位址(Call by reference)

**Example 1: 傳值呼叫**

```
#include < stdio.h >
int func(a,b,c)
int a,b,c;
{
   int d;  d=a+b+c;
  return(d);
}
void main(void)
{      int x,y,z,p;
       x=y=z=3;
       p=func(x,y,z);
       printf("p-->%d\n",p);
}   result:  p-->9
```

**Example 2: 傳址呼叫**

```
#include < stdio.h >
void func(a,b,c,d)
int *a,*b,*c,*d;
{
    *d=*a+*b+*c;
}
void main(void)
{
    int x,y,z,p;
   x=y=z=3;
   func(&x,&y,&z,&p);
   printf("p-->%d\n",p);
}  result:   p-->9
```

# 基本函式範例-延遲副程式

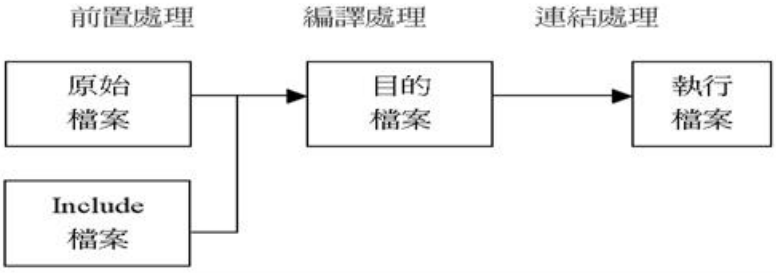
```
void delay10ms(int count)
{
     int i,j;
     for(i=0;i<count;i++)
          for(j=0;j<1940;j++);
}
main()
{
   delay10ms(100);
}
```

# 傳回值的函式範例

```
#include <reg51.h>
char CHK_PORT(char inp);
void main(){
    P0=0x00;
    P1=0x00;
    while(1) {
      P0=CHK_PORT(P1);
}}
char CHK_PORT(char inp){
     char temp_out;
     switch(inp){
          case 0x01 :   temp_out=0x80; break;
          case 0x02 :   temp_out=0xC0; break;
          case 0x04 :   temp_out=0xE0; break;
          case 0x08 :   temp_out=0xF0; break;
          default:      temp_out=0;
       }
       return temp_out;
}
```

# 前置處理器 (preprocessor)

我們在寫程式的時候,常常會碰到某些資料重覆使用或是某些程式片段因特定的條件下須放棄,如果以正常的方式來寫作程式,可能會增加無謂的程式片段,使的程式笨重不堪,但如果能好好運用C語言所提供的前置處理器,則上述的狀況將改善許多.這個功能並非C語言本身的式,但仍爲編譯程式所能接受.對前置處理器的解釋,如圖所示,因爲它是在程式編譯之前執行,所以顧名思義稱之前置處理.




# 前置處理器格式與種類

<1> 前置處理器的格式如下：
#前置處理器名稱　處理內容
<2> 前置處理器敘述可寫在程式中的任何地方 .
<3> 前置處理器的種類如下表一所列,其各種前置處理器之使用方法,我們將在以下的投影片中一一介紹 .

| 功能 | 前置處理器之敘述 |
|---|---|
| 檔案的含入 | #include |
| 字串的置換/巨集定義 | #define / #undef |
| 條件編譯 | #if .... #elif .... #else .... #endif<br>#ifdef (#ifndef) .... #else .... #endif |

## #include（檔案的含入）

此敘述之主要目的是讓我們將某個程式檔或標頭檔包括在目前的程式內,使目前的程式可引用該檔內的資料或程式.語法如下:

<1> #include "檔案名稱"
此表示系統將會到目前的目錄(路徑)下尋找所指定的檔案.

<2> #include <檔案名稱>
此表示系統將會到系統設定的目錄底下尋找所指定的檔案.

## #include（ Example ）

# #include（ Example ）

**Example:**

```
#include < stdio.h >
#include "def.h"

main()
{
    printf("PI=%2.5f\n",PI);
    printf("a+b=%d\n",a+b);

}
```

-- def.h 的檔案內容 -

```
#define PI  3.14159
#define a   10
#define b   20
```

---

執行結果 :

```
PI=3.14159
a+b=30
```

# #define(字串的置換/巨集定義) #undef

<1> 巨集 #define 的使用 :

其主要之功能是以簡單的名稱取代某些特定的常數或字串,函數,建立更簡單更易懂的程式.語法如下:

#define 巨集名稱 常數或函式或字串

<2> 巨集 #undef 的使用 :

其主要之功能則是取消最接近之#define所定義的敘述 .

#undef 巨集名稱

# #define/#undef (Example)

**Example1:**

```
#include < stdio.h >
#define PI  3.14159
main()
{
        printf("PI=%2.5f",PI);
}
```

執行結果 :

PI=3.14159

Example2 :

```
#include < stdio.h >
#define add(a,b) a+b
void main(void)
{
       int i,j,k;
       i=2;
       j=3;
       k=add(i,j);

       printf("i+j=%d\n",k);
}
```

執行結果 :  i+j=5

# 條件式的編譯

**<1>** 條件式的編譯,此用法的最大特色在於其敘述不僅可以放在程式的頂端,亦可將其敘述放置於程式中.以下所列(1),(2),(3)項爲條件式編譯的成員.

**(1) #if 和 #endif**

**(2) #else / #elif**

**(3) #ifdef 和 #ifndef**

**<2>** 由以上的敘述,我們不難發現這些條件式編譯的成員與前幾章控制流程所提到 **if ... Then ... else** 的語法十分相似,且連使用方法與原理都非常相似.

# 條件式的編譯

**Example1:**

```
#include < stdio.h > #define
value  99
void main(void)
{
    #if value < 100
      printf("value < 100");
    #else
      printf("value >= 100");
     #endif
}
```

**執行結果:**

```
value < 100
```

Example2 :

```
#include < stdio.h >
#define value   100
void main(void)
{
 #if value  < 100
    printf("value < 100");
 #elif value  > 100
    printf("value > 100");
  #else
    printf("value = 100");
  #endif  }
```

**執行結果:**

```
value  = 100
```

# C語言標準函式庫

- C-1 標準輸出輸入函數<stdio.h>
- C -2 字元檢查函數<ctype.h>
- C -3 字串函數<string.h>
- C -4 數學函數<math.h>
- C -5 日期/時間函數<time.h>
- C -6 工具函數<stdlib.h>

# C-1 標準輸出輸入函數<stdio.h>

- FILE* fopen(const char* filename, const char* mode)：使用mode 模式開啓參數filename 的檔案，傳回檔案串流，失敗傳回NULL。
- FILE* freopen(const char* filename, const char* mode, FILE* stream) ：關閉檔案後重新開啓檔案。
- int fflush(FILE* stream)：清除緩衝區的內容，成功傳回0，失敗傳回 EOF。
- int fclose(FILE* stream)：關閉檔案。
- int remove(const char* filename)：刪除參數的檔案，失敗傳回非零值。
- int rename(const char* oldname, const char* newname)：將檔案名稱 oldname 改爲newname，失敗傳回非零值。
- FILE* tmpfile()：建立"wb+"模式的暫存檔案，當結束程式後就會關閉且刪除此檔案。
- char* tmpname(char s[L_tmpnam])：指定暫存檔案的名稱爲s。

# C-1 標準輸出輸入函數<stdio.h>

- int setvbuf(FILE* stream, char* buf, int mode, size_t size)：指定串流暫存區尺寸size，使用mode參數值_IOFBF 爲完整暫存區、_IOLBF 是線性暫存區或_IONBF 沒有暫存區。
- void setbuf(FILE* stream, char* buf)：指定串流的暫存區爲參數buf。
- int fprintf(FILE* stream, const char* format, ...)：將格式化字串寫入檔案串流。
- int printf(const char* format, ...)：在標準輸出顯示格式化字串。
- int sprintf(char* s, const char* format, ...)：將格式化字串輸出到字串 s。
- int fscanf(FILE* stream, const char* format, ...)：從檔案串流讀取指定格式的資料。
- int scanf(const char* format, ...)：從標準輸入讀取指定格式的資料。
- int sscanf(char* s, const char* format, ...)：從字串s 讀取指定格式的資料。
- int fgetc(FILE* stream)：從檔案串流讀取一個字元。

# C-1 標準輸出輸入函數<stdio.h>

- char* fgets(char* s, int n, FILE* stream)：從檔案串流讀取一個字串。
- int fputc(int c, FILE* stream)：寫入一個字元到檔案。
- char* fputs(const char* s, FILE* stream)：寫入一個字串到檔案。
- int getc(FILE* stream)：從檔案串流讀取一個字元。
- int getchar(void)：從標準輸入讀取一個字元。
- char* gets(char* s)：從標準輸入讀取一個字串。
- int putc(int c, FILE* stream)：寫入一個字元到檔案。
- int putchar(int c)：在標準輸出顯示一個字元。
- int puts(const char* s)：在標準輸出顯示一個字串。
- int ungetc(int c, FILE* stream)：將一個字元放回檔案串流。
- size_t fread(void* ptr, size_t size, size_t nobj, FILE* stream)：從檔案讀取指定大小的資料。

# C-1 標準輸出輸入函數<stdio.h>

- size_t fwrite(const void* ptr, size_t size, size_t nobj, FILE* stream)：將指定大小的資料寫入檔案。
- int fseek(FILE* stream, long offset, int origin)：移動檔案指標到offset位移量， 其方向是origin 參數值SEEK_SET 的檔案開頭、SEEK_CUR 是目前位置或SEEK_END 檔尾。
- long ftell(FILE* stream)：目前檔案指標的位置。
- void rewind(FILE* stream)：重設檔案指標到檔頭。
- int feof(FILE* stream)：是否到達檔尾。
- int ferror(FILE* stream)：是否檔案串流產生錯誤。

# C-2 字元檢查函數<ctype.h>

- int isalnum(int c)：isalpha(c)或isdigit(c)的字元。
- int isalpha(int c)：isupper(c)或islower(c)的字元。
- int iscntrl(int c)：是否是ASCII 控制字元。
- int isdigit(int c)：是否是數字。
- int isgraph(int c)：是否是顯示字元，不含空白字元。
- int islower(int c)：是否是小寫字元。
- int isprint(int c)：是否是顯示字元0x20 (' ')到0x7E ('~')。
- int ispunct(int c)：是否是顯示字元，不包含空白、字母、數字字元。
- int isspace(int c)：是否是空白字元。
- int isupper(int c)：是否是大寫字元。
- int isxdigit(int c)：是否是十六進位字元。
- int tolower(int c)：轉換成小寫字元。
- int toupper(int c)：轉換成大寫字元。

# C-3 字串函數<string.h>

- char* strcpy(char* s, const char* ct)：將字串ct 複製到字串s。
- char* strncpy(char* s, const char* ct, size_t n)：將字串ct 前n 個字元複製到字串s。
- char* strcat(char* s, const char* ct)：連結字串ct到字串s 之後。
- char* strncat(char* s, const char* ct, size_t n)：連結字串ct 前n 個字元到字串s。
- int strcmp(const char* cs, const char* ct)：比較字串cs 和ct。
- int strncmp(const char* cs, const char* ct, size_t n)：比較字串cs 和ct的前n 個字元。
- char* strchr(const char* cs, int c)：傳回字元c 第一次出現在字串cs 位置的指標。
- char* strrchr(const char* cs, int c)：傳回字元c 第後一次出現在字串cs 位置的指標。

# C-3 字串函數<string.h>

- char* strpbrk(const char* cs, const char* ct)：傳回字串ct 任何字元在字串cs 第一次出現的位置指標。
- char* strstr(const char* cs, const char* ct)：傳回字串ct 在字串cs 第一次出現的位置指標。
- size_t strlen(const char* cs)：傳回字串cs 的長度。
- char* strerror(int n)：傳回指定錯誤代碼的說明文字內容。
- char* strtok(char* s, const char* t)：以字串t 的任何字元爲分隔字元，找尋字串s 中下一個token 記號。
- void* memcpy(void* s, const void* ct, size_t n)：從位置ct 複製n 個字元到位置s，傳回s。
- void* memmove(void* s, const void* ct, size_t n)：從位置ct 搬移n 個字元到位置s，傳回s。

# C-3 字串函數<string.h>

- int memcmp(const void* cs, const void* ct, size_t n)：比較位置ct 和位置cs 的前n 個字元。
- void* memchr(const void* cs, int c, size_t n)：傳回cs 位置開始前n 個字元第一次出現字元c 的位置指標。
- void* memset(vo id* s, int c, size_t n)：取代cs 位置開始前n 個字元成爲字元c，傳回位置指標s。

# C-4 數學函數<math.h>

- double exp(double x)：自然數的指數e^x。
- double log(double x)：自然對數logx
- double log10(double x)：十爲底的對數log10x。
- double pow(double x, double y)：傳回參數x 爲底，參數y 的次方值x^y。
- double sqrt(double x)：參數x 的平方根。
- double ceil(double x)：傳回大於或等於參數x 的最小double 整數。
- double floor(double x)：傳回小於或等於參數x 的最大double 整數。
- double fabs(double x)：傳回參數x 的絕對值。
- hypot(double x, double y)：傳回√(x^2+y^2)公式的值
- double ldexp(double x, int n)：x 乘以2 的n 次方是x*2^n。
- double frexp(double x, int* exp)：將參數x 的浮點數分解成尾數和指標，x = m*2^exp，傳回m值的尾數，將指數存入參數exp。

