

## 歡迎參加 MPLAB C18 v2.0 教育訓練

### ◆ 使用軟體工具

- MPLAB IDE v8.20 (或更新版本)
- MPLAB C18 v3.15 Student Edition (或更新版本)
- MPASM , MPLINK , MPLIB

### ◆ 使用硬體工具

- MPLAB ICD2
- Microchip WAP001 實驗版 (PIC18F4520 inside)

### ◆ 參考書籍

- MPLAB C18 Compiler User's Guide & Library
- MPASM User's Guide with MPLINK and MPLIB
- MPLAB IDE User's Guide



## ANSI C

ANSI (American National Standards Institute) 是美國負責訂定國家標準的機構。它除了訂定各種工業及產品的標準外，也同時負責電腦 程式語言的標準。ANSI 在 1989 年發表了語言的標準 X3.159，也就是我們一般所稱的 ANSI C 標準。



# MPLAB C18 Workshop

## 第一天課程

認識 MPLAB 的環境  
C 比組合語言更簡單  
基本周邊功能操作  
MPLAB C18 使用詳述



## 第一章

### 認識 MPLAB 的環境

1. C18 vs. 組合語言
2. PIC18F4520 主要架構
3. 認識 MPLAB IDE & MPLAB C18



## 想一想 ...

- ◆ C 語言和組合語言，那一個比較容易寫？
- ◆ C 語言和組合語言，那一個比較容易閱讀？
- ◆ 在一定複雜功能需求下用 C 語言和組合語言來撰寫程式，那一個比較省時間？
- ◆ C 語言和組合語言，誰的程式相容性高、換個微控制器也簡單？
- ◆ 你看的懂別人寫的組合語言嗎？換作是 C 語言是不是就好維護了？



## 為什麼要用 C

- ◆ 複雜程式簡易化
  - 少於2K的程式 - 可用組合語言
  - 2K ~ 8K的程式 - 最好用C來完成
  - 大於8K的程式 - C 語言
- ◆ 程式容易撰寫
- ◆ 高時效性
- ◆ 相容性高
- ◆ 程式的閱讀性
- ◆ 日後的維護



## C 語言與組合語言比較

### ◆ 組合語言

- 困難不容易理解
- 必需對MCU架構很清楚
- 寫完後容易忘記
- 除錯較為困難
- 寫大程式的時間很辛苦
- 花較長的時間
- .....

### ◆ 優點

- 程式碼較小 - 省錢
- 執行速度較快
- 讓老板覺得你很厲害

### ◆ C 語言

- 簡單易讀易懂
- 僅需對MCU有概念即可
- 寫大程式時較簡單
- 除錯較簡單
- 別人也很容易看懂程式
- 容易修改
- .....

### ◆ 缺點

- 組合語言優點的相反，就是C的缺點



## C 面對組合語言各項優勢的 solution

### ◆ 組合語言速度較快

- 加快振盪頻率
- PIC18Fxxxx @40MHz，指令執行速度100nS

### ◆ 組合語言編譯後的程式碼較少

- PIC18F8722 最大的程式空間為128K-Byte

### ◆ 組合語言對周邊硬體較好控制

- MPLAB C18 提供了完整的周邊控制函數功能

### ◆ 組合語言對時序精確度較容易掌握

- MPLAB C18 提供了內建組合語言功能



## 如果有一設計案是這樣 ... C or 組合語言比較適合呢？

- ◆ 有一工業控制溫度為 4 ~ 20mA 的恆流源輸入，經12-bit 解析度A/D 來轉換，每次取樣後需計算出20次的移動平均值，其間需將最高與最低的值過濾掉忽略不計
- ◆ User 可透過 RS-232 設定最高、最低溫度點並存入外部的串列式 EEPROM (24LC02B)
- ◆ 輸出為五個控溫繼電器
  - 1 ON, 溫度超過最高設定點
  - 2 ON, 溫度介於兩點間的80% - 99%
  - 3 ON, 溫度介於兩點間的21% - 79%
  - 4 ON, 溫度介於兩點間的1% - 20%
  - 5 ON, 溫度低於最低設定點
- ◆ C 的流程控制容易, 如果有足夠的程式碼空間及價格彈性，使用 C 是較佳的選擇

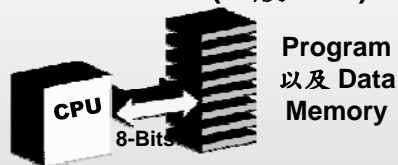


## PIC18F4520 主要架構



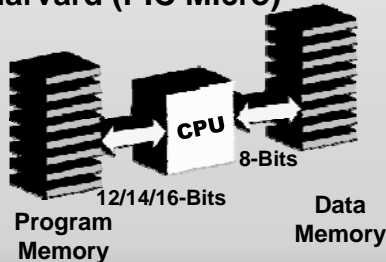
## PICmicro® 的架構

### Von Neumann (一般MCU)



- 經由相同的記憶體及匯流排來提取程式及存取資料
  - 指令與資料無法有效率的同時被處理
  - 運作效率受到此結構影響而變慢

### Harvard (PIC Micro)



- 使用兩個不同的記憶體空間與匯流排來存取程式及存取資料
  - 增加處理資料的效能與執行效率
  - 使得MCU可以具有不同寬度的程式記憶體與資料記憶體

© 2002 Microchip Technology Taiwan. All Rights Reserved.

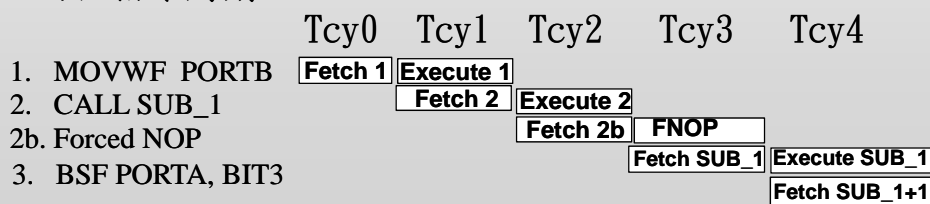
11

MPLAB-C18 Workshop



## PICmicro® 管狀式存取方式

- ◆ 對大部份的 MCU 而言，指令的提取與執行是連續發生的，但每一個指令周期只有一個動作發生。
- ◆ PICmicro 使用 Harvard 架構且使用 Pipeline 的運作模式，讓指令的提取與執行可同時進行。
- ◆ 指令的執行時間只需一個周期。
- ◆ 程式跳躍指令（例如：GOTO, CALL 或 BRA）則需要兩個指令周期。



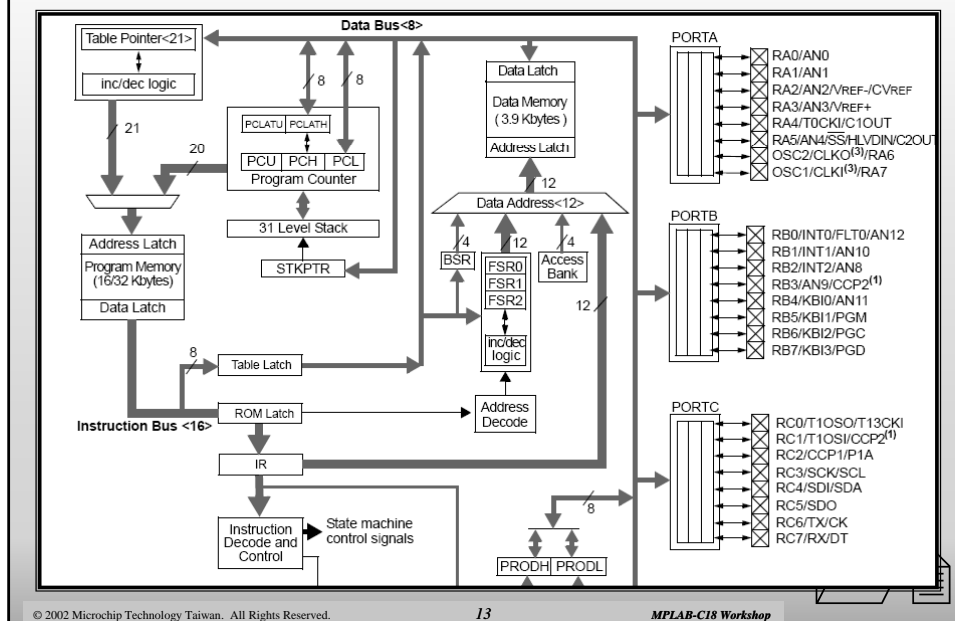
© 2002 Microchip Technology Taiwan. All Rights Reserved.

12

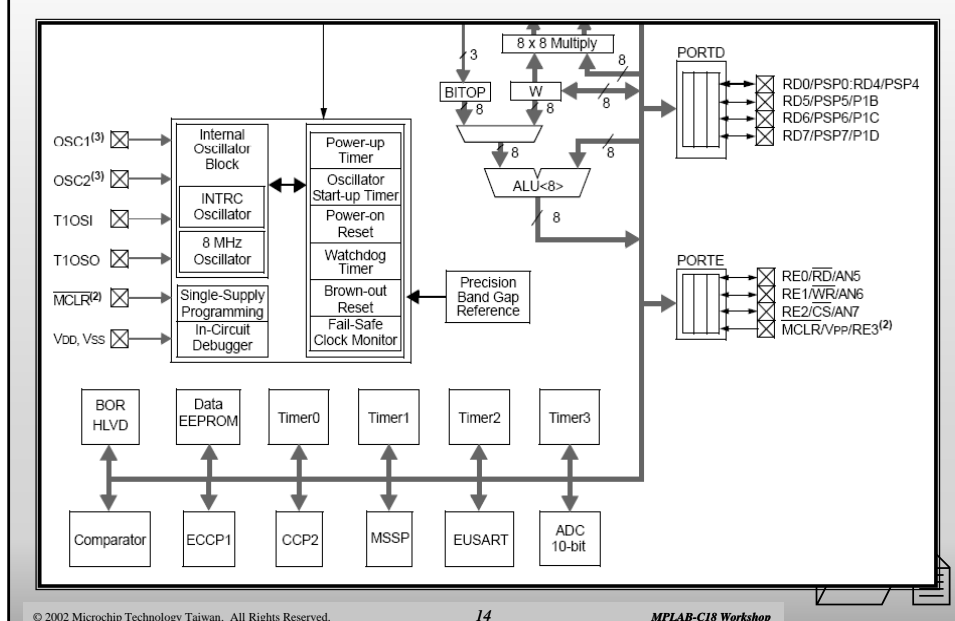
MPLAB-C18 Workshop



## PIC18F4520 架構圖(一)



## PIC18F4520 架構圖(二)



## PIC18F4520 主要周邊

- ◆ Flash Memory : 32K Bytes (10 萬次的重寫次數)
- ◆ Data RAM : 1536 Bytes
- ◆ EEPROM : 256 Bytes (100 萬次重寫能耐)
- ◆ Timer : 4 組
- ◆ CCP Module : x 1
- ◆ Enhanced CCP : x 1
- ◆ 串列通訊 : EUSART , MI<sup>2</sup>C , SPI
- ◆ 電壓比較器 : x 2
- ◆ AD轉換器 : 10-bit/13CH
- ◆ Reset : POR , BOR , WDT , MCLR , Stack ,  
Reset 指令
- ◆ I/O : 最多 36