# C-4 數學函數<math.h>

- double modf(double x, double* ip)：將浮點數x 分解成整數和小數部分，傳回小數部分，將整數部分存入參數ip。
- double fmod(double x, double y)：如果y 爲非零值，傳回浮點數x/y 的餘數。
- double sin(double x)：正弦函數。
- double cos(double x)：餘弦函數。
- double tan(double x)：正切函數。
- double asin(double x)：反正弦函數。
- double acos(double x)：反餘弦函數。
- double atan(double x)：反正切函數。
- double atan2(double y, double x)：參數y/x 的反正切函數值。
- double sinh(double x)：hyperbolic 正弦函數，sinh(x)=(e^x-e^(-x))/2。
- double cosh(double x) ： hyperbolic 餘弦函數， cosh(x)=(e^x+e^(-x))/2。
- double tanh(double x) ： hyperbolic 正切函數， tanh(x)=(e^x-e^(-x))/(e^2+e^(-x))。

# C-5 日期/時間函數<time.h>

- clock_t clock(void)：傳回程式開始執行後所使用的CPU 時間，以 ticks 爲單位，除以常數CLK_TCK就是秒數。
- time_t time(time_t* tp)：傳回目前的曆法時間（Calendar Time），也會指定給參數的tp 指標，如爲無效時間，傳回-1。
- double difftime(time_t time2, time_t time1)：傳回參數time2 和time1 的時間差，即time2-time1。
- time_t mktime(struct tm* tp)：將參數*tp 的當地時間改爲曆法時間，如果不能轉換傳回-1。
- char* asctime(const struct tm* tp)：傳回參數tm 結構指標轉換成日期/時間格式的字串，字串最後有新行字元\n。
- char* ctime(const time_t* tp)：傳回參數time_t 指標轉換成當地日期/時間的字串，字串最後有新行字元\n。

# C-5 日期/時間函數<time.h>

- struct tm* gmtime(const time_t* tp)：傳回將參數的time_t 指標轉換成
- UTC（Coordinated Universal Time）日期/時間的tm 結構指標。
- struct tm* localtime(const time_t* tp)：傳回將參數的time_t 指標轉換成當地日期/時間的tm結構指標。
- size_t strftime(char* s, size_t smax, const char* fmt, const struct tm* tp)：將參數tp 的日期/時間以格式化字串fmt 輸出到字串s，s 最多儲存smax個字元。

# C-6 工具函數<stdlib.h>

- int abs(int n)、long labs(long n)：傳回整數n 的絕對值。
- double atof(const char* s)：將參數字串s 轉換成浮點數，如果字串不能轉換傳回0.0。
- int atoi(const char* s)：將參數字串s 轉換成整數，如果字串不能轉換傳回0。
- long atol(const char* s)：將參數字串s 轉換成長整數，如果字串不能轉換傳回0。
- double strtod(const char* s, char** endp)：函數忽略字串s 前的空白字元，將數字部分轉換成浮點數，如果尚有未轉換的部分字串，則設成參數endp 指標。
- long strtol(const char* s, char** endp, int base)：函數忽略字串s 前的
- 空白字元，將數字部分轉換成長整數，如果尚有未轉換的部分字
- 串，則設成參數endp 指標。

# C-6 工具函數<stdlib.h>

- unsigned long strtoul(const char* s, char** endp, int base)：如同strtol函數，其傳回值是無符號長整數。
- void* calloc(size_t nobj, size_t size)：傳回一塊參數nobj 陣列大小的記憶體指標，nobj 元素大小爲size 初值爲0，錯誤傳回NULL。
- void* malloc(size_t size)：傳回大小size 記憶體指標，沒有指定初值，錯誤傳回NULL。
- void* realloc(void* p, size_t size)：將指標p 的記憶體改爲size 大小，不會更改原記憶體的值，多配置部分初值爲0，錯誤傳回NULL。
- void free(void* p)：釋放參數p 指標的記憶體空間。
- void abort()：強迫程式以不正常方式結束，如同呼叫raise(SIGABRT)函數。
- void exit(int status)：程式以正常方式結束，傳回系統環境狀態值， 0 表示正常結束。

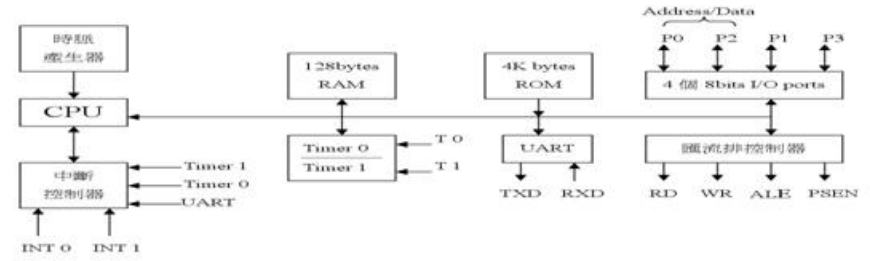
# C-6 工具函數<stdlib.h>

- int system(const char* s)：將字串s的指令傳給環境來執行，也就是執行MS-DOS的指令。
- char* getenv(const char* name)：傳回參數name 的環境字串，如果沒有傳回NULL。
- void* bsearch(const void* key, const void* base, size_t n, size_t size, int
- (*cmp)(const void* keyval, const void* datum))：陣列基礎的二元搜尋函數，陣列是參數base，鍵值是參數key，n 是陣列大小， size 是每個元素的大小，最後的參數是指向函數的指標，這是比較元素大小的函數，找到傳回該元素指標，沒有找到傳回NULL。
- void qsort(void* base, size_t n, size_t size, int (*cmp)(const void*, const void*))：陣列基礎的快速排序法函數，陣列是參數base，n 是陣列大小，size 是每個元素的大小，最後的參數是指向函數的指標，這是比較元素大小的函數。

# C-6 工具函數<stdlib.h>

- int rand(void)：傳回亂數的整數值，其值的範圍是0 到RAND_MAX 常數，其值爲0x7FFF。
- void srand(unsigned int seed)：指定亂數的種子數，參數是無符號整數，如果沒有指定，預設的種子數爲1。

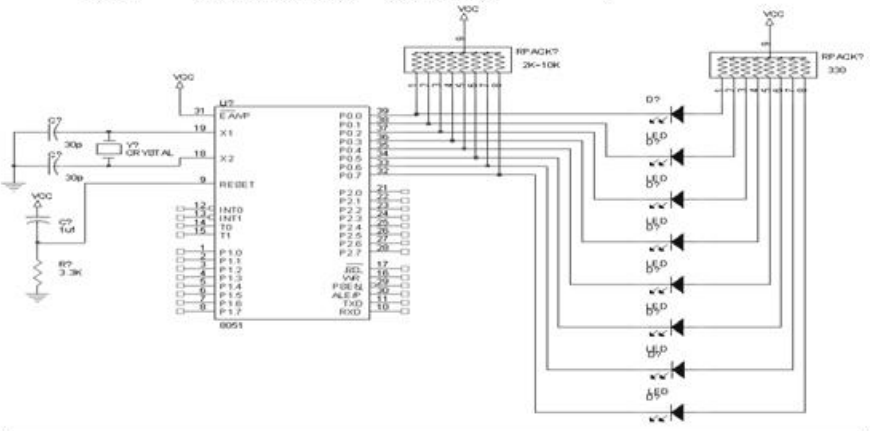
# 8051單片的內部結構及接腳





# 實習一

- 8個LED I/O埠控制電路實習
  - 兩個LED由左至右顯示：電路分析(LED81.c)



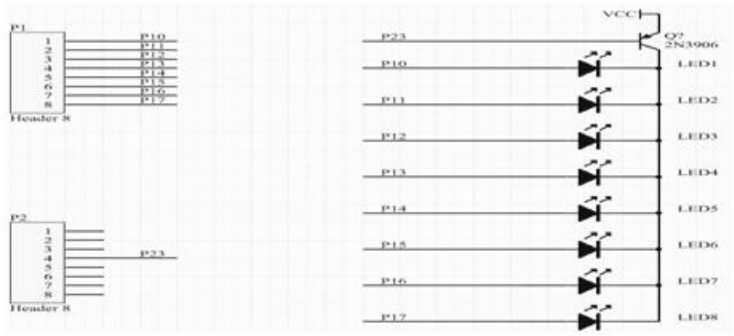
# 8個LED I/O埠共陽控制電路實習

```
//    P0所接的8個LED(初始狀態:最左2個亮；其餘6個LED熄著),並進行右旋的變化
//    程式分析              sbit Led_Com = 0xB3;          //int1 P3.3
      unsigned char tP0 ;
      Led_Com = 0;
      tP0 =0xc0;
      P0=~tP0 ;
      for(;;) {
        Delay100ms() ;
        if(tP0&0x01){                 //P0右旋1位元
              tP0 >>=1 ;
              tP0 |=0x80 ;            //P0.0==1時，P0.7補 ’ 1'
        }
        else{
              tP0 >>= 1 ;
        }
        P0=~tP0 ;
      }
```

# 8個LED I/O埠共陽控制電路實習

- 兩個LED由左至右顯示： 軟體模擬：

## 8個LED I/O埠共陽控制電路實習下載目標板



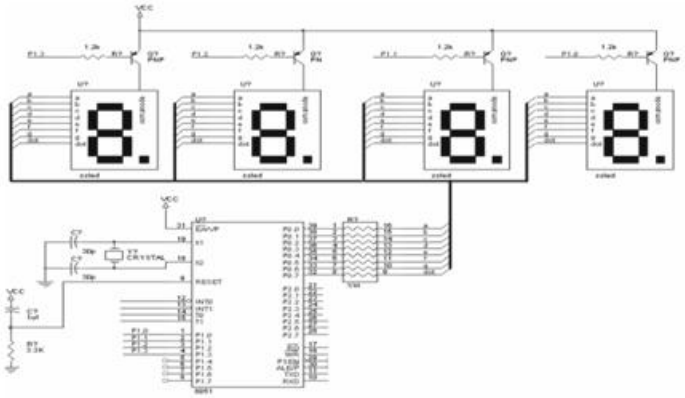

- sbit Led_Com = 0xB3;  //int1 P3.3
- Led_Com = 0;
- P0更改爲P1

## 8個LED I/O埠共陽控制電路實習

- 題目：剛剛爲兩顆LED由左至右顯示
- 目前修改爲兩顆LED由右至左顯示

# 實習二

- 4個七段式LED控制電路實習
  - 顯示數字“1234“： 電路分析(SSLED3.c)

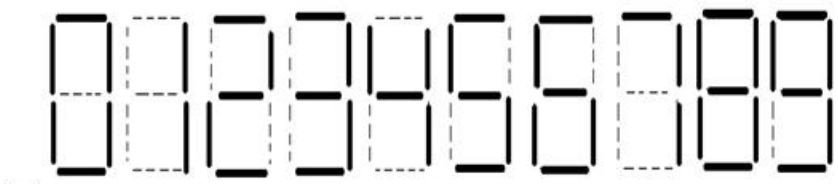


# 七段顯示器顯示分析

a：b0　　b：b1　　c：b2

d：b3　　e：b4　　f：b5

g：b6　　dp：b7



# 七段顯示器的編碼

- 0:共陽 C0H，共陰 3FH……

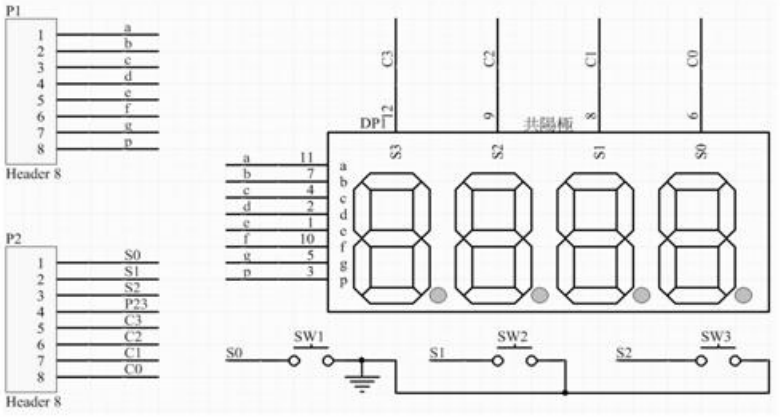
| 顯示 | Dp | g | f | e | d | c | b | a | HEX |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | C0H |
| | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 3FH |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | F9H |
| | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 06H |
| 2 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | A4H |
| | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5BH |
| 3 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | B0H |
| | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 4FH |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 99H |
| | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 66H |
| 5 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 92H |
| | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 6DH |
| 6 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 82H |
| | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 7DH |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | F8H |
| | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 07H |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80H |
| | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7FH |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 90H |
| | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 67H |

# 4個七段式LED控制電路實習

- 顯示數字“1234"，其掃瞄間隔爲100ms：　程式分析：
- unsigned char code sstbl[ ]= {
  0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90} ;
- unsigned char code loc[ ]={ 0xf7,0xfb,0xfd,0xfe} ;
- main()
- {
- 　　char i ;
- 　　i=0 ;
- 　　for( ; ;){
- 　　P1=0xff ;　　//關閉共陽
- 　　P0=sstbl[ i+1] ;
- 　　P1=loc[ i++] ;
- 　　if(i >=4){
- 　　i=0 ;
- 　　}
- 　　Delay100ms() ;
- 　　}
- }