## PIC18F4520 主要周邊

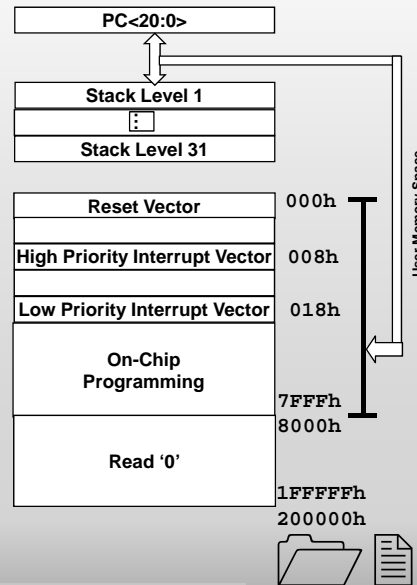
- ◆ 振盪模式
  - HS、XT、LP、HSPLL、EC 及 External RC
  - 內建 RC 震盪 31KHz ~ 8MHz
    - ✓ 可使用 x4 PLL
- ◆ Clock Failure Monitoring
- ◆ Power Done Mode
  - Run、Idle、Sleep Mode





## PIC18F4520 程式位置配置

- ◆ PIC18Fxxxx 最大可達2M Bytes 的程式定址能力；採線性定址模式，無需切換程式頁(Page)
- ◆ PIC18F4520 的程式存取空間為 32K Bytes。
- ◆ Reset 位址 = 0x000000
- ◆ 高優先權中斷向量 = 0x0008
- ◆ 低優先權中斷向量 = 0x0018
- ◆ 硬體堆疊共31層



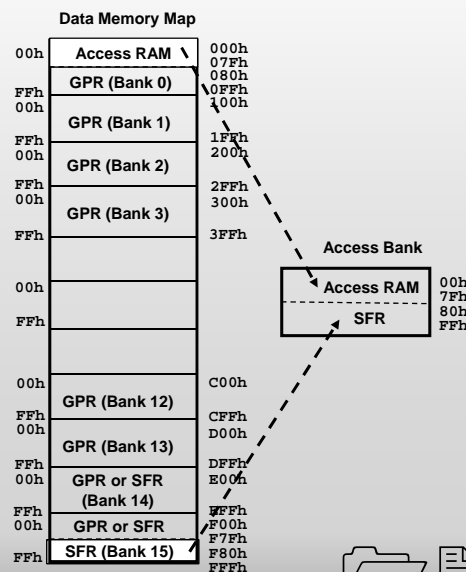
© 2002 Microchip Technology Taiwan. All Rights Reserved.

17

MPLAB-C18 Workshop

## PIC18F4520 資料記憶空間配置

- ◆ 共有16 banks 的Data Memory 每個 Bank 有256 Bytes。
- ◆ Special Function Registers (SFRs) 被安排在 Bank 15
- ◆ 對 C 語言而言，變數無須考慮放在那一個Bank，連結器 (Linker)會自行安排。
- ◆ 變數當然也可以直接指定放置在Access Bank，加速變數存取速度。
- ◆ 變數在記憶體的位址是由連結器來設定的，基本上無須知道其實際位置，但須知道記憶體空間是否足夠。
  - .lkr 檔即用來讓 Linker 檢查記憶體範圍的依據



© 2002 Microchip Technology Taiwan. All Rights Reserved.

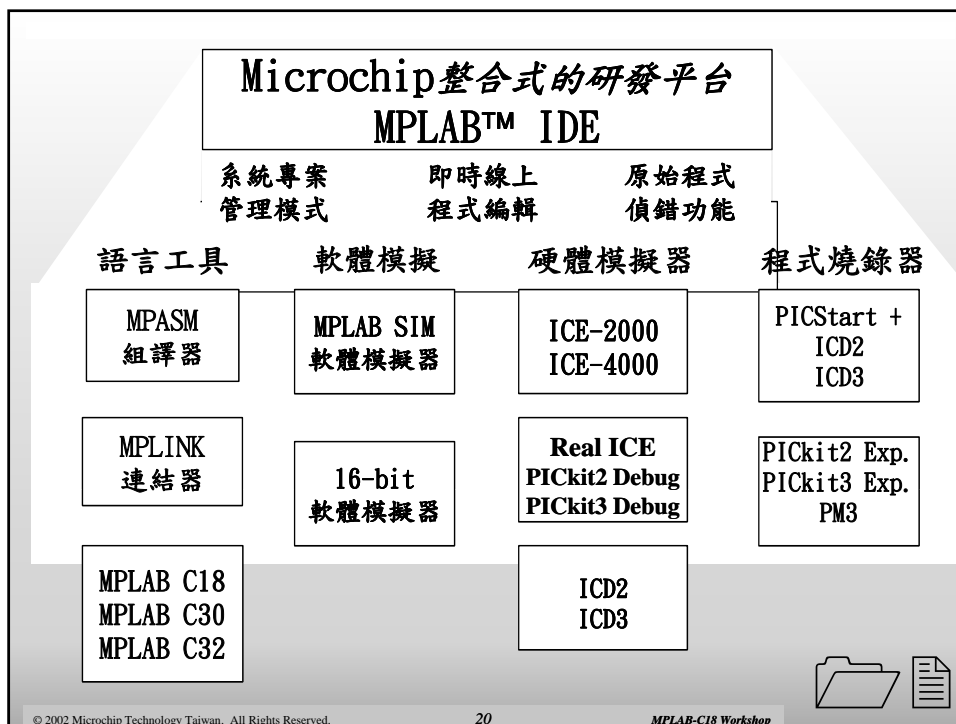
18

MPLAB-C18 Workshop

認識  
MPLAB IDE & MPLAB C18



© 2002 Microchip Technology Taiwan. All Rights Reserved. 19 MPLAB-C18 Workshop



## MPLAB C18 概述 (1)

- ◆ 與 ANSI '89 相容
- ◆ 支援浮點運算
  - 32-bit浮點運算
  - 使用IEEE浮點運算格式
- ◆ 原始程式除錯(Source Level Debugging)
  - 與MPLAB IDE系統相容
- ◆ MPLAB-C18支援之周邊控制函數庫
  - SPI, I2C, USART, A/D, Timer, .... etc.
  - 提供原始程式可供修改



## MPLAB C18 概述 (2)

- ◆ 支援軟體堆疊
  - 使用RAM空間做為軟體堆疊
  - 參數及引數之傳遞
  - 函數呼叫與中斷使用硬體堆疊
- ◆ 內建嵌入式組合語言
  - In-Line Assembly
- ◆ 程式裡可直接呼叫組合語言
- ◆ 可直接處理中斷服務程式
- ◆ 可將變數直接指定到 Access Bank  
以提高執行效率



## 介紹 MPLINK

### ◆ 什麼是MPLINK?

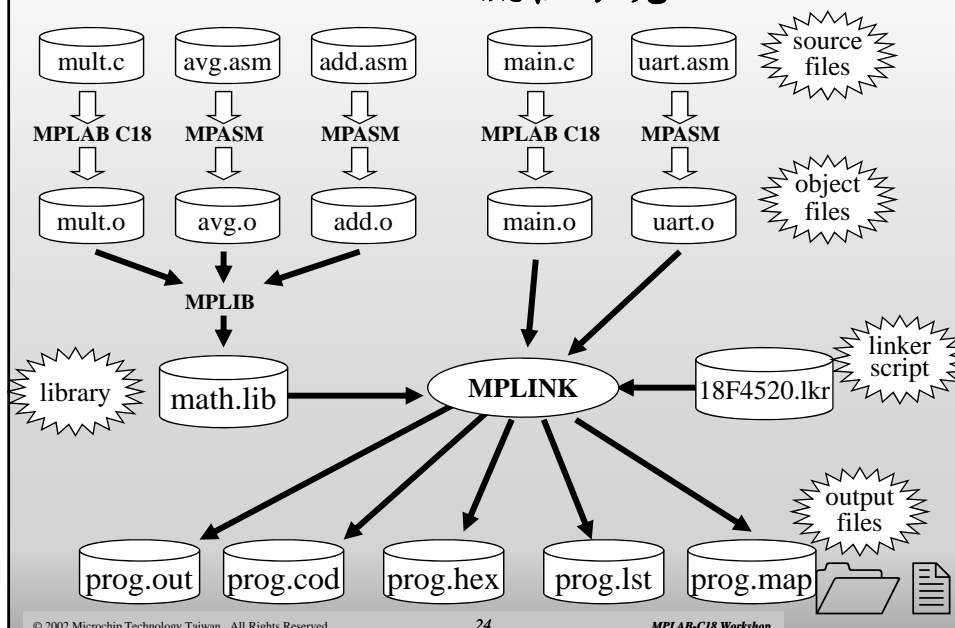
- MPLINK 是將組合語言的組譯(Assembler) 或 C- Compiler 所產生的 obj 檔加以連結並排定各程式及變數位址後，輸出一個可執行的 hex 檔

### ▲ MPLINK 能作做什麼？

- ▲ 安排實際位址給程式(CODE)及資料(RAM)
- ▲ 產生可執行檔 (HEX)
- ▲ 安排堆疊位址及深度給 MPLAB C18
- ▲ 提供 COD 檔以利程式偵錯
- ▲ 讓程式更容易模組化
- ▲ 連結資料庫 (Library)



## MPLINK 流程方塊



## 練習 1

1. 認識 MPLAB IDE
2. 編輯 C 原始程式
3. 設定使用環境
4. 編譯程式與除錯



## 專案管理 ( Project Management )

- ◆ MPLAB IDE 是採專案管理方式來完成軟體研發與設計，所以在使用 MPLAB IDE 時，就必需建立一個 Project。
- ◆ Project 的附加檔案名稱是 xx.mcp，最好與原始檔案放在同一個目錄以方便管理。
- ◆ 一個 Project 可記錄眾多資訊：
  - 視窗位置、大小、個數
  - 相關檔案的名稱、位置
  - 相關偵錯訊息的設定值
  - 組譯器、編譯器的選擇與設定
- ◆ 以 Project 觀念來保持一個完整的軟體設計，以便日後程式的維護、修改。





## 本文檔的編輯（EDIT）

- ◆ 檔案管理（File）
  - 開啓新檔 (New File)
  - 開啓舊檔 (Open File)
  - 檔案儲存同一檔名 (Save)
  - 檔案儲存另一檔名 (Save As)
- ◆ 游標控制
  - Home - 游標移至本行起頭
  - End - 游標移至本行尾
  - Ctrl + Left - 向左移一個字
  - Ctrl + Right - 向右移一個字
  - Ctrl + Pg Up - 移至本頁起頭
  - Ctrl + Pg Dn - 移至本頁尾
  - Ctrl + Home - 移至本檔案起頭
  - Ctrl + End - 移至本檔案尾
- ◆ 編輯控制
  - Ctrl + Z - 回復原狀
  - Ctrl + C - 複製選擇
  - Ctrl + V - 貼上複製
  - Ctrl + X - 清除選擇
  - Ctrl + A - 全選
  - Del - 刪除
  - Ctrl + G - 跳至第幾行
  - Ctrl + F - 尋找字元或字串
  - Ctrl + H - 替換所尋找字元或字串
  - F3 - 繼續向下尋找
  - Shift + F3 - 繼續向上尋找
  - Mouse 右鍵按兩下 - 選擇該字



## 輸入第一個程式 Ex1.C

```
#include <pl8cxxx.h>

int a;
int b;

/******
/*      Program Main ( )
/******
void main(void)
{
    int i;

    a = 2;
    b = 0;

    for (i=0 ; i<=100 ; i++)
    {
        b = b + a * i;
    }

    while(1);
}
```

1. 在MPLAB畫面下點選“File”後再點選“New”以開啓原始程式編輯視窗。
2. 開始輸入左邊的程式，並利用下一張所列的方法來進行程式編輯。
3. 輸入完成後，點選“File”後再點選“Save As”並輸入如下所示之檔名並儲存。
4. C:\RTC\C18\_V2\Answer\_APP001V3\Ans1\ex1.c

## 啓動 MPLAB® IDE

### 步驟 1

目前的版本

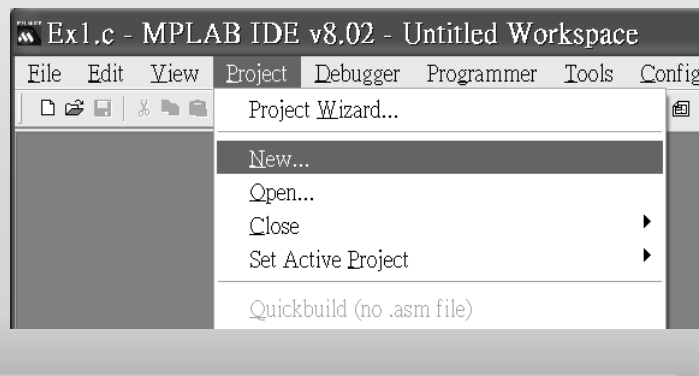


所選定的元件

## 建立一個 New Project

步驟 2

- ◆ 建立一個 New Project 的步驟：
  - Project → New 開啟建立 Project 視窗
  - 輸入 Project Name 及 Project 的路徑



© 2002 Microchip Technology Taiwan. All Rights Reserved.

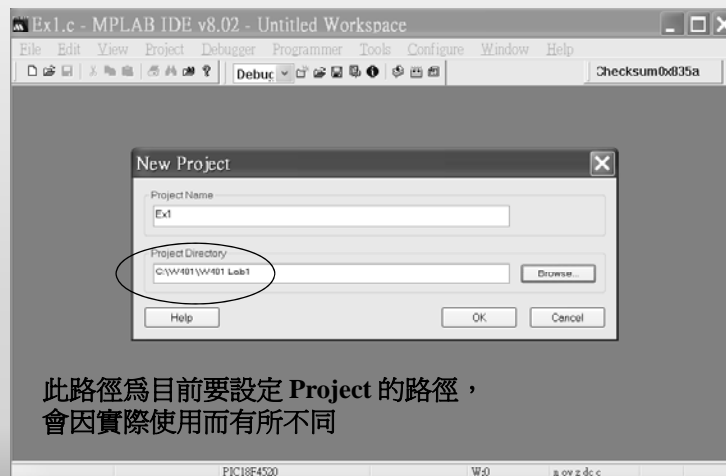
31

MPLAB-C18 Workshop

## 輸入 New Project 的名稱

步驟 3

- Project Name 為 : Ex1
- 路徑為 : ..\Ans1



此路徑為目前要設定 Project 的路徑，  
會因實際使用而有所不同

© 2002 Microchip Technology Taiwan. All Rights Reserved.

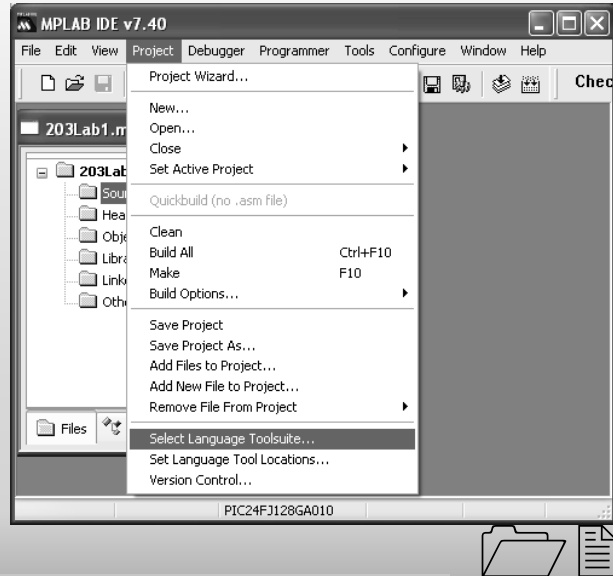
32

MPLAB-C18 Workshop



## 設定語言工具 C18, MPASM, 或其它 步驟 4

MPLAB IDE 是不  
會知道你要使用的  
語言工具。  
所以要正確的設定  
所使用的語言工具  
為何? 及其編譯程  
式的路徑與程式名  
稱。

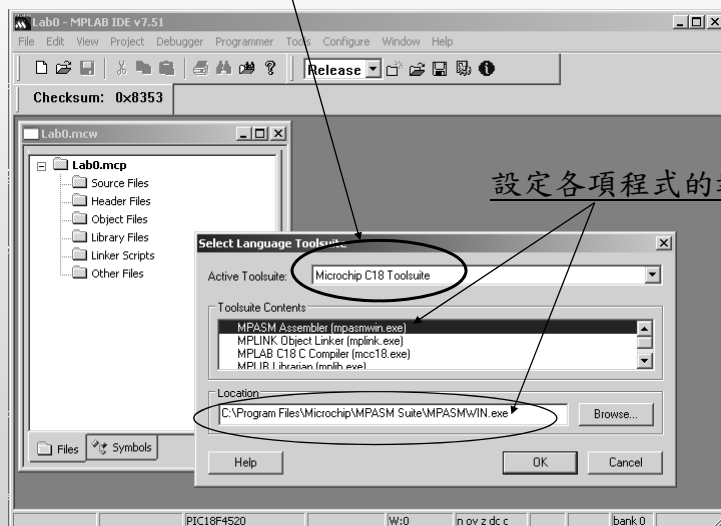


© 2002 Microchip Technology Taiwan. All Rights Reserved.

33

MPLAB-C18 Workshop

## 選擇 MPLAB C18 為語言工具 步驟 5



設定各項程式的執行路徑

註: mplink.exe, mcc18.exe & mplib.exe 的路徑位於  
c:\mcc18\bin (C18 的預設路徑)

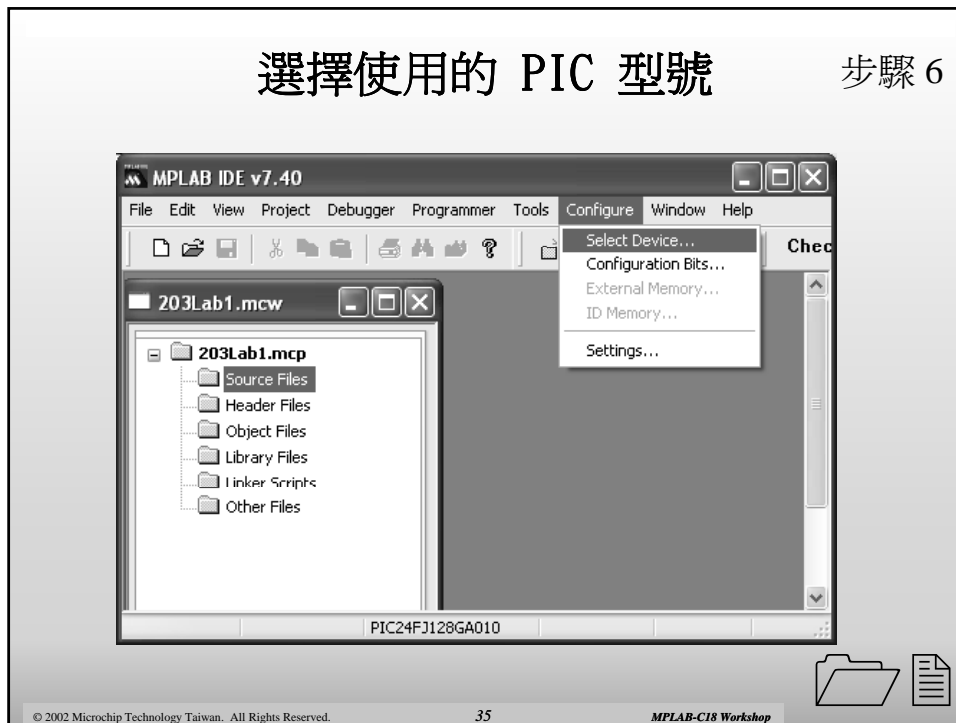
© 2002 Microchip Technology Taiwan. All Rights Reserved.

34

MPLAB-C18 Workshop

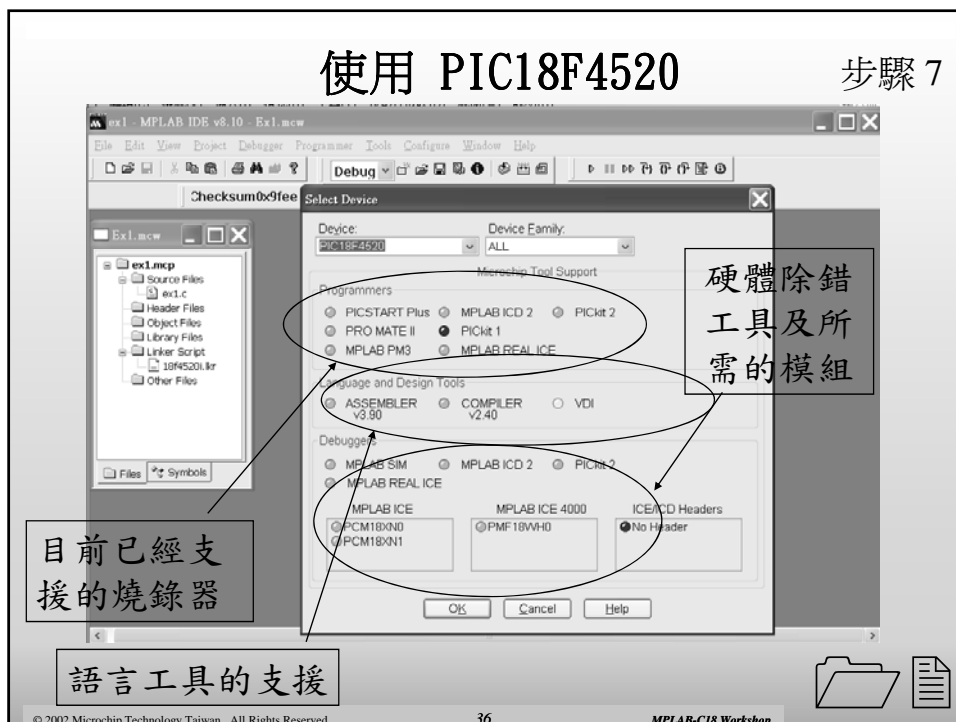
## 選擇使用的 PIC 型號

步驟 6



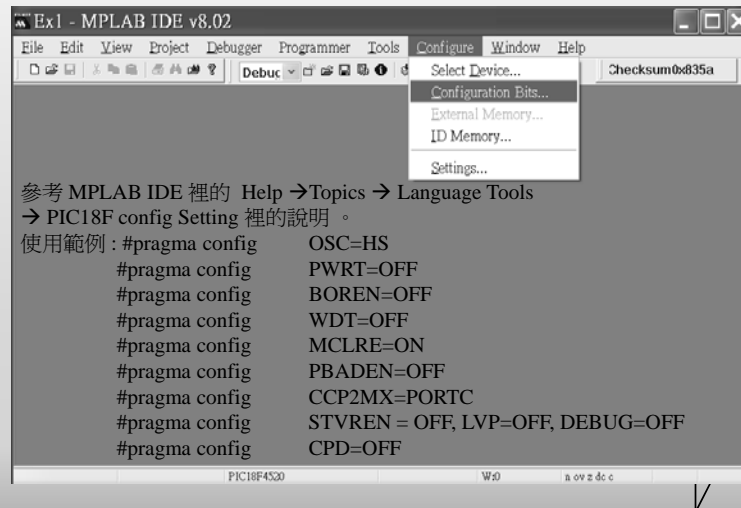
## 使用 PIC18F4520

步驟 7



## 設置 Configuration Bits 步驟 8

假如你要使用手動方式設定 Configuration Bits 可以透過底下的視窗對話來設定。強烈的建議在程式裡使用 **#pragma config** 的巨集來設定