# 4個七段式LED控制電路實習

- 顯示數字“1234“，其掃瞄間隔爲100ms：
  下載到目標板：




# 4個七段式LED控制電路實習

- 程式需修改部分
- P0更改爲P1
- P1更改爲P3 ，高低位元組交換

- 題目：剛剛爲顯示數字“1234“，其掃瞄間隔爲100ms
- 目前修改爲 顯示數字“1234“，其掃瞄間隔爲1ms

# 4個七段式LED控制電路實習

- 下載目標板修改後程式，顯示數字“1234"，其掃瞄間隔爲100ms：
- unsigned char code sstbl[ ]= {
  0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90} ;
  unsigned char code loc[]={ 0x7f,0xbf,0xdf,0xef} ;  //電腦鼠
- main()
- {
- char i ;
- i=0 ;
- for( ; ;){
- P3=0xff ;  //關閉共陽
- P1=sstbl[ i+1] ;
- P3=loc[ i++] ;
- if(i >=4){
- i=0 ;
- }
- Delay100ms() ;
- }
- }

# 實習三

- TIMER/ Inerrupt/ PWM 控制電路軟硬體模擬實習

# 中斷的功能

- 8051單晶片的中斷服務功能，可使中斷服務的需求以中斷的方式通知8501CPU，以使CPU獨立執行主程式，而提升執行效率。在8051單晶片中提供5個中斷源，分別爲：
  1. INT0：外部中斷，由8051單晶片第12接腳輸入。
  2. Timer0：計時/計數器中斷。
  3. INT1：外部中斷，由8051單晶片第13接腳輸入。
  4. Timer1：計時/計數器中斷。
  5. UART：串列埠中斷。

  產生中斷服務副程式的呼叫，跳到該中斷所對應之中斷向量位址去執行，CPU執行該中斷服務副程式，直到「RETI」指令後才結束中斷副程式

# MCS-51的中斷向量與中斷相關暫存器

| 中斷源 | 中斷向量(位址值) | 旗標 | 所屬暫存器 |
|---|---|---|---|
| INT0 | 0003H | IE0 | TCON.1 |
| Timer0 | 000BH | TF0 | TCON.5 |
| INT1 | 0013H | IE1 | TCON.3 |
| Timer1 | 001BH | TF1 | TCON.7 |
| UART(TXD) | 0023H | TI | SCON.1 |
| UART(RXD) | 0023H | RI | SCON.0 |

- 中斷源INT0與INT1分別位於8051單晶片接腳第12與13支，當此二接腳爲低電位(或〝0〞)時，則IE0與IE1會設定爲〝1〞；而當對應之中斷服務副程式執行完畢後，則8051會自動清除IE0與IE1旗標。Timer0與Timer1的中斷產生則如第六章所介紹，當計時/計數值產生溢位時，則對應之旗標TF0與TF1設定爲〝1〞；而當對應之中斷服務副程式執行完畢後，則8051會自動清除TF0與TF1旗標。UART爲串列埠中斷源，當串列埠做爲傳送或接收時其對應不同的旗標TI與RI，其使用方式將在下一章介紹，當其對應旗標設定爲〝1〞後，且中斷致能，則中斷服務副程式將會執行。當對應之中斷服務副程式執行完畢後，則8051會自動清除TI與RI旗標。

# 中斷致能暫存器(IE )

| EA | ---- | ---- | ES | ET1 | EX1 | ET0 | EX0 |
|---|---|---|---|---|---|---|---|

- 中斷致能暫存器　位址：A8H

- EA(IE.7)：若EA＝0，則禁止所有中斷；若EA＝1，則各中斷是否致能可由各自的中斷致能位元來各別設定。
- ----(IE.6)：未使用。
- ----(IE.5)：未使用，但於8052單晶片中則爲Timer2致能位元。
- ES(IE.4)：致能串列埠的中斷。
- ET1(IE.3)：致能Timer1的中斷。
- EX1(IE.2)：致能INT1的中斷。
- ET0(IE.1)：致能Timer0的中斷。
- EX0(IE.0)：致能INT0的中斷。

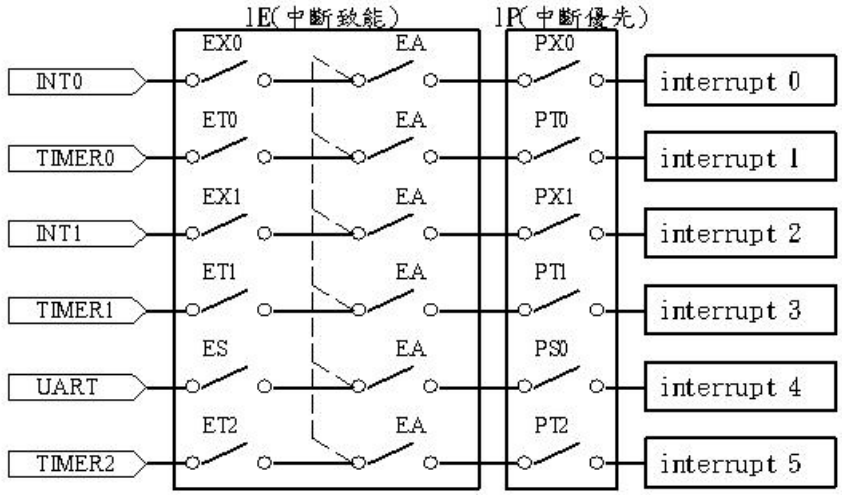
# 中斷優先暫存器

| ---- | ---- | ---- | PS | PT1 | PX1 | PT0 | PX0 |
|---|---|---|---|---|---|---|---|

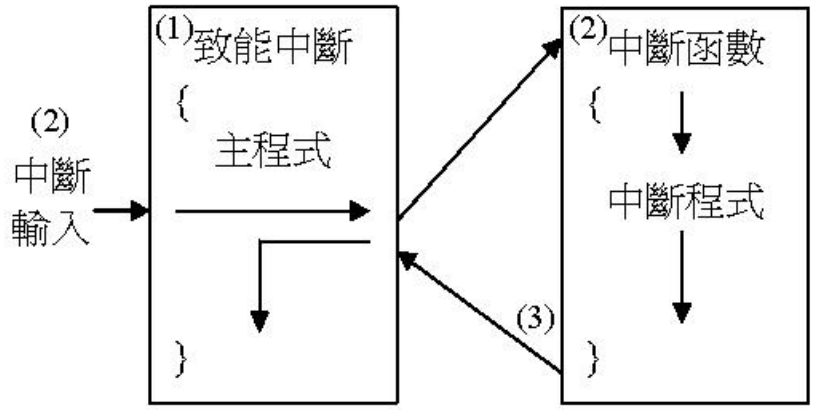
- 中斷優先暫存器　　位址：B8H

- ----(IP.7)：未使用。
- ----(IP.6)：未使用。
- ----(IP.5)：未使用，但在8052單晶片中，則爲Timer2的優先權位元。
- PS(IP.4)：定義串列埠優先權位元。
- PT1(IP.3)：定義Timer1優先權位元。
- PX1(IP.2)：定義INT1優先權位元。
- PT0(IP.1)：定義Timer0優先權位元。
- PX0(IP.0)：定義INT0優先權位元。

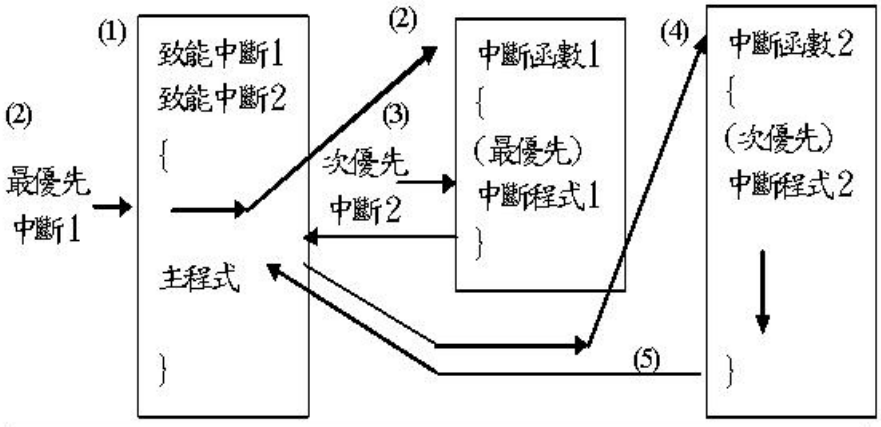
## 中斷致能暫存器與中斷優先暫存器 圖示



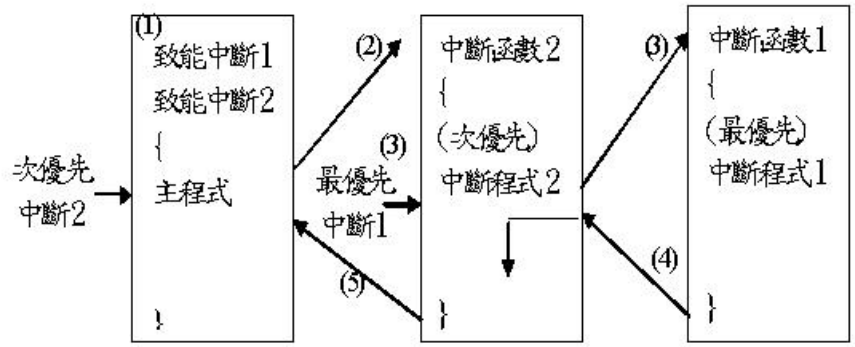

## 中斷：僅有一個中斷工作時



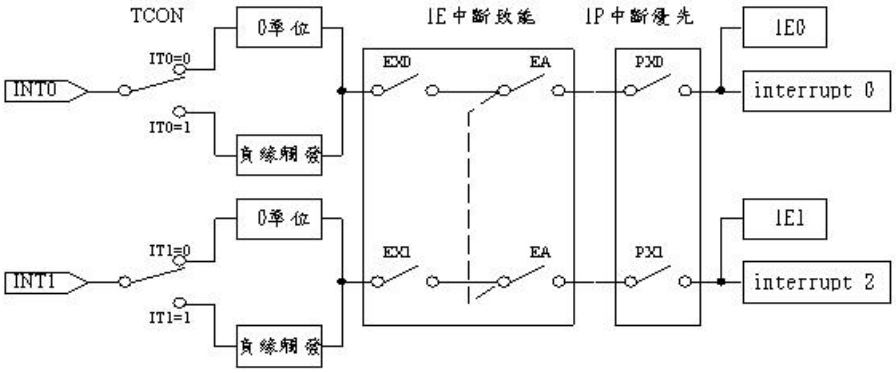
# 中斷：最優先中斷先輸入



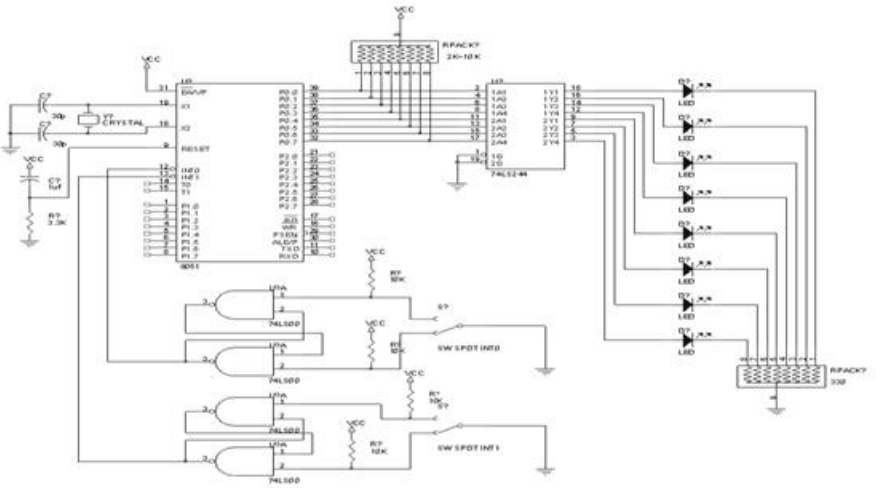

# 中斷：次優先中斷先輸入



# 中斷：外部中斷控制




# 中斷：外部中斷控制



# 中斷：外部中斷控制

```
char Count ;
main()
{
        P0=Count=0 ;
        IT0=1 ; // 偵測負緣觸發動作
        EA=1;
        EX0=1 ;
        for(;;) ;
}
void external0(void  ) interrupt 0
{
        Count++ ;
        P0=Count ;
}
```

# TIMER0-1計時計數功能

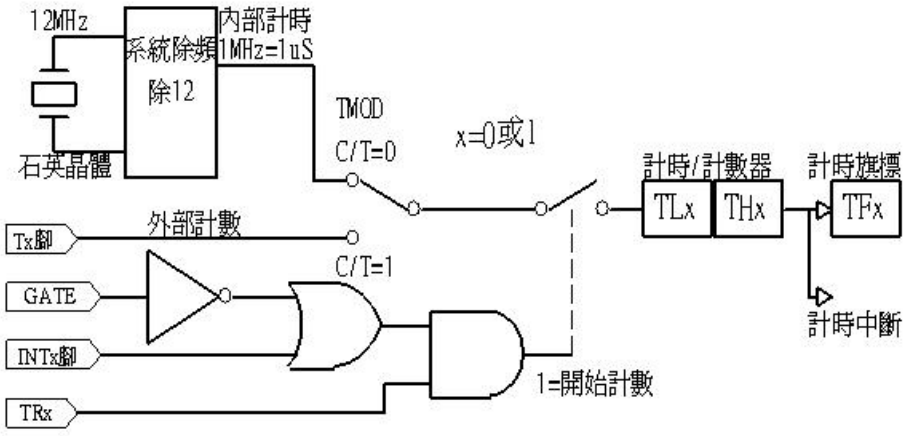
MCS-51具2~3個16位元的計時/計數器T0, T1(或T2), 可用指令規劃為計時器, 或計數器使用.