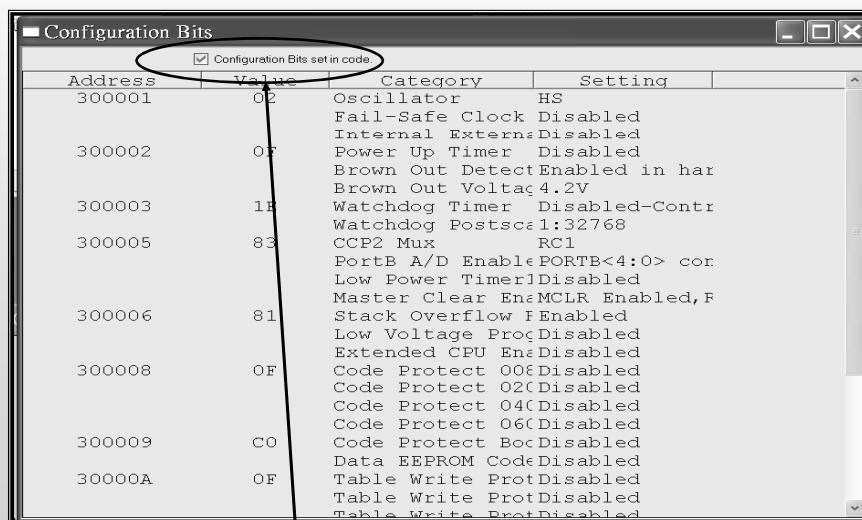


© 2002 Microchip Technology Taiwan. All Rights Reserved.

37

MPLAB-C18 Workshop

## 先檢查一下 config 的設定 步驟 9



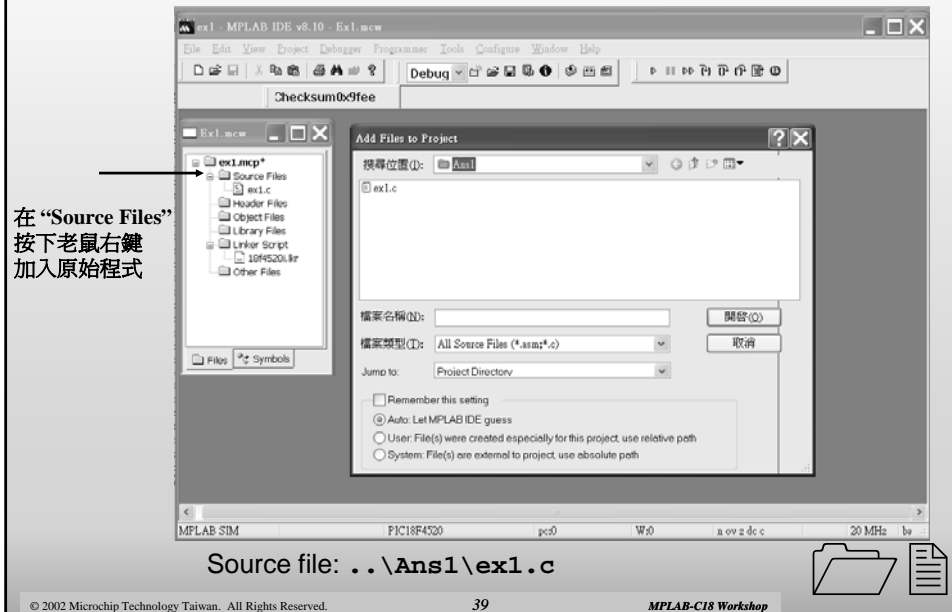
打勾拿掉後才可以用手動方式修改

© 2002 Microchip Technology Taiwan. All Rights Reserved.

38

MPLAB-C18 Workshop

## 將原始檔案加入目前的 Project 步驟 10



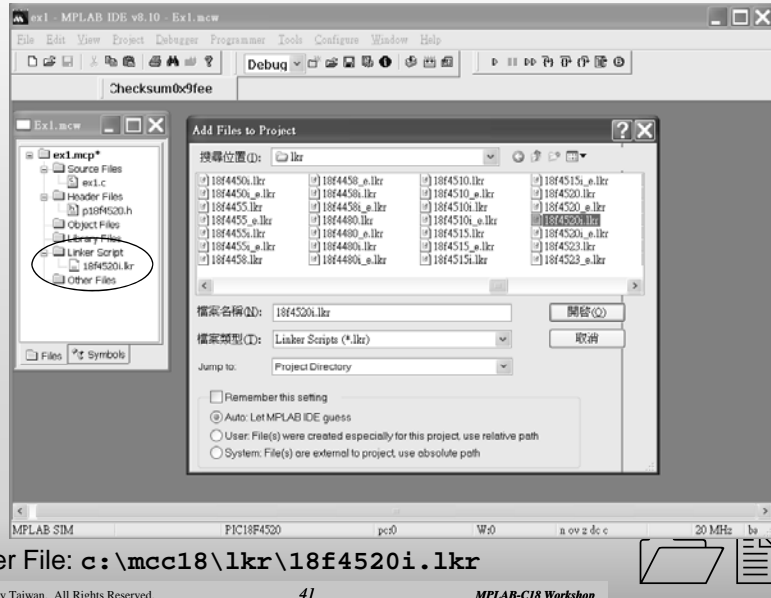
## 加入該零件的 H 檔案 步驟 11



## 加入連結描述檔 (LKR)

步驟 12

LKR 有數種不同的檔案使用時需注意不要弄錯：  
18f4520.lkr  
18f4520i.lkr  
18f4520\_e.lkr



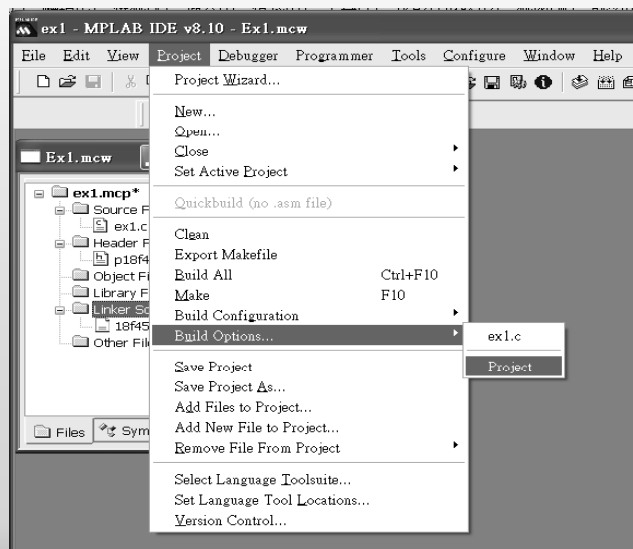
© 2002 Microchip Technology Taiwan. All Rights Reserved.

41

MPLAB-C18 Workshop

## 設定 Project 裡的 'Build' Options

步驟 13



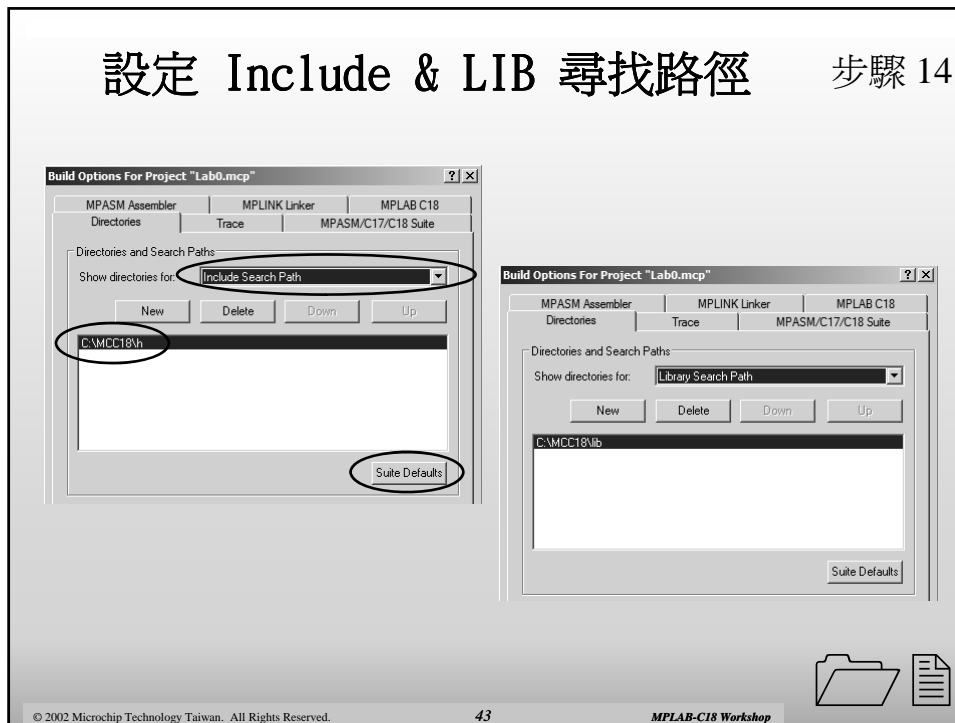
© 2002 Microchip Technology Taiwan. All Rights Reserved.

42

MPLAB-C18 Workshop

## 設定 Include & LIB 尋找路徑

步驟 14



## 產生 .map 及 HEX 輸出格式

步驟 15



## 編譯程式 (使用 Build All)

步驟 16



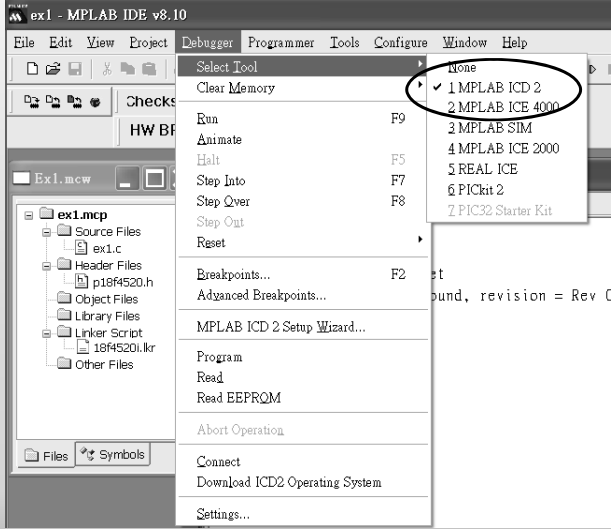
## 確定編譯成功

步驟 17



## 選擇 ICD2 作為除錯工具

步驟 18



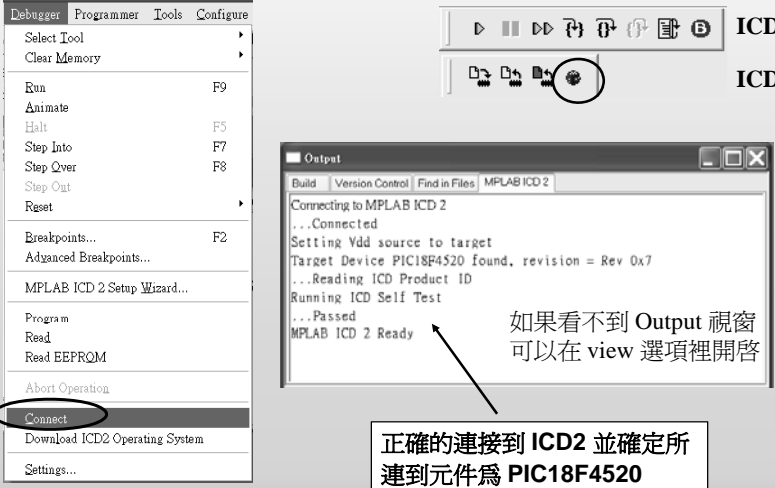
請先確定 ICD2 USB 的驅動程式已經正確安裝在 MPLAB IDE 下，安裝方法及路徑請參考：  
**C:\Program Files\Microchip\MPLAB IDE\ICD2\Drivers\clnrcd2.htm**

MPLAB ICD2 所支援的元件有數百種之多，當選擇 ICD2 時，MPLAB IDE 會自動檢查 F/W 的版本自行更新軟體。

© 2002 Microchip Technology Taiwan. All Rights Reserved. 47 MPLAB-C18 Workshop

## 連接 ICD2 到 APP001

步驟 19



ICD2 除錯選項  
ICD2 燒錄選項

如果看不到 Output 視窗可以在 view 選項裡開啓

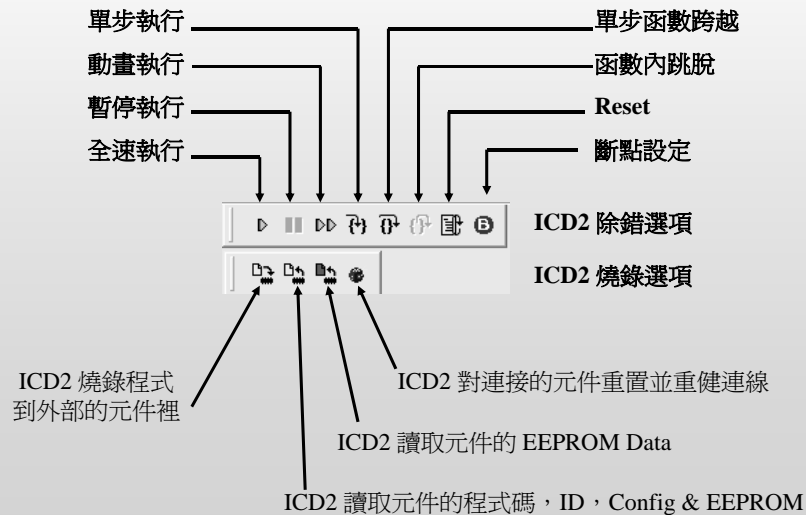
正確的連接到 ICD2 並確定所連到元件為 PIC18F4520

© 2002 Microchip Technology Taiwan. All Rights Reserved. 48 MPLAB-C18 Workshop



## 燒錄 HEX 檔到元件裡除錯

步驟 20



© 2002 Microchip Technology Taiwan. All Rights Reserved.

49

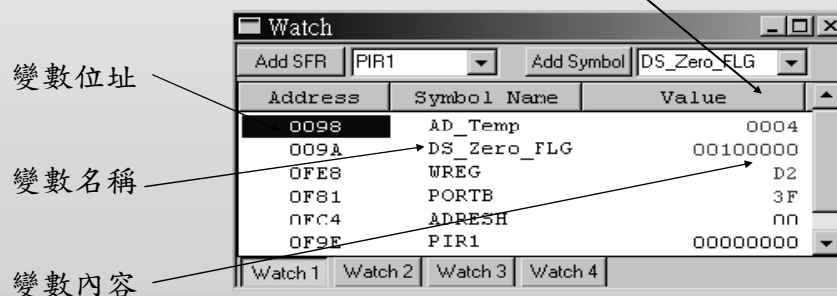
MPLAB-C18 Workshop

## Watch Window 的重要性

- ◆ 變數在 RAM 的位置是誰安排的
- ◆ 你知道怎樣才能看到變數的內容
- ◆ MPLAB IDE 有專屬的變數觀察視窗

➢ View → Watch

在這裡按老鼠右鍵可以設定顯示的格式

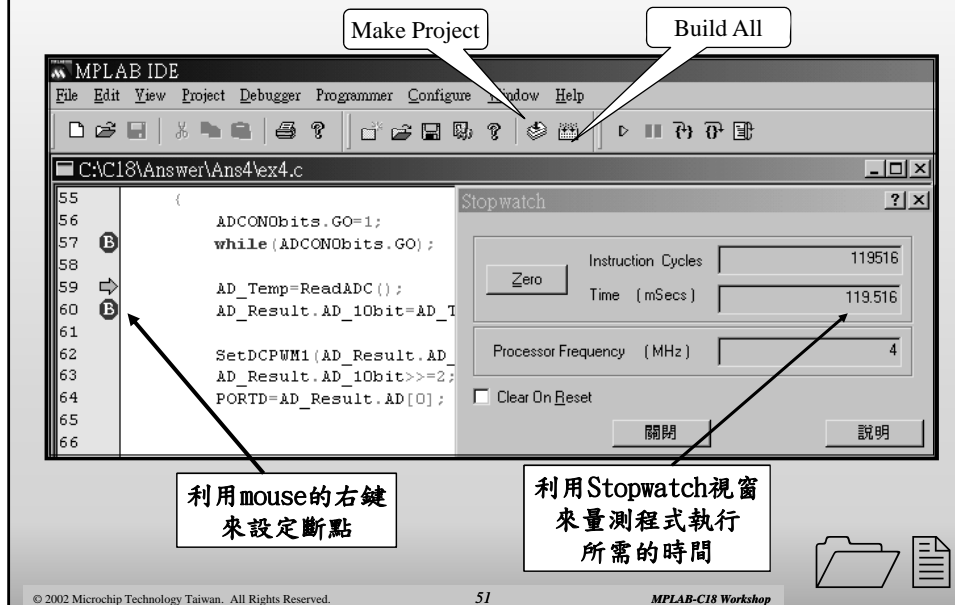


© 2002 Microchip Technology Taiwan. All Rights Reserved.

50

MPLAB-C18 Workshop

## 斷點設定及使用軟體模擬的計時器



## 編譯程式與除錯

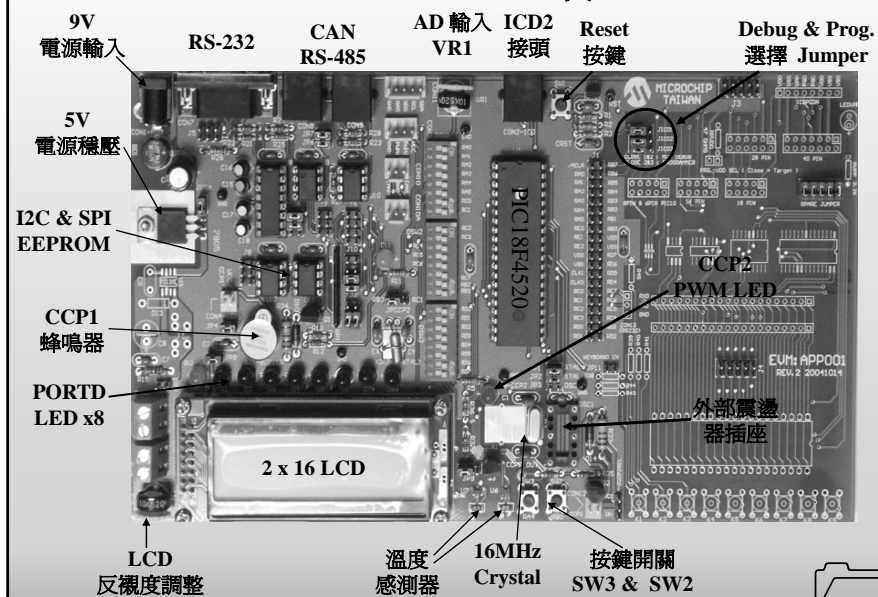
- ◆ 利用 Build All (Ctrl + F10) 來編譯程式
  - 如有錯誤，請參考錯誤訊息並加以修正
  - 在錯誤訊息處按滑鼠兩次即能輕易切換至錯誤行
- ◆ 利用 Watch Window 來觀察變數
- ◆ 利用 Mouse 的右鍵來設定中斷點
  - 熟悉 Reset, Run, Halt 功能
  - Step, Step Over 功能
  - RAM, SFR 視窗在 C 語言下就不是那麼重要了!
  - 其它的視窗？

## 第二章

### C 比組合語言更簡單



## 認識 APP001 實驗板



## 基本 C 的架構

```
#include    <p18C4520.h>  ← 處理器的定義檔頭

void function(void);      ← 函數原型宣告

void main(void)           ← 主程式函數
{
    /* User source code here */
}

void function(void)        ← 定義函數
{
    /* User function code here */
}
```



## 基本的運算子

### ◆ 基本運算順序

- 記住括號最優先
- 先乘除後加減
- 寫程式時不敢確定順序可用括號來輔助

#### ➢ 基本運算順序

- ✓ \* , /
- ✓ + , -
- ✓ << , >>
- ✓ < , <= , >=
- ✓ == , !=
- ✓ & , |
- ✓ && , ||

高

低

### ◆ 算術運算子

- \* , / , + , -
- x = - 5 + 2 \* 3 - 4

### ◆ 關係運算子

- > , < , == , !=
- if (a <= b)

### ◆ 邏輯運算子

- && , || , !
- (a && b) || c

### ◆ 位元運算子

- & , | , ^ , >> , <<
- a=(b & c) >> 6



## 指令 - #define

**#define      巨集名稱                  字串**

- ◆ 以一巨集名稱來代表一個字串，字串中可以是個常數、運算式，目的是提高程式的閱讀性
- ◆ C 編譯器會以巨集名稱代換字串後再進行編譯

例：定義 SW6 及 SW2 兩個按鍵開關

```
#define SW2 PORTAbits.RA4
#define SW6 PORTEbits.RE1

if (SW2 && SW6)
    debounce --;
else
    debounce =30;
```



## 指令 - for

**For (算式1；算式2；算式3 )**  
**{ 動作 }**

- ◆ 算式1:通常是起始值的設定，算式2:通常是條件判斷式，算式3:通常是步進運算式

例：讓 PORTD LED 閃五次，每次間隔 200 mS

```
for ( i=0 ; i <5 ; i ++ )
{
    PORTD = 0xff;           // LED on
    Delay10KTCYx(10 );     // Delay 100ms @4MHz
    PORTD = 0x00;           // LED off
    Delay10KTCYx(10 );     // Delay 100ms @4MHz
}
```



## 指令 - while

while (條件判斷式)

{ 動作 }

- ◆ 條件判斷成立時執行迴圈內動作，做完後又繼續跳回條件式作測試，直到條件式不成立為止。

例：

```
while ( PORTAbits.RA4 ); // Loop test until pressed SW2
```

```
PORTD = PORTD + 1;
```

```
debounce = 30;
```



## 指令 - IF ELSE

if (條件判斷)

{ 動作 1 } ---->條件判斷成立時執行

else

{ 動作 2 } ---->條件判斷不成立時執行

例：

```
if ( PORTBbits.RB0 == 1 ) // 判斷位元 RB0 = 1?
```

```
    PORTD = 0xff; // RB0 = 1, PORTD = 0xFF
```

```
else
```

```
    PORTD = 0x00; // RB0 = 0, PORTD = 0x00
```



## C18 的三個幫手

- ◆ 微控制器的標準名稱定義檔
  - p18f4520.h , p18f8720.h
- ◆ 微控制器的周邊位址設定檔
  - p18f4520.lib , p18f448.lib
- ◆ 微控制器的起動模組
  - Reset Vector 的控制權
  - 初始變數如何處理