- 16位元的計時/計數器是由兩個8位元的暫存器所構成，包括有高位元組(THX)與高位元組(TLX)所組成
- 規劃成計時器, 可測量時間間隔(計時單位為石英振盪器頻率除以12。例如12MHZ -> 1微秒(us) )
- 規劃成計數器 ，可計數事件發生的次數，或計算週期性的脈波。T0或T1接腳輸入一個負緣訊號(1->0), T0或T1計數器會自動加1(最高計數頻率為石英振盪器頻率的1/24)

# TIMER0-1計時計數功能

- TCON：控制暫存器
- TMOD：模式暫存器
- TL0，TH0：Timer0值之暫存器
- TL1，TH1：Timer1值之暫存器
- IE：中斷致能暫存器

# TIMER0-1計時器方塊圖

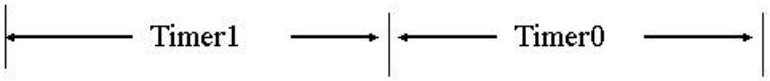


# 計時器觸發方式(TCON暫存器)

| TCON.7 | TCON.6 | TCON.5 | TCON.4 | TCON.3 | TCON.2 | TCON.1 | TCON.0 |
|---|---|---|---|---|---|---|---|
| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IR0 | IT0 |

| | |
|---|---|
| TF1 | 第1個計時計數溢位時設爲1,啓動計時中斷副程式才設回0 |
| TR1 | 第1個計時計數器的啓動位元 |
| TF0 | 第0個計時計數溢位時設爲1,啓動計時中斷副程式才設回0 |
| TR0 | 第0個計時計數器的啓動位元 |
| TCON.0~ TCON.3 | 外部中斷使用 |

# 計時器模組暫存器(TMOD)




| Gate | C/T | M1 | M0 | Gate | C/T | M1 | M0 |
|---|---|---|---|---|---|---|---|

| | | |
|---|---|---|
| Gate | | Gate=1, INTx與TRx=1計時器才動作<br>Gate=0, TRx=1計時器可動作 |
| C/T | | C/T=0當計時器 ; C/T=1當計數器 |
| M1 | M0 | |
| 0 | 0 | 模式0: 當13位元計時計數器 |
| 0 | 1 | 模式1: 當16位元計時計數器 |
| 1 | 0 | 模式2: 當8位元計時計數器 |
| 1 | 1 | 模式3: 當2個8位元計時計數器 |

# 8051計時計數器的四種工作模式

| M1 | M0 | 工作模式 | 功　能 |
|---|---|---|---|
| 0 | 0 | 模式 0 | 當 13 位元計時/計數器 |
| 0 | 1 | 模式 1: | 當 16 位元計時/計數器 |
| 1 | 0 | 模式 2: | 當 8 位元計時/計數器，自動載入 |
| 1 | 1 | 模式 3: | 當 2 個 8 位元計時計數器 |

工作模式的設定使用TMOD暫存器的M1, M0

# TMOD說明

- GATE = 1 需要 INTx 接腳爲高電位才計時，GATE = 0 則不需要。

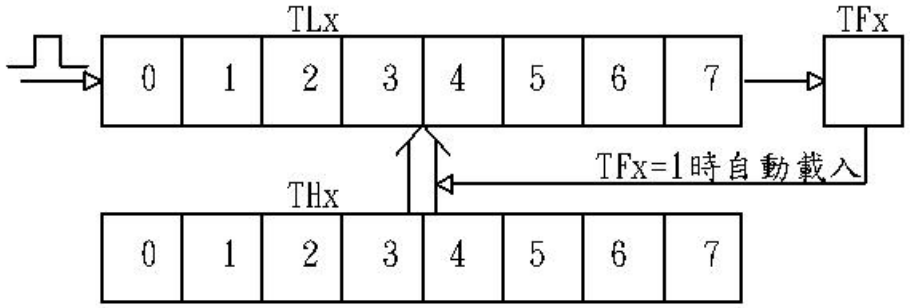
  也就是當 TRx=1 and GATE=0 開始計時，或者當 TRx=1 and GATE=1 and INTx=1 開始計時，其它條件都不計時。

- C/T = 0 使用內部時脈，C/T = 1 使用外部時脈。

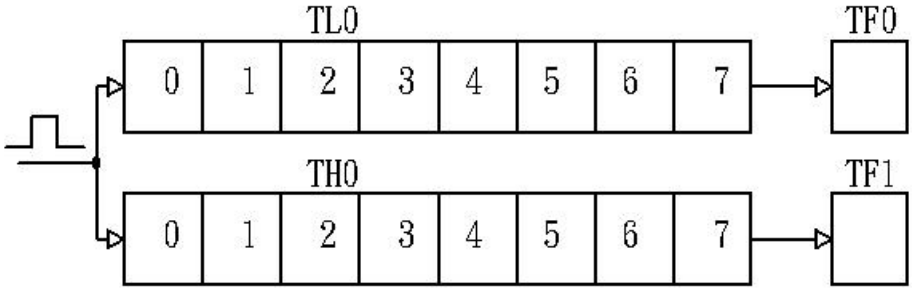
**Timer 的計時時脈來源有兩種**

- 一種是 8051 單晶片的內部時脈也就是從 XTAL1 與 XTAL2 接腳所輸入的內部時脈，一種是從 T0 與 T1 接腳所輸入的外部時脈。所以**一般來說設定成內部時脈時，稱爲計時器**，因爲振盪器頻率固定可以當計時器用；
- 而設定成**外部時脈時，稱爲計數器**，因爲不知使用者會接什麼裝置，但是都有計算次數功能。如果外部時脈也是接振盪器時，也可以算是計時器，只是沒人會如此做，因爲直接使用內部時脈即可，可以省成本又可以多些腳位使用。

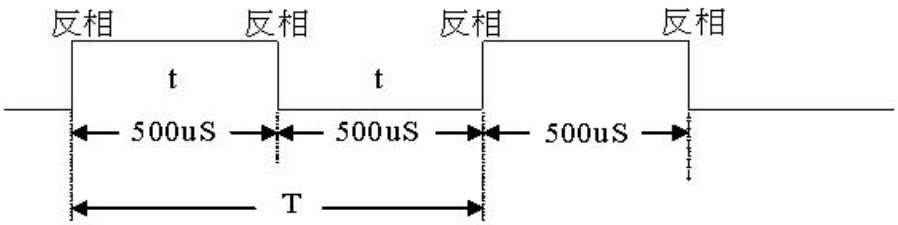
## Timer0-1：mode2自動載入方塊圖



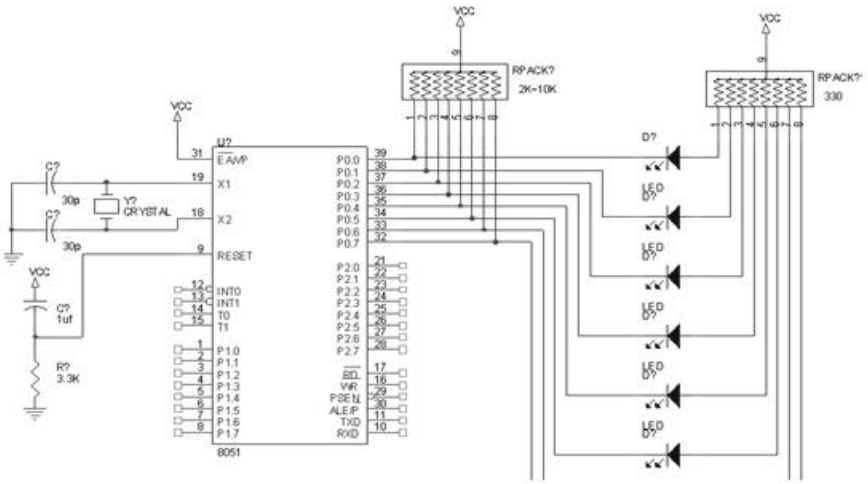

## Timer0：mode3方塊圖



# 輸出指定的頻率(聲音)




# Timer/Counter實習

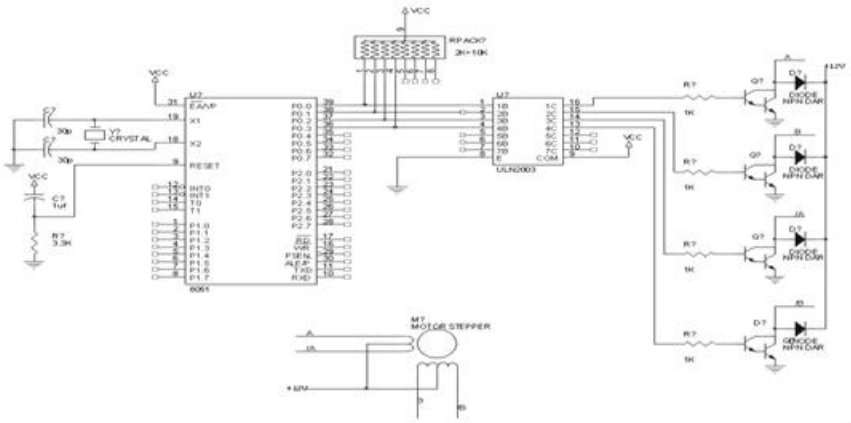


# Timer/Counter實習

```
char Count;
main()
{
        P0=Count=0;
        TMOD=0x10;
        TH0= (0xffff-10000)/256;
        TL0=(0xffff-10000)%256;
        EA=1;
        ET0=1;
        TR0=1;
        for(;;);
}
void timer0(void) interrupt 1
{
        Count++;
        P0=Count;
        TH0= (0xffff-10000)/256;
        TL0=(0xffff-10000)%256;
}
```

# 實習四

- 步進馬達控制電路實習
  - 電路分析( STEPM1B.C )

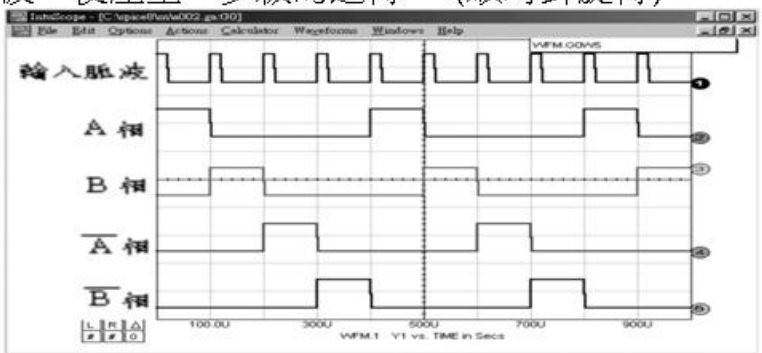


# 步進馬達的激磁方式

- 二相步進馬達的激磁方式有下列兩種:
- (1).全步激磁
- (2).半步激磁

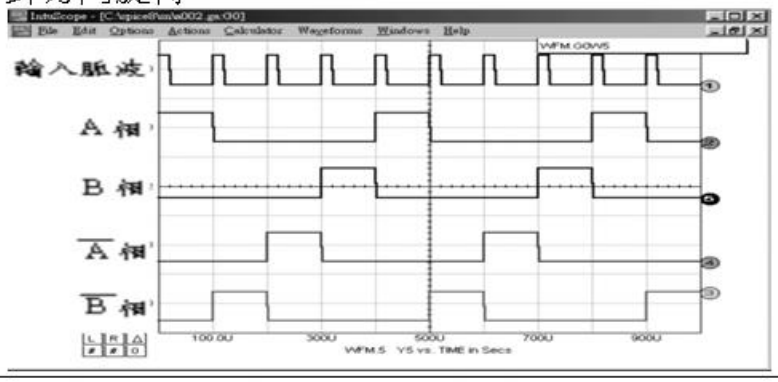
# 步進馬達的激磁方式

- 全步激磁方式又可分爲：1 相激磁與2相激磁兩種方式 。
- 1 相激磁：每次只激磁一相線圈，每輸入一個脈波，便產生一步級的迴轉 。(順時針旋轉)



## 步進馬達的激磁方式：全步- 1 相激磁

- 當激磁依A ->B -> /A->/B->A相順序，則馬達順時針方向旋轉;
- 若依B->A->/B->/A->B相順序激磁，則馬達依逆時針方向旋轉。

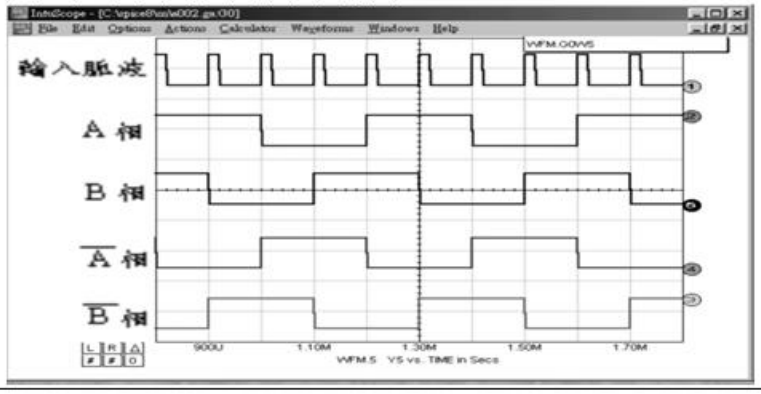



## 全步- 1 相激磁：優缺點

- 1 相激磁方式之優點：爲線圈消耗功率小，角精確度良好，
- 1 相激磁方式之缺點：其轉距小，加上阻尼特性不良，易失步

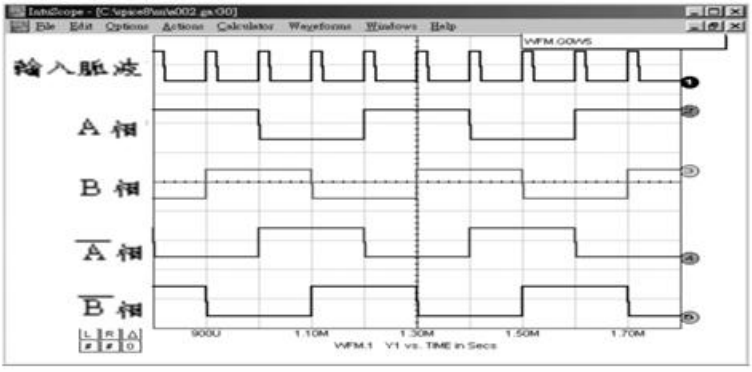
## 步進馬達的激磁方式：全步- 2 相激磁

- 2 相激磁：每輸入一個脈波，將有二相線圈激磁
- 當激磁依AB->B/A->/A/B->/BA->AB相順序，則馬達順時針方向旋轉。




## 步進馬達的激磁方式：全步- 2 相激磁

- 2 相激磁：每輸入一個脈波，將有二相線圈激磁
- 當激磁依BA->A/B->/B/A->/AB->BA相順序激磁，則馬達轉向爲逆時針方向。



## 全步- 2 相激磁：優缺點

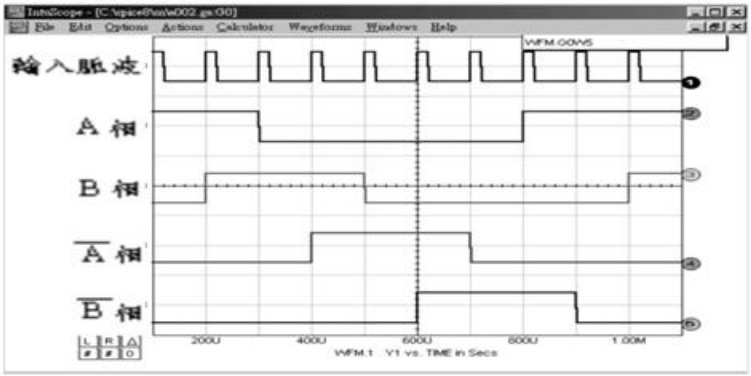
- 2 相激磁方式由於同時有兩組線圈激磁，優點：輸出轉距較大，加上阻尼效果良好，故能追蹤較高的脈波率。
- 2 相激磁方式其缺點爲耗電較大，容易發熱

## 步進馬達的激磁方式：半步激磁

- 此種激磁方式又稱1-2相激磁，激磁一相線圈和二相線圈交互進行，每加入一數位脈波所轉動之角度爲原步進角的一半，因此解析度可提高一倍，且運轉時相當平滑，故與2相激磁方式同受廣泛使用

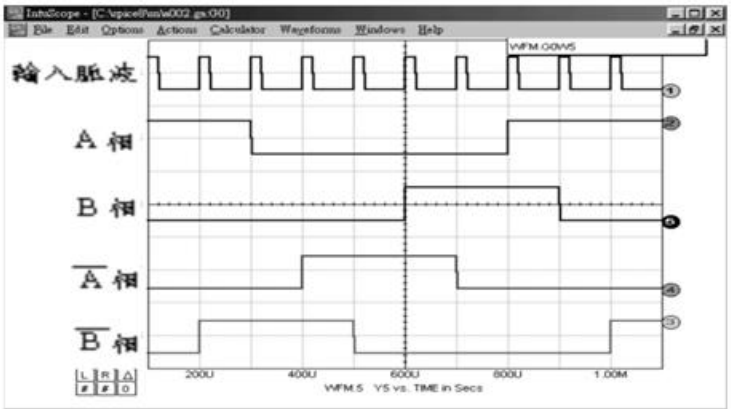
## 步進馬達的激磁方式：半步激磁

- A->AB->B->B/A->/A->/A/B->/B->/BA->A->AB相的順序激磁，則步進馬達將以順時針方向旋轉 。




## 步進馬達的激磁方式：半步激磁

- BA->A->A/B->/B->/B/A->/A->/AB->B->BA相順序激磁，則馬達逆時針方向旋轉。



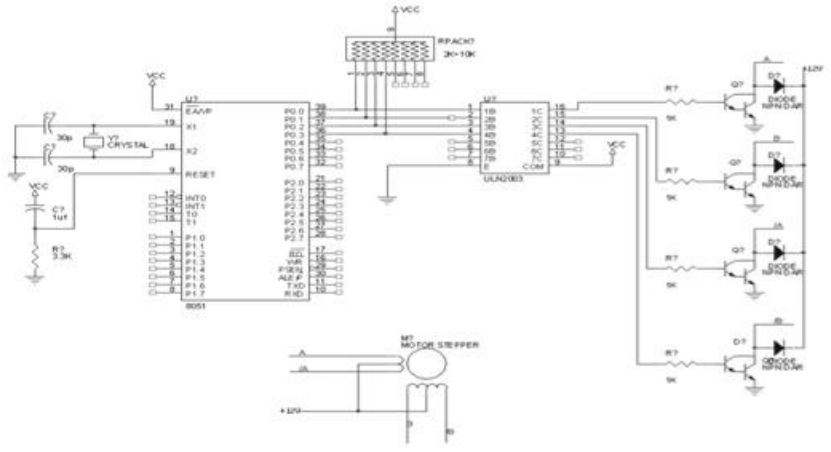
# 步進馬達控制電路實習

- 用單相激磁的方式,使步進馬達正轉10格　　程式分析：

```
#define n   10
main()
{
        unsigned char pp=0x1 ;      int Step ;

        for(Step=0 ;Step < (n+1) ; Step++){
           P0=pp ;
           pp <<=1 ;
           if(pp == 0x10){
                   pp=0x1 ;
           }
           Delay10ms() ;
        }
        for(;;) ;
}
```

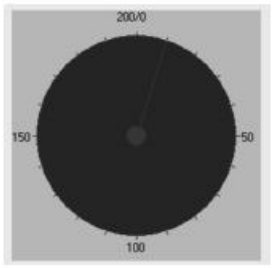
# 步進馬達控制電路實習

用單相激磁的方式,使步進馬達正轉10格



# 步進馬達控制電路實習

用單相激磁的方式,使步進馬達正轉10格




# 步進馬達控制電路實習

- 題目：使步進馬達正轉10格
- 目前修改爲馬達反轉10格

# 步進馬達控制電路實習

- 用半步激磁的方式,使步進馬達正轉10格(STEPM5.C)

```
#define n   10
char code pp[]={0x1 , 0x3 , 0x2 , 0x6 , 0x4 , 0xC , 0x8 , 0x9 } ;
main()
{
        int Step ;
        char idx_pp=0 ;
        for(Step=0 ;Step < (n*2+1) ; Step++){
           P0=pp[idx_pp++] ;
           if(idx_pp == 8){
                  idx_pp=0 ;
           }
           Delay10ms() ;
        }
        for(;;) ;
}
```

# 步進馬達控制電路實習

- 用半步激磁的方式,使步進馬達反轉10格(STEPM6.C)

```
#define n   10
char code pp[]={0x1 , 0x9 , 0x8 , 0xC , 0x4 , 0x6 , 0x2 , 0x3} ;
main()
{
        int Step ;
        char idx_pp=0 ;
        for(Step=0 ;Step < (n*2+1) ; Step++){
           P0=pp[idx_pp++] ;
           if(idx_pp == 8){
                  idx_pp=0 ;
           }
           Delay10ms() ;
        }
        for(;;) ;
}
```

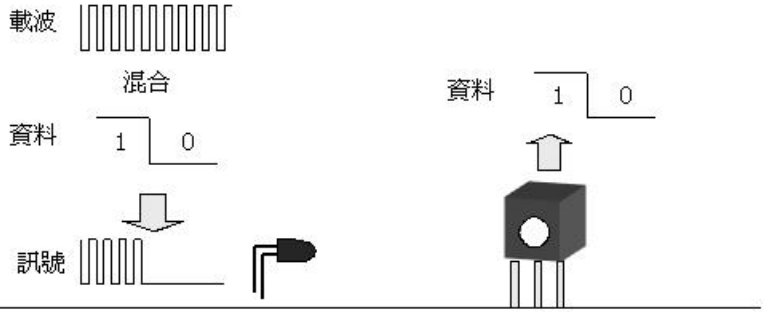
# 參考資料


| | |
|---|---|
| 步進馬達控制電路教學資料 | 陳德發 |
| 視窗５１模擬實務-C語言篇 | 蔡柏樟 |
| MCS-51與 k eil C語言入門實習-- | 董勝源,董浩文 |
| 8051單晶片實作-使用C語言 | 林振漢 |
| 視窗５１模擬實務-組合語言篇 | 蔡柏樟 |
| 單晶片微控器MCS-51 | 王宜楷 |


# 電腦鼠相關資料

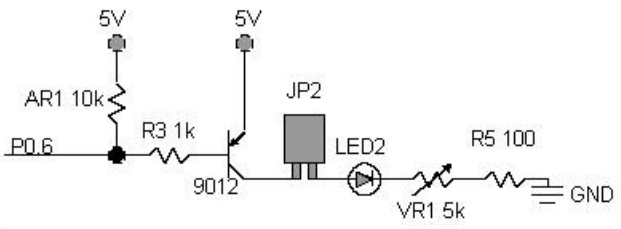
# 實驗1:紅外線接收模組始用

- 紅外線接收模組是一個利用帶通濾波器來做抗干擾的數位紅外線接收器，需將紅外光發射器混入濾波器中心頻率才能接收到，使用方式如下



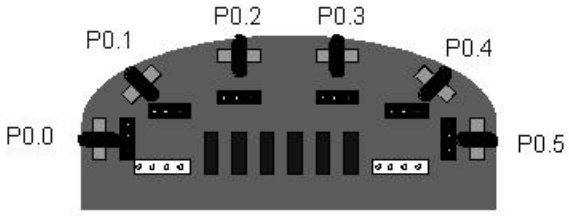

# 電路架構

- 爲了方便做不同的實驗，我們將紅外線發射器分爲兩組，由P0.6、P0.7輸出訊號推動Q1與Q2，並由紅外光發射LED旁的短路器JP2~JP7來做選擇。



## 電路架構

- 除了LED發射訊號分爲兩組，其他每個紅外線接收模組輸出訊號分別獨立輸入P0.X如下圖:
- 其中紅色點代表接到P0.6的短路點




## 紅外線接收器頻率與程式的關係

- 我們使用的紅外線接收器，頻率爲38kHz，故我們在使用上需要產生一個38kHz 50%的方波，讓電腦鼠發送。
  週期=1/38k(Hz)≈26.31579μ(s)要丟出一個50%脈波，故約13.1579μ(s)要反向一次。以12MHz石英震盪器爲例約13個週期就要反向一次。而依照Data Sheet 大約要持續25個脈波(650μs)才能判斷有無訊號

# 紅外線接收模組實驗一

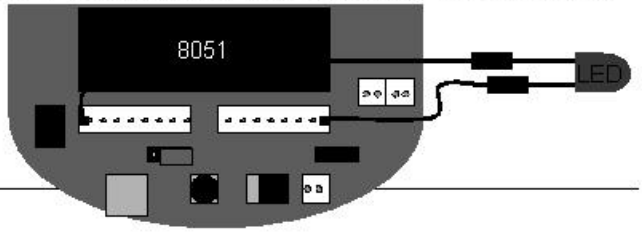
- 我們可以利用計時器中斷，每次進入中斷副程式時將P0.6反向產生所需脈波訊號，而計時器所需計時值要設定多少呢?我們可以這樣算:
- 38k(Hz) 50%方波的週期爲:
  1/38k(Hz)=26.31579μ(s)
- 由於50%時間要高態50%時間要低態故每
  26.31579μ(s)/2=13.15789μ(s)訊號反向一次
- 我們使用22.1184M(Hz)石英震盪器給8051故每
  12/ 25M(Hz)=0.54253μ(s)計時器值+1
- 故計時器需要記數:
- 13.15789μ(s)/0.54253μ(s)=24.252≈24次

# 紅外線接收模組測試程式1

```
#pragma oe db pw(80) SM SB CD     //產生高階組譯檔(除錯需要)
#include     <\Atmel\AT89X52.h>   //載入52系列特殊位址暫存器名稱(名稱對應位址)
#include     "Delay.h"            //載入SIMLAB 8051專用延遲副程式(資料夾內需要有SDelay.h才能用
unsigned char i;                  //設定八位元全域變數i
main()
 {
  P3_7=0;                         //作爲測試用接地端；
  TMOD  = 0x02;                   //0000 0010，設定Timer0爲mode2(8bit)自動載入計時
  IE=0x82;                        //致能Timer0中斷並允許中斷
  while(1)                        //無窮回圈將下面程式包住，希望能不斷重複執行
  {
   TL0=TH0=(256-24);              //計時爲27次記時中斷一次
   TR0=1;                         //啓動Timer0開始計時
   i=50;                          //希望輸出25次脈波(25*2=中斷的執行次數)
   while(i);                      //自我空轉，與中斷副程式的i--配合，等待25次脈披輸出完成
   TR0=0;                         //關閉計時器
   P1=P0  & 0x01;                 //將紅外光感測器值利用P0 AND 0x01作遮罩後丟給P1作顯示
   Delay10ms();                   //SIMLAB專用延遲副程式，這邊可以寫馬達控制程式之類的
  }
 }
void Delay_Out(void) interrupt 1  //Timer0中斷副程式
 {
  i--;                            //i=-1
  P0=P0  ^ 0x40;                  //利用 P0 XOR 0x40讓P0.6反相
 }
```