## 18F4520 對周邊的定義

- ◆ 利用一種資料型態來定義其周邊的控制暫存器，這個資料型態就是“位元結構型態”
- ◆ 位元結構型態可視為將不同位元長度的資料收集在一個整體，並給予一個結構名稱
- ◆ 在位元結構名稱內會有許多成員，可利用結構變數與成員名稱的組合來指定結構中的某個成員
- ◆ 範例：利用結構型態來定義 PORTD 的程式

```
struct
{
    unsigned RD0:1;
    unsigned RD1:1;
    unsigned RD2:1;
    unsigned RD3:1;
    unsigned RD4:1;
    unsigned RD5:1;
    unsigned RD6:1;
    unsigned RD7:1;
} PORTDbits;
```



## MPLAB C18 如何獲得暫存器的位址定義 來自於p18F4520.o 對相關特殊功能暫存器的宣告

### ◆ p18F4520.o

- 定義周邊暫存器的位址給 C18 使用
- 附有 source code

### ◆ p18F4520.h

- 定義周邊暫存器及相關位元名稱

### ◆ p18F4520.inc

- 是給組合語言使用的周邊暫存器定義含入檔，不可混淆

```
LIST P=18F4520
NOLIST
;-----
; $Id$
; MPLAB-Cxx PIC18F4520 processor definition module
;
; (c) Copyright Microchip Technology,
;-----
SFR_UNBANKED0 UDATA_ACS      H'F80'

PORTA
PORTAbits    RES 1           ; 0xF80
PORTB
PORTBbits    RES 1           ; 0xF81
PORTC
PORTCbits    RES 1           ; 0xF82
PORTD
PORTDbits    RES 1           ; 0xF83
PORTE
PORTEbits    RES 1           ; 0xF84
RES 4

LATA
LATAbits     RES 1           ; 0xF89
LATB
LATBbits     RES 1           ; 0xF8A
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

63

MPLAB-C18 Workshop



## P18F4520.H 定義檔

### ◆ 定義18F4520 特殊功能暫存器 (SFR)的名稱及相關位元名稱

- 位置：  
c:\mcc18\h

### ◆ 也定義一些常用的巨集

- Nop()
- ClrWdt()
- Sleep() ...

### ◆ 一般放在程式前面，用 #include 將其代入

- #include <p18f4520.h>

### ◆ 使用 #include<p18cxxx.h> 來提高程式的可攜性

### 有關ADCON0暫存器之定義

```
extern near unsigned char ADCON0;
extern near union {
    struct {
        unsigned ADON:1;
        unsigned :1;
        unsigned GO:1;
        unsigned CHS0:1;
        unsigned CHS1:1;
        unsigned CHS2:1;
        unsigned ADCS0:1;
        unsigned ADCS1:1;
    };
    struct {
        unsigned :2;
        unsigned NOT_DONE:1;
    };
    struct {
        unsigned :2;
        unsigned DONE:1;
    };
    struct {
        unsigned :2;
        unsigned GO_DONE:1;
    };
};
} ADCON0bits ;
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

64

MPLAB-C18 Workshop





## P18F4520.H 定義檔的位置

- ◆ 一般放在程式前面，用#include將其代入
  - 例：#include <p18f4520.h>
- ◆ 但是 … 更換 CPU 時很麻煩
  - 需把所有模組中的 #include<> 敘述都改掉
- ◆ 比較好的做法是使用以下通用方法

- #include <p18cxxx.h>
- 因為 MPLAB IDE 會將 user 選擇的 CPU 資訊 pass 給 MPLAB C18，然後用 #if defined 檢查

```
#elif defined(__18F4520)
#include <p18f4520.h>
#elif defined(__18F4523)
#include <p18f4523.h>
..
..
```



## C18 的位元定址

- ◆ 所有的周邊暫存器及相關位元均在各含入檔(include file)中定義
  - 例：PIC18F4520的各種周邊是定義在 “ P18F4520.H ”
- ◆ 撰寫程式時，只要按照 Data Book 所標示的名稱使用。對應的位元結構為暫存器名稱加上 “bits”

1. 設定PORTD為輸出，並將 0x55 的16進制值輸出到 PORTD：

```
TRISD = 0 ;
PORTD = 0x55 ;
```

2. 將 PORTD 的 bit3 & bit5 設定為 1：

```
PORTDbits.RD3 = 1 ;
PORTDbits.RD5 = 1 ;
```



## P18F4520.lib 周邊函式庫檔

- ◆ 定義18F4520所有的特殊功能暫存器(SFR)的位址
  - P18F4520.o 是以obj的型態存在於 P18F452.LIB
  - 原始程式為組合語言型態放在
    - ✓ c:\mcc18\src\proc\p18F4520.asm
- ◆ P18F4520.LIB 提供各周邊控制函式
  - 相關的周邊控制函數，各自分門別類的以 obj 型態存在於 PIC18F4520.LIB
  - 周邊控制函數請參閱 MPLAB C18 Reference Manual
    - ✓ A/D , USART , Timer x , EEPROM ... 等
  - 原始程式在 c:\mcc18\src\pmc 的目錄下



© 2002 Microchip Technology Taiwan. All Rights Reserved.

67

MPLAB-C18 Workshop

## MPLAB C18 啟動模組

- ◆ 啟動模組的功能(一定要用)
  - 掌管最初的執行、規劃(Power-On Reset)
  - 規劃軟體堆疊區
  - 設定變數初始值
  - 轉移控制權到main( )函數
- ◆ 啟動模組有三個，放在 clib.lib 函數庫裡
  - c018.o - 無須為變數規劃初始值時使用
  - c018i.o - 須為變數規劃初始值時使用
  - c018iz.o - 先將RAM清除，再為變數規劃初始值
    - ✓ 原始程式位置：\mcc18\src\traditional\startup



© 2002 Microchip Technology Taiwan. All Rights Reserved.

68

MPLAB-C18 Workshop

## C018i.c 啟動模組程式

```
#pragma code _entry_scn = 0x000000
static void
entry (void)
{ _asm goto _startup _endasm }
```

RESET 位址:  
0x000000

```
#pragma code _startup_scn
static void _startup (void)
{
    _asm
    // Initialize the stack pointer
    LFSR 1, _stack LFSR 2, _stack CLRFB TBLPTRU, 0
    // Initialize rounding flag for floating point libs
    BSF FPFLAGS, RND, 0
    _endasm
```

給予 STACK 及  
TBLPTRU 初使值

```
_do_cinit ( );
```

設定初始變數值

```
loop:
// Call the user's main routine
main ( );
goto loop;
} /* end _startup() */
```

將控制權交至 main()

© 2002 Microchip Technology Taiwan. All Rights Reserved.

69

MPLAB-C18 Workshop

## 在 18F4520.1kr 描述檔

```
// Sample linker command file for 18F452
// $Id: 18f452.1kr,v 1.3 2002/07/29 19:09:08 sealep Exp $
```

```
LIBPATH .
```

連結啟動模組

```
FILES c018i.o
```

連結標準函數庫

```
FILES clib.lib
```

```
FILES pl8f4520.lib
```

連結18F4520周邊函數庫

CODEPAGE	NAME=vectors	START=0x0	END=0x29	PROTECTED
CODEPAGE	NAME=page	START=0x2A	END=0x7FFF	
CODEPAGE	NAME=idlocs	START=0x200000	END=0x200007	PROTECTED
CODEPAGE	NAME=config	START=0x300000	END=0x30000D	PROTECTED
CODEPAGE	NAME=devid	START=0x3FFFFE	END=0x3FFFFFFF	PROTECTED
CODEPAGE	NAME=eedata	START=0xF00000	END=0xF000FF	PROTECTED

ACCESSBANK	NAME=accessram	START=0x0	END=0x7F
DATABANK	NAME=gpr0	START=0x80	END=0xFF
DATABANK	NAME=gpr1	START=0x100	END=0x1FF
DATABANK	NAME=gpr2	START=0x200	END=0x2FF
DATABANK	NAME=gpr3	START=0x300	END=0x3FF
DATABANK	NAME=gpr4	START=0x400	END=0x4FF
DATABANK	NAME=gpr5	START=0x500	END=0x5FF
ACCESSBANK	NAME=accesssfr	START=0xF80	END=0xFFFF

```
STACK SIZE=0x100 RAM=gpr5
```

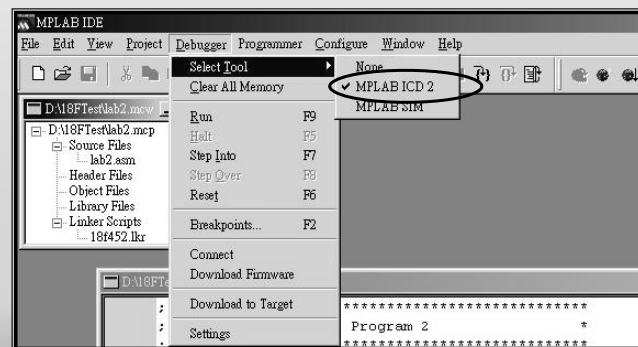
© 2002 Microchip Technology Taiwan. All Rights Reserved.

70

MPLAB-C18 Workshop

## 使用 MPLAB ICD2

- ◆ Debugger → Select Tool → MPLAB ICD2
- ◆ 詳細的 MPLAB ICD2 使用方法，參閱 MPLAB IDE v6.10 使用手冊（第五章）



© 2002 Microchip Technology Taiwan. All Rights Reserved.

71

MPLAB-C18 Workshop

## 練習 2

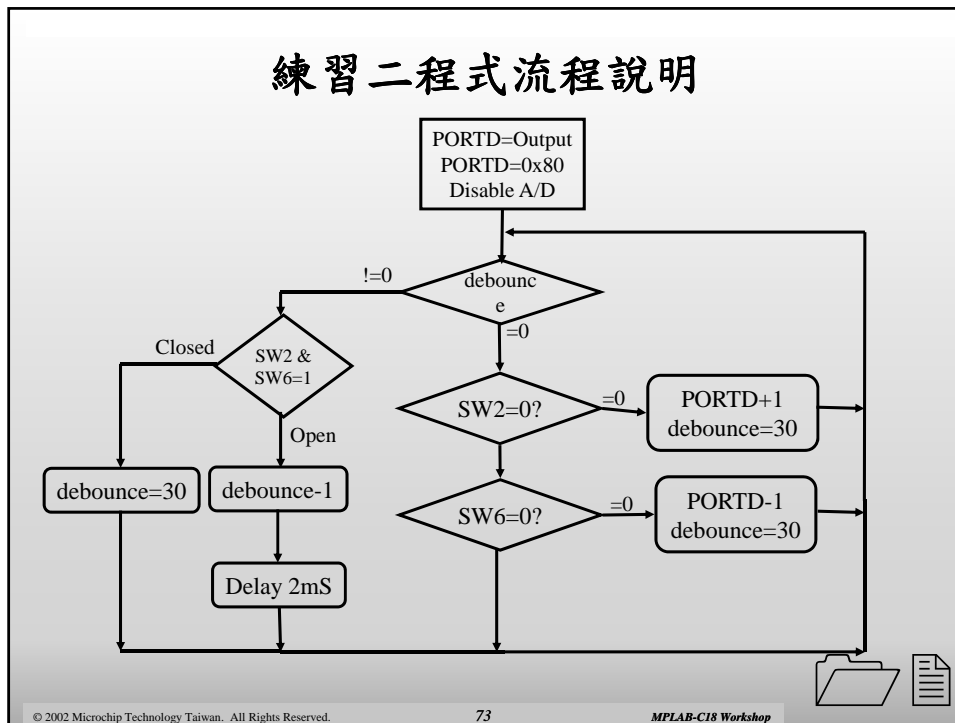
- ◆ 開啟  
” c:\RTC\C18\_V2\Answer\_APP001V3\Ans2\ex2.mcp”
- ◆ 將 PORTD (LED) 設定為輸出模式
- ◆ 程式開始執行時，LED 顯示0x80
- ◆ 按下 SW2，LED 自動加一
- ◆ 按下 SW6，LED 自動減一
- ◆ 利用for 迴圈來延遲及處理按鍵的彈跳
  - 通常機械式按鍵都會有數 ms 的彈跳現象，在此我們以軟體來過慮此種現象
- ◆ 參考下頁之流程圖

© 2002 Microchip Technology Taiwan. All Rights Reserved.