## 測試方法

- 記得先將剛剛燒錄的JP1短路器回復原位，在將JP2短路器用來短路前面投影片的紅點與中間共同端。使P0.6訊號送進LED2再由LD1接收丟進8051，我們可以在P1.0透過杜邦線接到LED長腳端(+)，將測試接地端點P3.7透過杜邦線接到LED短腳端(-)完成測試用硬體線路

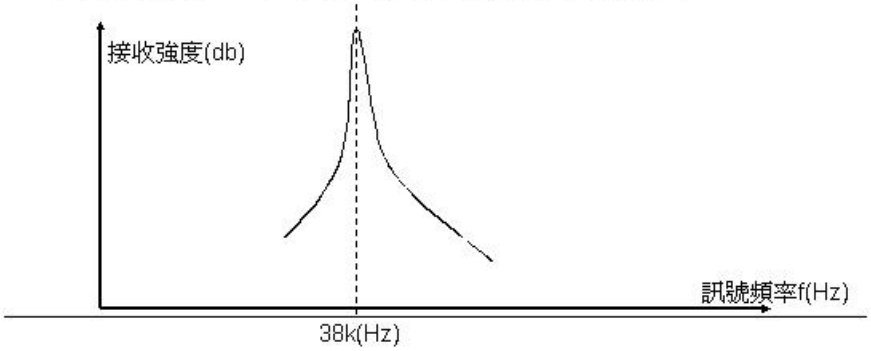



## 測試結果

- 正常狀況下有感測到紅外光LED發出的光，P1.0將會是低態(測試用led是暗的)假設沒感測到，P1.0將會是高態(測試用led是亮的)，我們發現LED一直都是暗的，原因在於紅外光接收模組在38k(Hz)有極高的靈敏度，正常來說可以感測到10公尺外反射回來的紅外光LED的訊號，在加上室內有太多可反光的物體，故不管你轉哪個角度都會收到紅外光訊號。

## 感測距離調整

- 我們如果把頻率訂在38k(Hz)靈敏度實在太高不適合做短距離偵測，該如何改進呢?
  我們觀察一下紅外線接收模組的特性:




## 感測距離調整

- 以上圖來看，如果我們的訊號頻率調高(調低也OK)，他的接收靈敏度就會降低許多，利用此特性，我們嘗試使用軟體來調整紅外光接收模組的靈敏度

# 紅外線接收模組測試程式2

```
#pragma oe db pw(80) SM SB CD    //產生高階組譯檔(除錯需要)
#include    <\Atmel\AT89X52.h>   //載入52系列特殊位址暫存器名稱(名稱對應位址)
#include    "Delay.h"            //載入SIMLAB 8051專用延遲副程式(資料夾內需要有SDelay.h才能用
#define     ti          17       //設定一個自定數值ti=20，利用它來控制脈波頻率
unsigned char i;                 //設定八位元全域變數i
main()
 {
  P3_7=0;                        //作爲測試用接地端；
  TMOD  = 0x02;                  //0000 0010，設定Timer0爲mode2(8bit)自動載入計時
  IE=0x82;                       //致能Timer0中斷並允許中斷
  while(1)                       //無窮回圈將下面程式包住，希望能不斷重複執行
  {
   TL0=TH0=(256-ti);             //計時爲" ti"次記時中斷一次
   TR0=1;                        //啓動Timer0開始計時
   i=1198/ti;                    //希望輸出650μ(s)/ti/(12/22.1184M(Hz))次中斷
                                 //使其訊號持續時間接近650us
   while(i);                     //自我空轉，與中斷副程式的i--配合，等待脈波輸出完成
   TR0=0;                        //關閉計時器
   P1=P0  & 0x01;                //將紅外光感測器值利用P0 AND 0x01作遮罩後丟給P1作顯示
   Delay10ms();                  //SIMLAB專用延遲副程式，這邊可以寫馬達控制程式之類的
  }
 }
void Delay_Out(void) interrupt 1 //Timer0中斷副程式
 {
  i--;                           //i=-1
  P0=P0  ^ 0x40;                 //利用 P0 XOR 0x40讓P0.6反相
 }
```

# 測試結果

- 這次感測距離拉到10cm左右，我們推算一下每17個脈波反相一次的脈波週期爲:
  - 17(脈波數)*2(高與低態各20次)*12/22.1184M(Hz)
  - =18.44618μ(s)
- 故脈波頻率=1/脈波週期=1/19.2μ(s)≈54k(Hz)
- 我們發現頻率離38k(Hz)差距越大，感測距離越短，同學們可以試試其他脈波數設定。
  - 利用此特性是否可以用來感測距離?

# 思考與實驗

- 假設我希望同時感測6個紅外光模組，我們可以怎麼做?
- 假設我們希望6個模組裡，四個感測距離較長，兩個感測距離較短，我們能怎麼作?
- 既然我們有兩組的脈波輸出選擇，那我們能利用它做什麼?

# 紅外線接收模組測試程式3

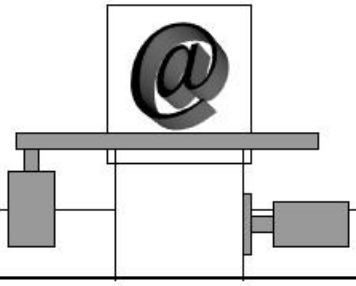
```
#pragma oe db pw(80)  SM SB CD        //產生高階組譯檔(除錯需要)
#include   <\Atmel\AT89X52.h>         //載入52系列特殊位址暫存器名稱(名稱對應位址)
#include <IR.h>                       //載入紅外線模組副程式(使用Timer2)

main()
{
 unsigned char X,Y;                   //設定變數X與Y用來暫存不同次的感測值
 IR_inc();                            //初始化紅外線接收模組
 while(1)
 {
  X=IR(17,0);                         //感測距離參數 設17，丟入P0.6輸出
  Y=IR(23,1);                         //感測距離參數 設16，丟入P0.7輸出
  P1=(X &0x2d) + (Y &0x12);           //依照短路器選擇狀態作兩組訊號的合成
 }
}
```

## 紅外線接收模組測試程式3

- 我們將測試程式2製作出IR.H的副程式檔。
- 並改進了程式的寫法
- 程式3，我們將”左” ”前” ”右” ”4顆感測器設置在P0.6而”左前” ”右前”兩顆感測器設置在P0.7，我們可以設定不同的感測距離，並將期訊號合成後再輸出控制其他功能(比如控制馬達走向)

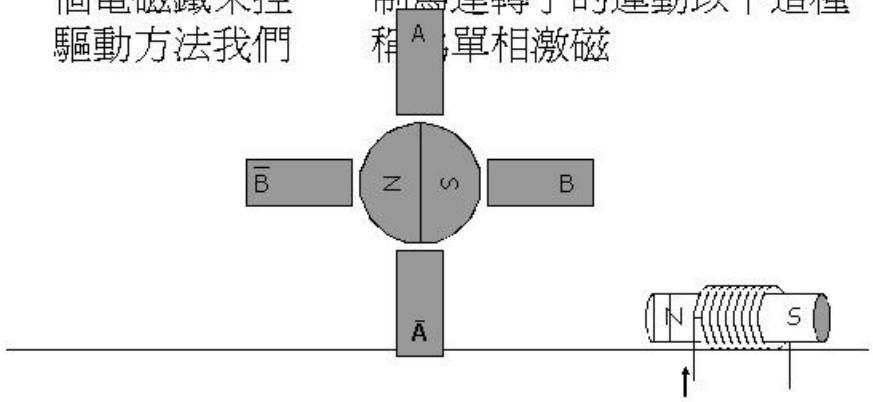
## 步進馬達是什麼樣的馬達?

- 步進馬達常使用於線性移動控制上、常裝在印表機內的用來移動印刷頭與紙張、特點爲不用回朔機制即可有很強的精準度，馬達可轉到某一定點停住還有很強的力量。
- 但相對的步進馬達價格較高，而且耗電流較大



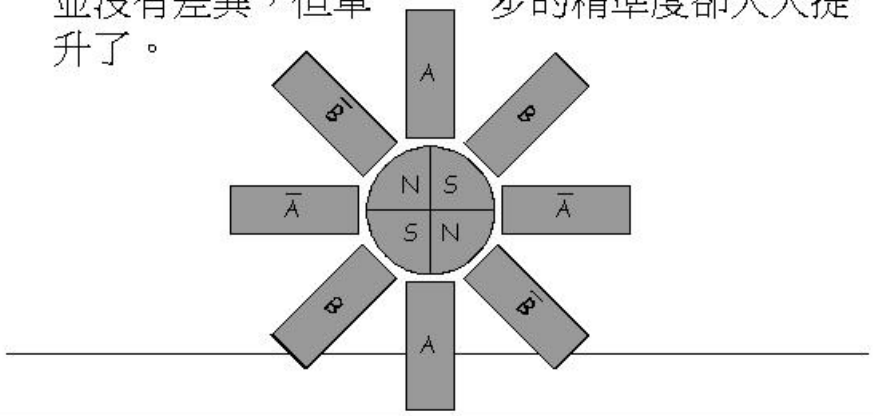
## 步進馬達控制實驗

- 步進馬達是利用磁鐵同姓相斥異性的原理製程，請看動畫，這是步進馬達設計的構想，利用四個電磁鐵來控　　制馬達轉子的運動以下這種驅動方法我們　　稱為單相激磁



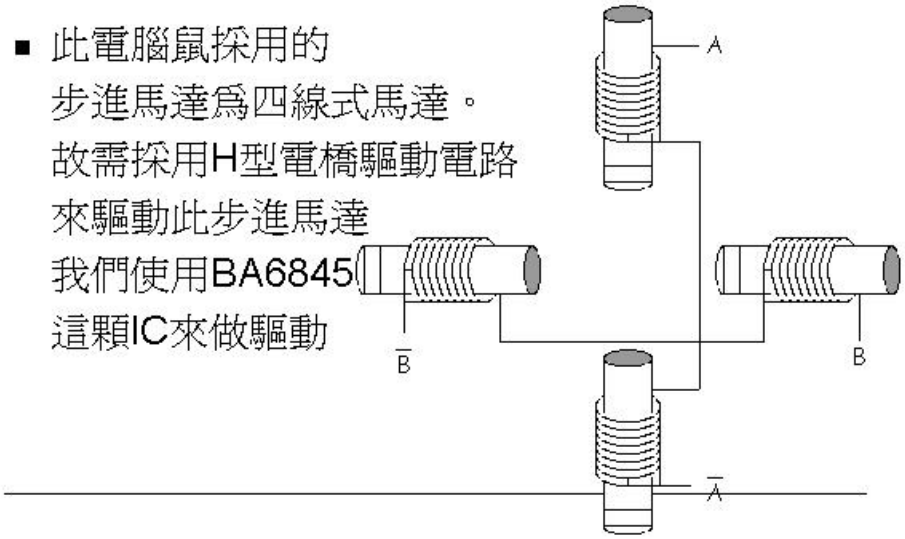

## 更高精度的步進馬達如何製作的？

- 爲了提高精準度我們會提高磁極的極數，再將同一型(同名稱)線圈做連接。控制方式與單極並沒有差異，但單　　步的精準度卻大大提升了。



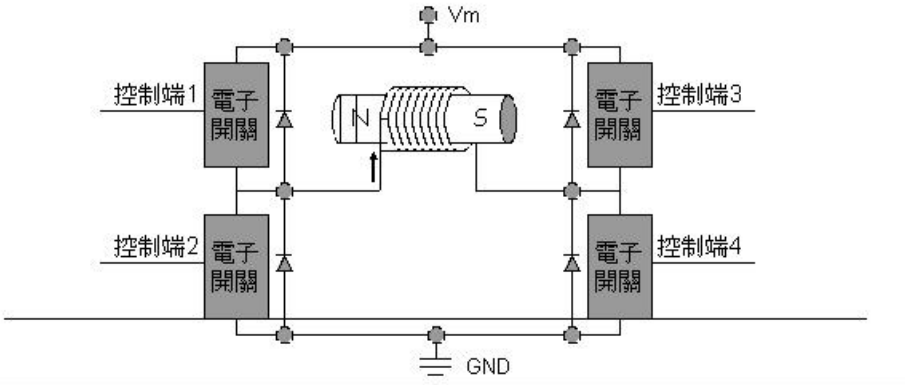
# 步進馬達驅動IC的控制方式

- 此電腦鼠採用的
  步進馬達爲四線式馬達。
  故需採用H型電橋驅動電路
  來驅動此步進馬達
  我們使用BA6845
  這顆IC來做驅動



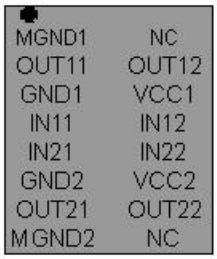

# H型電橋驅動方式示意圖

- 架設我們開啓電子開關1、4則如下圖供給電流，如改爲2、3開啓，則電流反相，二極體則用來吸收電感開路電壓用的，在BA6845已有內建。

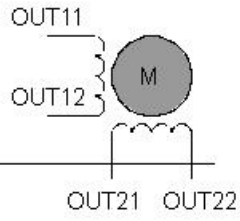


# 電路控制電路解說

- BA6845是一顆16pin IC，
  他有兩組H型電橋電路，
  可控制兩組線圈，
  (步進馬達正好兩組線圈組)
  而兩個vcc就接到馬達電源7.4v，
  四個gnd就接地。控制端則爲in如下:




| IN11/21 | IN12/22 | OUT11/21 | OUT12/22 |
|---|---|---|---|
| 0 | 0 | 高阻抗 | 高阻抗 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 高阻抗 | 高阻抗 |
| 1 | 1 | 0 | 1 |




# 電路控制電路解說

- 我們由上一張表可得知INX1是控制馬達線圈是否通電，INX2則是控制順向通電或反相通電。由於8051重置後每個Port都是高態，如果8051Port直接接到BA6845則會進入反相通電(由OUTX2流進OUTX1)的狀態不斷耗電
- 假設我們不希望剛開機馬達就不斷的消耗電力則必須讓INX1一開始就是0V，故我們加入反相器在INX1與8051之間，故8051控制馬達則如下表

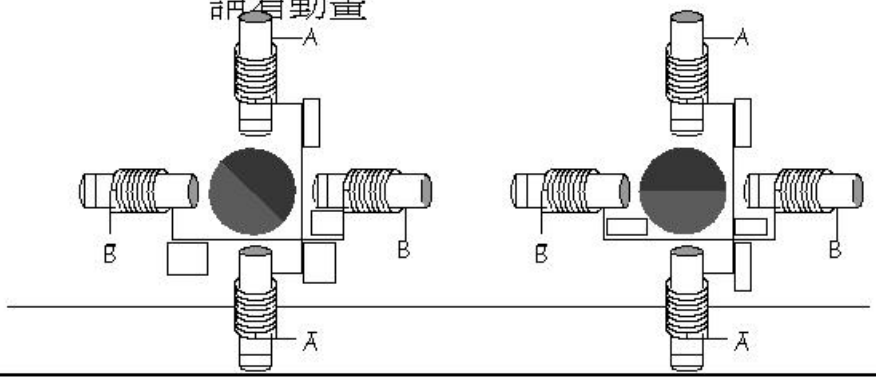
| 左馬達 | P2.4/P2.5 | P2.6/P2.7 | A/B | Ā/B̄ |
|---|---|---|---|---|
| | 0 | 1 | 高阻抗 | 高阻抗 |
| | 0 | 0 | 1 | 0 |
| | 1 | 1 | 高阻抗 | 高阻抗 |
| | 1 | 0 | 0 | 1 |

| 右馬達 | P2.0/P2.1 | P2.2/P2.3 | A/B | Ā/B̄ |
|---|---|---|---|---|
| | 0 | 1 | 高阻抗 | 高阻抗 |
| | 0 | 0 | 1 | 0 |
| | 1 | 1 | 高阻抗 | 高阻抗 |
| | 1 | 0 | 0 | 1 |

# 步進馬達激磁方式

- 步進馬達驅動方式除了一開始所介紹的單相激磁外，還有雙相激磁、半步激磁、微步進系統，我們介紹一下雙相與半步激磁這兩種方式。

請看動畫



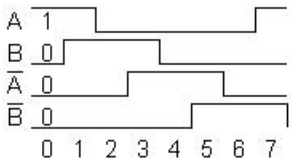

# 三種激磁方式的特性

- 對我們電腦鼠的步進馬達而言:
  - 一步是7.5度
  - 單相激磁的平均的功率損耗爲P

| | 步進角 | 功率損耗 | 特性 |
|---|---|---|---|
| 單向激磁 | 7.5 | P | 1.功率小，比較省電<br>2.容易脫步，無法快速加減速 |
| 雙向激磁 | 7.5 | 2*P | 1.功率高，吃電比較兇<br>2.步進穩定(不易脫步)，可快速加減速<br>3.溫度上升快，可能需要散熱 |
| 半步激磁 | 3.78 | 1.5*P | 1.步進角爲原本一半，控制性提高<br>2.低速時平穩性高(較不會讓車體震動)。<br>3.其他特性介於單相與雙相激磁之間 |

# 控制方式

- 我們採用半步激磁的方式如下




| 步數 | A | B | A | B |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 0 | 1 |
| 7 | 1 | 0 | 0 | 1 |

| 步數 | P2.3 | P2.2 | P2.1 | P2.0 |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 1 | 0 | 1 | 1 |
| 5 | 0 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 1 |
| 7 | 0 | 0 | 1 | 0 |

# 第一個步進馬達程式

- 我們開始做一個簡單的程式，讓馬達轉一圈後停止，加入cycle_time用來設定計時器計時值控制移動速度、i來做驅動查表、j來記錄移動距離

# 步進馬達範例程式1

```
#pragma oe db pw(80) SM SB CD          //產生高階組譯檔
#include <\Atmel\AT89X52.H>            //載入SFR(特殊功能暫存器)對映名稱
#include "Delay.h"                     //載入SimLab8051萬用延遲副程式(22.1184MHz)
unsigned char stepTAB[]={0xaa,0x00,0x55,0x11,0xbb,0x33,0x77,0x22};   //陣列內容爲半步激磁順序性資料
unsigned char i,j;                     //設定變數i當查表計數器、j當步數計數器
#define cycle_time7000                 //設定馬達速度，建議4608~65535範圍內
main()
{
 TL2=RCAP2L=(65536-cycle_time);                //由我們所設速度，設定計時器低8位元資料
 TH2=RCAP2H=(65536-cycle_time)>>8;             //由我們所設速度，設定計時器高8位元資料
 IE=0xa0;                              //致能Timer2中斷並允許中斷
 TR2=1;                                //開始計時
 i=0,j=0;                              //將i查表計數器設定初值爲0、j步數計數器清除爲0
 P2=stepTAB[i] | 0xf0;                 //依照i值查表(陣列)給定P2輸出激磁步進馬達(第0步)
 while(j<96);                          //等待移動96(半)步後停止，使其正好轉動一圈
 TR2=0;                                //關閉計時器，讓步進馬達不再移動，
 Delay_ms(50);                         //等待50ms，由於此時持續供電給步進馬達，利用靜止扭力來煞車(克服慣性)，
 P2=0xff;                              //煞車完畢，停止供電給步進馬達以節約電能
 while(1);                             //進入無限迴圈不再做任何控制馬達的動作
}
void Timer2_ISR() interrupt 5          //計時器中斷副程式
{
 P2 = P2 | 0xfc;                       //關閉所有激磁避免步進馬達錯位
 i++,j++;                              //查表計數與步數計數部+1
 if(i==8) i=0;                         //如果查表計數=8，則將查表計數歸零
 P2 = stepTAB[i] | 0xf0;               //對應查表給定馬達驅動值
 TF2=0;                                //清除中斷旗標，以利下次中斷使用
}
```

# 思考與程式改進方式

- 如何反轉?
- 如何控制另一顆馬達?
- 改成單相激磁(需要省電)
- 改成雙相激磁(需要大扭力)
- 如果兩顆馬達要同時控制，我們要如何控制?
- 爲方便使用，是否可能副程式化?

# 步進馬達程式2

- 我們將馬達控制程式整個做在Timer2(包括左右馬達控制)，只需在主程式做參數設定，即可控制步進馬達。
- 其中:
  - ML_BS → 左馬達移動步數。(-32768~32767)
  - ML_TIME → 左馬多少單位時間移動一步
  - ML_COM →左馬達控制狀態
    - 0滑行停止、1前進、2後退、3煞車停止
  - L →左馬達
  - R→右馬達

# 步進馬達程式2 (一小段控制範例程式碼)

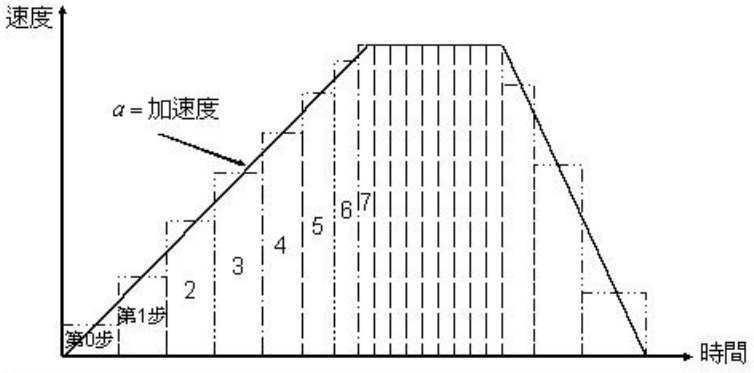
```
ML_BS = MR_BS = 0;          //步數計數器歸零
ML_TIME=15,ML_COM=1,MR_TIME=15,MR_COM=1;
                            //左右馬達前進，移動週期15
while(ML_BS<=96 && MR_BS<=96);
                            //判斷是否前進96(半)步了
ML_TIME=1,ML_COM=3,MR_TIME=1,MR_COM=3;
                            //左右馬達煞車
Delay100ms();               //等待0.1秒
ML_TIME=1,ML_COM=0,MR_TIME=1,MR_COM=0;
                            //左右馬達停止
ML_BS = MR_BS = 0;          //步數計數器歸零
```

# 更改方式與實際運用

- 利用參數的設定，我們控制步進馬達將變的很簡單，我們利用參數的設定就能控制我們的步進馬達。
- 試試看花些時間觀察一下，範例是如何將馬達控制參數化的，我們可以修改哪裡來產生我們所需的控制參數
- 題目:到定點停止、倒車入庫、Z型移動

# 加減速控制實驗

- 以下是我們將步進馬達加減速以圖示表示出來速度越快每步的間格時間越小



# 理論值計算

我們設定:

n = 步數、t = 時間、$v$ = 速度、$a$ = 加速度並且$a$為常數(等加速度)

如果我們最小距離單位設定為1(步)等於馬達走一步的距離則

$v = \frac{\text{距離}}{\text{時間}} = \frac{\text{移動了幾步}}{\text{單位時間}}$ 可得到

$n = \int_0^t v(f)\,dt = \int_0^t at\,dt = \frac{at^2}{2}$ 為從第0秒 ~ t秒所移動的步數n

則走過n步所花的時間為

$t_n = \sqrt{\frac{2n}{a}}$ 由此可推得第n步所需要持續的時間 $T_n = t_{n+1} - t_n = T_0(\sqrt{n+1} - \sqrt{n})$ 其中 $T_0 = \sqrt{\frac{2}{a}}$ = 第0步所花時間

故我們只需要設定完$T_0$則可將其他部數的相關性透過公式算出來

我們先定出 $a = 1000(\frac{\text{步}}{\text{秒}^2})$ 再Excel建表定義出0 ~ 300 步的所需時間

在換算成步進馬達所需控制值後建表貼進程式內。之後在程式內即可利用查表輕鬆控制加減速。

# 步進馬達控制程式3

- 我們將程式2建立步數查表，依照目前前進步數給定不同的計時器值來做穩定的加減速動作。詳細控制方法請看step3.c內的注解，加入加減速控制後，步進馬達較不易脫步，同學們可以試試看。
- 經過測試，我們發現加入了加減速控制的電腦鼠，不需再使用煞車這個控制指令，我們可以再次精簡我們的步進馬達控制副程式，使其控制性質更好，我們將左右馬達分別控制，改成車體方向控制。

# 步進馬達控制程式4

- 步進馬達控制程式4是將3精簡化並改進效能的控制程式，我們將煞車精簡掉，並將效能最佳化(以T0與T1分別控制左右馬達取代T2一次控制兩個馬達使移動更加順暢、並將查表值馬達加減速解析度提高，並將可以精簡的變數移除)，之後我們將程式隱藏進motor.h使得程式更爲易讀。
- 不多說我們直接再下一張看看我們的新程式。

# 步進馬達控制程式4

```
#pragma oe db pw(80)  SM SB CD        //產生高階組譯檔
#include <\Atmel\AT89X52.H>           //載入特殊功能暫存器名稱對應檔
#include <motor1.h>                   //載入馬達控制副程式
void        main(void)
{
 motor();                             //馬達控制副程式初始化
 while(1)
 {
  pulseL=pulseR=1000;                 //左馬達步數=右馬達步數=1000步
  dirL=dirR=ahead;                    //左馬達移動方式=右馬達移動方式=前進(ahead)
                                      //註:前進(ahead)、後退(back)
  stateL=stateR=run;                  //左馬狀態=右馬狀態=跑(run)
                                      //註:停止(stop)、跑(run)、暫停(wait)
  while(1);                           //程式空轉，此時可以插入別的程式
 }
}
```

## 步進馬達控制程式4

- 我們後面的程式將都以步進馬達控制程式4爲基準發展，而當遇到超出副程式可控制的範圍，我們則修改.h檔(副程式)來達到我們所需功能，當然同學們也能試試看各種不同的控制方法

## 閃避障礙物自走車設計

- 我們將先前所學的紅外線接收模組與步進馬達作個結合例用IR.h與motor1.h即可做出自動閃避障礙物的自走車
- 其中閃避的法則我們可以透過表格將其可能狀態都列出，依照其表格化簡，來製成我們的控制程式。

# 控制簡化後的構想

- 當正前方有障礙物時:
  →左馬達後退、右馬達前進。
- 當左(右)前方有障礙時:
  →左(右)馬達前進、右(左)馬達停止。
- 當左(右)方有障礙時:
  →左(右)馬達前進，右(左)馬達低速前進