72

MPLAB-C18 Workshop

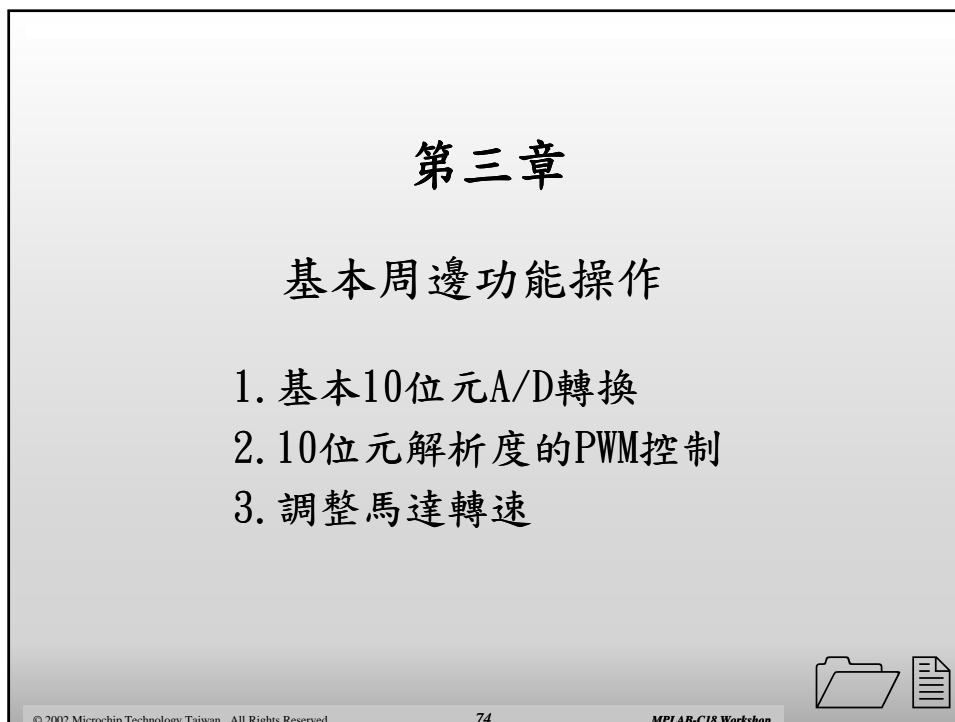
## 練習二程式流程說明



## 第三章

### 基本周邊功能操作

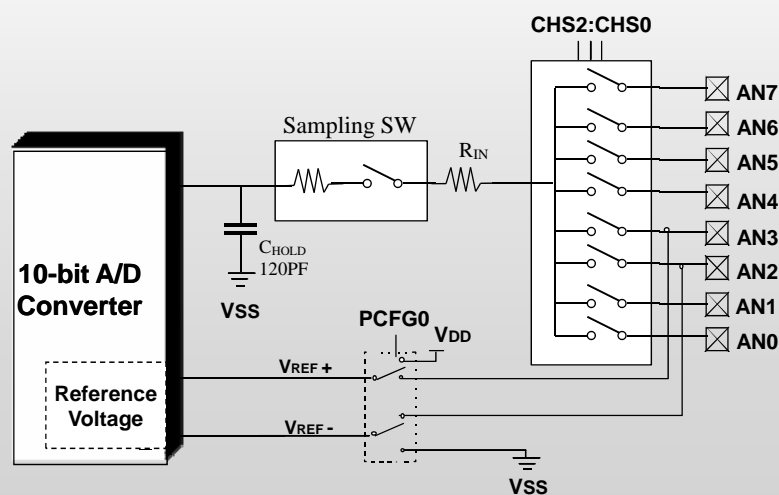
1. 基本10位元A/D轉換
2. 10位元解析度的PWM控制
3. 調整馬達轉速



## 10-bit A/D 轉換器

- ◆ 8組類比轉換多工輸入選擇，10 bits 解析度
- ◆ 類比輸入取樣時間：20  $\mu\text{S}$  (輸入阻抗 $<10\text{K}$ )
- ◆ 類比輸入轉換時間：19.2  $\mu\text{S}$  ( $12 T_{\text{AD}}$ )
- ◆ 10-bit 解析度時，只有一位元的誤差
- ◆ 允許使用外部參考電壓：VREF+ & VREF-
- ◆ 轉換的結果允許自動向左、向右對齊修正
- ◆ 完整的轉換時間共須 39.2  $\mu\text{s}$ 
  - 如輸入腳位固定，其轉換時間只需：29.2  $\mu\text{s}$

## 10-bit A/D 方塊圖



## 18F 系列 ADC 的主要差異

- ◆ 舊式的 ADC 必須自己控制取樣到轉換的時間
  - 例如 PIC18F452 , 只有 ADCON0 & ADCON1
- ◆ 較新的 ADC 可以設定取樣要多久 (以 TAD 為單位), 使用者只要切換 Channel 後啟動轉換, ADC 會先取樣 nTAD 後再轉換
  - PIC18F4520 有 ADCON0 , ADCON1 & ADCON2 !!
- ◆ 新的 ADC 可以有較快的 TAD
  - PIC18F452           TAD min = 1.6uS
  - PIC18F4520        TAD min = 0.7uS (Vref > 3.0V)
- ◆ 不包含取樣時間的轉換需時 12 TAD
  - PIC18F452 的轉換時間為 19.2 uS
  - PIC18F4520 的轉換時間只要 8.4 uS



© 2002 Microchip Technology Taiwan. All Rights Reserved.

77

MPLAB-C18 Workshop

## A/D 控制暫存器 - PIC18F452

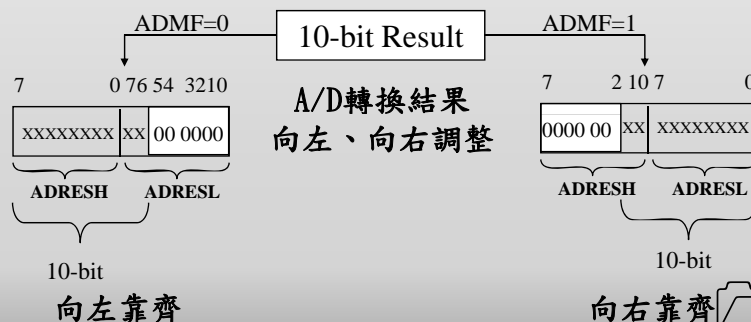
See Data Book

ADCON0 Register

ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	---	ADON
bit7							bit0

ADCON1 Register

ADFM	ADCS2	----	----	PCFG3	PCFG2	PCFG1	PCFG1
------	-------	------	------	-------	-------	-------	-------



© 2002 Microchip Technology Taiwan. All Rights Reserved.

78

MPLAB-C18 Workshop

## A/D 控制暫存器 - PIC18F4520 See Data Book

### ADCON0 REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON
bit 7							bit 0

### ADCON1 REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0 <sup>(1)</sup>	R/W <sup>(1)</sup>	R/W <sup>(1)</sup>	R/W <sup>(1)</sup>
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

### ADCON2 REGISTER

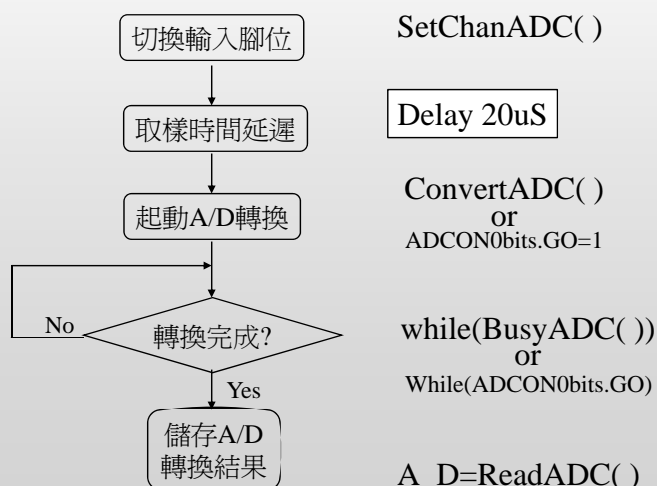
R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

© 2002 Microchip Technology Taiwan. All Rights Reserved.

79

MPLAB-C18 Workshop

## A/D 轉換基本流程



© 2002 Microchip Technology Taiwan. All Rights Reserved.

80

MPLAB-C18 Workshop



## A/D 轉換程式比較

### 組合語言

```
list    p=18f452
include <p18f452.inc>

main    clrfs    TRISD
        movlw    b'00001110'
        movwf    ADCON1
        movlw    b'10000001'
        movwf    ADCON0
Loop    call    Convert
        movwf    PORTD
        bra     Loop
;
Convert:
        call    Delay_20uS
        bsf     ADCON0,GO
        btfsc   ADCON0,GO
        goto    $-1
        movf    ADRESH,W
        return
```

### C 語言

```
#include <p18f4520.h>
unsigned char Convert(void);
void main(void)
{
    TRISD=0x00;
    ADCON1=0b00001110;
    ADCON0=0b10000001;
    while(1)
        PORTD=Convert();
}
unsigned char Convert(void)
{
    unsigned char i;
    for (i=0;i<5;i++);
    ADCON0bits.GO=1;
    while (ADCON0bits.GO);
    return ADRESH;
}
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

81

MPLAB-C18 Workshop



## 使用C18的A/D轉換函數

- ◆ 使用 MPLAB C18 提供的周邊函式庫時，要參考 Help File 中的說明
- ◆ 以 ADC 為例，可以在 C:\mcc18\doc 找到 AD Converter.htm
- ◆ 參考其中的內容，可以知到使用到的 MCU 之 ADC 數於哪一個 Version

Version name	Device number
ADC_V1	18C242, 18C252, 18C442, 18C452, 18F242, 18F252, 18F442, 18F452, 18F248, 18F258, 18F448, 18F458, 18F2439, 18F2539, 18F4439, 18F4539
ADC_V2	18C601, 18C801, 18C658, 18C858, 18F6620, 18F6720, 18F8620, 18F8720, 18F6520, 18F8520
ADC_V3	18F1220, 18F1320
ADC_V4	18F1230, 18F1330
PIC18F4520 爲 V5	18F2220, 18F2320, 18F4220, 18F4320, 18F2420, 18F2520, 18F4420, 18F4520, 18F2423, 18F2523, 18F4423, 18F4523, 18F2455, 18F2550, 18F4455, 18F4550, 18F2410, 18F2510, 18F2515, 18F2610, 18F4410, 18F4510, 18F4515, 18F4610, 18F2525, 18F2620, 18F4525, 18F4620, 18F6310, 18F6410, 18F8310, 18F8410, 18F6390, 18F6490, 18F8390, 18F8490, 18F6527, 18F6622, 18F6627, 18F6722, 18F8527, 18F8622, 18F8627, 18F8722, 18F6585, 18F6680, 18F8585, 18F8680, 18F6525, 18F6621, 18F8525, 18F8621, 18F2450, 18F4450, 18F2480, 18F2580, 18F4480, 18F4580, 18F2585, 18F2680, 18F4585, 18F4680, 18F2682, 18F2685, 18F4682, 18F4685, 18F2221, 18F2321, 18F4221, 18F4321
ADC_V5	

© 2002 Microchip Technology Taiwan. All Rights Reserved.

82

MPLAB-C18 Workshop



## 使用C18的A/D轉換函數

- ◆ 使用A/D轉換函數庫時，須同時使用定義檔“adc.h”