# 程式

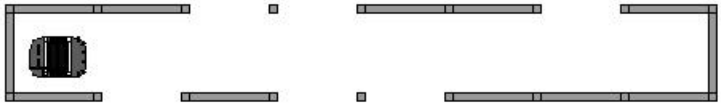
```
#pragma oe db pw(80)  SM SB CD                    //產生高階組譯檔
#include <\Atmel\AT89X52.H>                       //載入特殊功能暫存器名稱對應檔
#include <motor1.h>                               //載入馬達控制副程式
#include <IR.h>                                   //載入紅外線接收模組副程式
void        main(void)
{
 unsigned char x,y,ir_reg;
 IR_inc();
 motor_inc();                                     //馬達控制副程式初始化
 while(1)
 { x=IR(20,0);
  y=IR(20,1);
  ir_reg= (x & 0x2d)+(y & 0x12);
  if((ir_reg  & 4) ==0)pulseL=pulseR=20,dirL=back,dirR=ahead,stateL=stateR=run;
  else
  { if((ir_reg  & 16)==0)pulseR=70,dirR=ahead,stateR=run,stateL=stop;
   else
   { if((ir_reg  & 2)==0)pulseL=70,dirL=ahead,stateL=run,stateR=stop;
    else
    { if((ir_reg  & 32)==0)pulseL=1000,pulseR=70,dirL=dirR=ahead,stateL=stateR=run;
     else
     { if((ir_reg  & 1)==0)pulseL=70,pulseR=1000,dirL=dirR=ahead,stateL=stateR=run;
      else pulseL=pulseR=1000,dirL=dirR=ahead,stateL=stateR=run; }}}}}
```

## 程式講解

- 我們利用if敘述將前面6種狀態寫進程式裡，即可將構想實現。
- 程式尚可改善，比如說馬達正轉→反轉的變化，並沒有將馬達先停止而直接變化容易產生脫步。
- 同學們也可以想想，怎麼作貼著左(右)側牆面固定距離移動程式

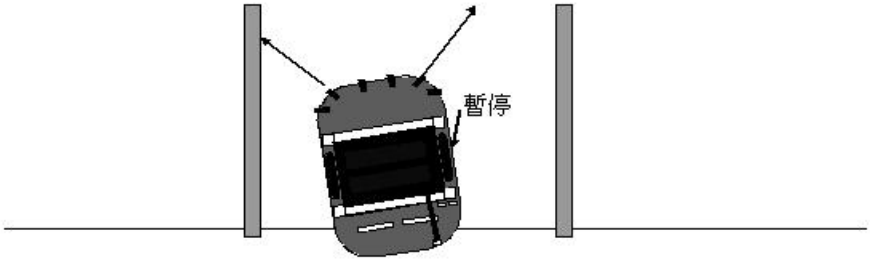
## 自動矯正直線移動程式

- 在學會馬達與紅外線接收模組後，我們開始嘗試在迷宮內讓電腦鼠移動，首先必須避免碰撞迷宮，故我們先來嘗試直線校正移動程式
- 我們先組出以下迷宮形式邊牆可以隨意移除幾個而故定柱必須全數接上，讓車體移動到對面盡頭停止。



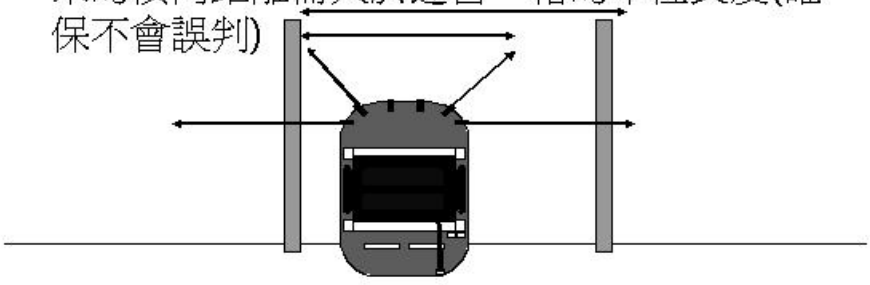
## 直線矯正程式

- 我們利用LD2(左前)與LD5(右前)方感測器來作感測，正常都沒感測到則直走，如果其中一個感測到障礙物，則使對應馬達暫停來達到矯正的目的



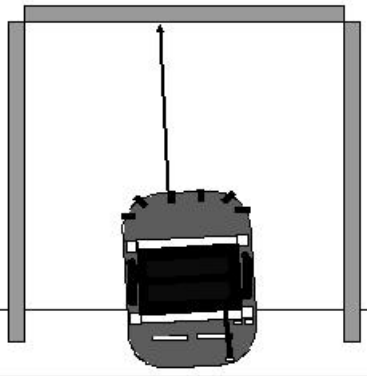

## 感測器調整

- LD2與LD5感測距離總和出來的橫向距離，需略小於迷宮大小，同學們可以調整紅外線接收程式來達成(或利用可變電阻)，”左” ”右”側感測距離與”右前” ”左前”感測器總和出來的橫向距離需大於迷宮一格的單位長度(確保不會誤判)



## 煞車

- 當前方有障礙物，必須經過煞車避免脫步，當前方感測器感測到有牆面則開始進行煞車，我們緩衝多走x公分後停止,使車體盡量停止再正中間。

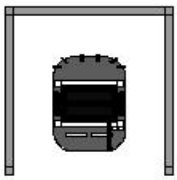


## 試試看

- 我們將測試範例程式燒進去，調整完感測器後，放到跑道移動，看看結果如何?

## 右(左)手法則，中右(中左)法則

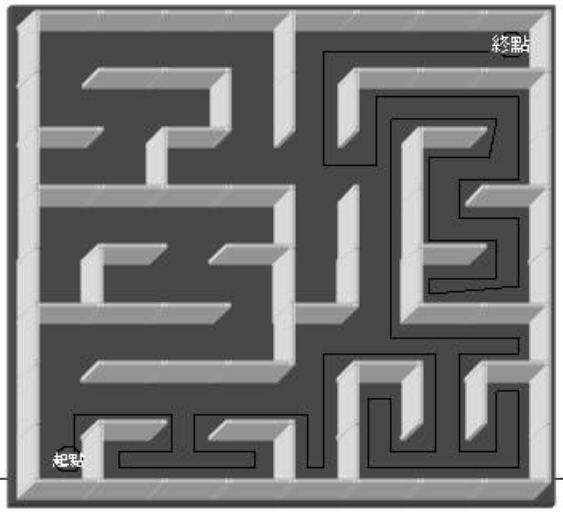
- 右手法則顧名思義，假設車體右方有缺口即往右方移動，其次前進(前方有缺口)，再來左移、後移。(左手法則同裡)
- 中右(中左)則是將前方移動的優先權拉到比右(左)方高(再向心法則需要用到



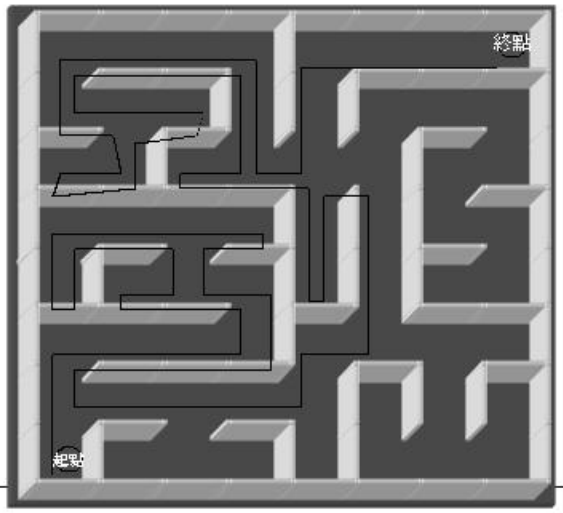
## 無記憶迷宮移動

- 假設迷宮型式未定，但出口與入口都在迷宮外側，則左手法與右手法必定能走出迷宮

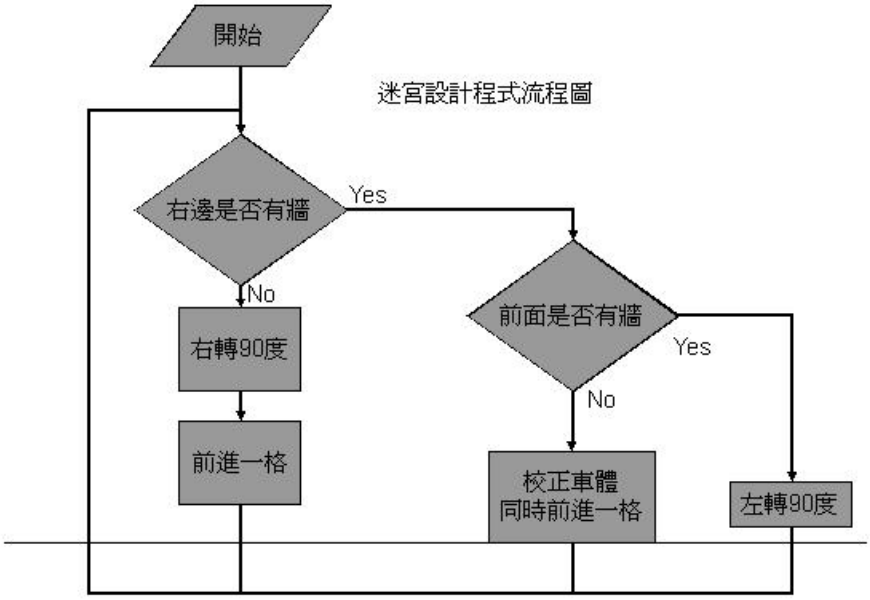
## 右手法示意




## 左手法示意



# 右手法則流程圖

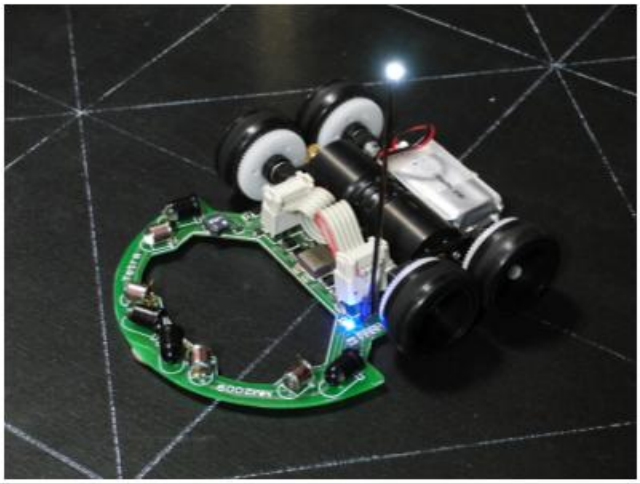



# 相關資源簡介

- Min7.2 Ng BengKiat 黃明吉 Singapore新加坡
- 2011台灣古典電腦鼠國際邀請賽
- 日本加藤的電腦鼠 Tetra（2009）

- 新加波黃明吉教授Min6電腦鼠。
  2007、2008日本電腦鼠冠軍。

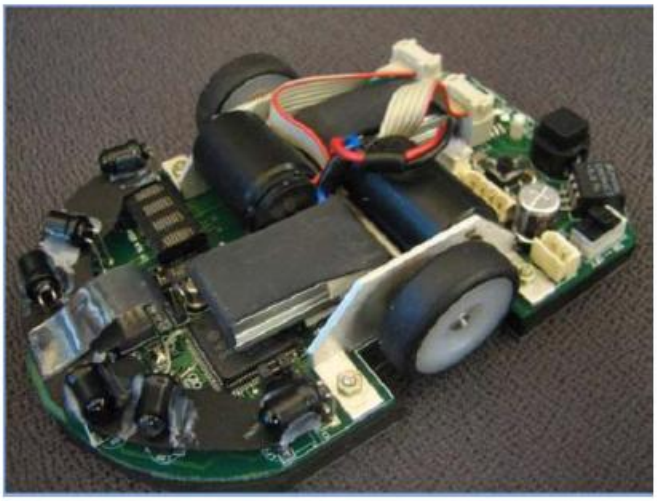
## 日本加藤的電腦鼠 Tetra (2009)



## 日本加藤的電腦鼠 Tetra (2009)

- Tetra (加藤雄資氏/名古屋工業大学)。
- 2009日本1/2 電腦鼠冠軍。
- 暱稱：Tetra
- 長寬高：9*7.4*2.2cm
- 重量：71g
- 微處理器：STM32(64MHz)
- 陀螺儀：ADXR300
- 伺服馬達：Faulhaber 1717
- 減速比：60：16
- 輪胎寬度：8.5mm
- 紅外線發射：SFH4550
- 紅外線接收：TPS601

## 新加坡黃明吉的Min6



## 新加坡黃明吉的Min6

- 新加波黃明吉教授Min6電腦鼠。
- 2007、2008日本電腦鼠冠軍。
- 暱稱： Min 6
- 長寬高：11.7*7.4*2.5cm
- 重量：94g
- 微處理器：Hitachi 2633R
- 陀螺儀：ADXR300
- 伺服馬達：Faulhaber 1717SR
- 減速比：32：12
- 輪胎寬度：機密
- 紅外線發射：OPE5594
- 紅外線接收：TSL262R

- 簡報完畢
  敬請指教

- 承辦機構: 秉華科技有限公司
- 網址: http://www.be-friend.com.tw
- E-MAIL： service@be-friend.com.tw
- be.friend@msa.hinet.net