```
#include <adc.h>
```

OpenADC 根據不同的 MCU 有所區分, 以下為 V5 的 Prototype

```
void OpenADC(unsigned char config,unsigned char config2 ,  
             unsigned char portconfug)
```

- OpenADC( ADC\_FOSC\_32 & ADC\_RIGHT\_JUST & ADC\_4\_TAD,  
 ADC\_CH0 & ADC\_INT\_OFF & ADC\_REF\_VDD\_VSS,  
 ADC\_1ANA); (範例)

```
void ConvertADC (void)
```

- ConvertADC();

```
void SetChanADC (unsigned char channel)
```

- SetChanADC(ADC\_CH0);

```
int ReadADC (void)
```

- int result;  
➢ result = ReadADC( );



## 練習 3-1

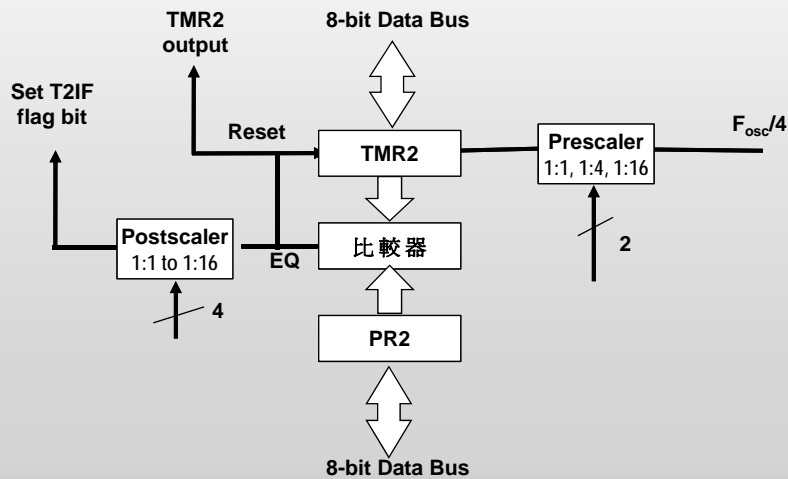
- ◆ 開啟

” c:\RTC\C18\_V2\Answer\_APP001V3\Ex3-1\ex3-1.mcp”

- ◆ 利用A/D函數庫來撰寫
- ◆ 將 PORTD (LED) 設定為輸出模式
- ◆ 設定AN0為A/D輸入端，其餘均設為一般I/O功能
- ◆ 調整VR以改變AN0的輸入電壓，並將其轉換結果向左調整後，取最高的八位元顯示在LED



## Timer 2 架構圖



© 2002 Microchip Technology Taiwan. All Rights Reserved.

85

MPLAB-C18 Workshop

## Timer 2

- ◆ 8-bit 採比較模式的計時器
- ◆ 有預除器及後除器之功能
- ◆ TMR2 為一可讀、寫並具有自動載入功能的計時器
- ◆ 計時器(TMR2)會自動加一並與設定的值(PR2)相比  
若相等則送出訊號至後除器或產生中斷，並自動  
將計時器(TMR2)清除為零，重新計時
- ◆ PWM 模式下，是 Period & Duty 的基本頻率源
- ◆ 也可以當成 MSSP (SPI™) 傳送速率的設定

© 2002 Microchip Technology Taiwan. All Rights Reserved.

86

MPLAB-C18 Workshop



## 利用VR控制PWM輸出(可控制 DC 馬達轉速)

```
void main(void)
{
    TRISD=0;                                // Set PORTD for LED output
    TRISCbits.TRISC2=0;                     // set CCP1(RC2) for PWM output
    InitializeAD();                          // Initial A/D module
    T2CON=0b00000100;                       // Timer2 ON
    CCP1CON=0b00001100;                     // Set PWM mode
    PR2=0xFF;                               // Set Max. Period

    while(1)
    {
        ConvertADC();                       // Convert A/D
        while(BusyADC( ));                  // Waiting A/D until done
        PORTD=ADRESH;                       // Put A/D result on LEDs
        CCP1L=ADRESH;                       // Set A/D result to Duty Reg.
                                            // with 8-bit Resolutions
    }
}
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

89

MPLAB-C18 Workshop



## 使用PWM & Timer 2 函數庫

- ◆ 使用PWM函數庫時，須同時使用定義檔“pwm.h”
- ◆ 使用Timer函數庫時，須同時使用定義檔“timers.h”

```
#include <pwm.h>
```

```
#include <timers.h>
```

```
void OpenTimer2 (unsigned char config)
```

```
    ✓ OpenTimer2 (TIMER_INT_OFF&T2_PS_1_4 & T2_POST_1_16);
```

```
void OpenPWM1(char period)
```

```
    ✓ OpenPWM1(0xFF); // 0xFF 為 PR2 暫存器的設定值
```

```
void SetDCPWM1 (unsigned int dutycycle)
```

```
    ✓ SetDCPWM1(1023);
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

90

MPLAB-C18 Workshop



## 練習 3-2

- ◆ 開啟

” c:\RTC\C18\_V2\Answer\_APP001V3\Ex3-2\ex3-2.mcp”

- ◆ 利用A/D、PWM及Timer2函數庫來撰寫
- ◆ 將 PORTD (LED) 設定為輸出模式
- ◆ 設定使用PWM1做為輸出(CCP1/RC2 接腳)
- ◆ 設定PWM的PR2為0xFF (Period 為最大值)
- ◆ 調整VR以改變AN0的輸入電壓，並將其A/D轉換結果直接控制馬達轉速，並且取出A/D最高的八位元顯示在LED

© 2002 Microchip Technology Taiwan. All Rights Reserved.

◆ LED 將因 PWM 的 Duty 變化而有不同亮度



## 第四章

### MPLAB C18 使用詳述

1. 資料型別
2. 變數形別
4. 運算元
3. 程式控制
4. 模組化設計
5. 指標與陣列
6. 結構的資料形態

© 2002 Microchip Technology Taiwan. All Rights Reserved.

92

MPLAB-C18 Workshop



## C18 資料型別



## MPLAB C18 的資料型別

### ◆ MPLAB C18 支援的整數資料型別

Type	Size	Minimum	Maximum
char <sup>(1,2)</sup>	8 bits	-128	127
signed char	8 bits	-128	127
unsigned char	8 bits	0	255
int	16 bits	-32,768	32,767
unsigned int	16 bits	0	65,535
short	16 bits	-32,768	32,767
unsigned short	16 bits	0	65,535
short long	24 bits	-8,388,608	8,388,607
unsigned short long	24 bits	0	16,777,215
long	32 bits	-2,147,483,648	2,147,483,647
unsigned long	32 bits	0	4,294,967,295

Note 1: A plain char is signed by default.

2: A plain char may be unsigned by default via the -k command-line option.



## MPLAB C18 的資料型別

### ◆ MPLAB C18 支援的浮點數資料型別

Type	Size	Minimum Exponent	Maximum Exponent	Minimum Normalized	Maximum Normalized
float	32 bits	-126	128	$2^{-126} \approx 1.17549435e-38$	$2^{128} * (2-2^{-15}) \approx 6.80564693e+38$
double	32 bits	-126	128	$2^{-126} \approx 1.17549435e-38$	$2^{128} * (2-2^{-15}) \approx 6.80564693e+38$



## 資料型別 (signed & unsigned)

### ◆ unsigned

- 通稱為“無號數”，即正整數。

### ◆ signed

- 通稱為“有號數”，即正、負整數。
- 一般是將“無號數”取 2 ‘補數

位元數	unsigned	signed
8	0x00	0
8	0x7F	127
8	0x80	-128
8	0xFF	-1





## 資料型別 (short & long)

- ◆ short ( 與 int 相同 )
  - 應用的場合為一 16-bit 的整數(正、負數)
- ◆ short long
  - 應用的場合為一 24-bit 的整數(正、負數)
  - 非常適合使用在 “*far rom*” 長指標運算
- ◆ long
  - 應用的場合為一 32-bit 的整數(正、負數)



## 資料儲存的順序

- ◆ 低位元組儲存在較低位址

long Var = 0xAABBCCDD ;

RAM Address	0x100	0x101	0x102	0x103
Content	0xDD	0xCC	0xBB	0xAA



## 記憶體模式 (near , far & rom , ram )

	程式記憶體區(rom)	資料記憶體區(ram)
far	2M bytes 定址模式 (用24-bit的指標)	4K bytes 定址模式 (用16-bit的指標)
near	<64K bytes定址模式 (用16-bit的指標)	Access RAM 定址模式 (用8-bit的指標)

注意：紅色字框為內定的記憶體模式 (Default)



## 記憶體模式 - 範例說明

```
const rom far char LCD_MSG1[]="PIC18F4520";
```

在16-Bit定址中擴展為24-bit位址存取

指定用程式記憶體

常數宣告 (不可變更)

```
int AD_Read;
```

宣告變數在 4K RAM的區域

宣告變數在 Access RAM

```
near int AD_Read;
```



## 常數的表示 (Const)

- ◆ 表示單一ASCII字元碼
  - 'a'、'A'、'\r' (=0x0d)、'\n' (=0x0a)
  - '\x41' (=0x41='A')、
- ◆ 二進制表示法
  - 0b10100101 (=0xA5)
- ◆ 八進制表示法
  - o101 (歐壹零壹=0x41='A')
- ◆ 十六進制表示法
  - 0x5A、0x3FFF
- ◆ 字串進制表示法
  - "How are you?"



## C18 變數類別



## 何謂變數

- ◆ 變數是用來存放資料的，資料內容會隨程式的執行而改變；就一般而言，變數存放於 RAM。
  - 變數的第一個字母必須是英文字，或是底線符號
  - 變數的字母有大小寫之分：
    - ✓ count，Count 與 COUNT 是不同的變數，切記！
  - 變數名稱內不可有“+”，“-”，“\*”，“/”，“.” .... 等符號
  - 變數必須宣告其型別，也可以直接指定初始值

```
int x,y,z;  
x=3;  
y=5;  
z=x*y;
```

```
int x=3,y=5,z;  
z=x*y;
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

103

MPLAB-C18 Workshop



## 變數的類別 (一) 區域變數 (local , auto)

- ◆ 區域變數是在函數內部所宣告的變數，其視野僅在本函數內，其它的函數無法使用。
- ◆ 區域變數的名稱在不同的函數內可相同。
- ◆ 區域變數的生命週期是函數被使用時開始存在，函數執行結束時消失。

```
Void main(void)  
{  
    unsigned char i=0;  
    i++;  
  
    func();  
}  
  
/* function call */  
void func(void)  
{  
    unsigned char i=0;  
  
    i--;  
}
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

104

MPLAB-C18 Workshop



## 變數的類別 (二) 公用變數 (Global)

- ◆ 公用變數是在函數以外的地方宣告，實際佔有記憶體變數。
- ◆ 只要是視野內的函式均可存取此變數。
- ◆ 若要在程式中使用其它檔案所宣告的公用變數時，可使用extern來達成。
- ◆ 公用變數若要讓許多Module 使用，可製作一個.h 的 header file
- ◆ 公用變數的生命週期永遠存在。

```
extern unsigned char KeyData;
unsigned int ADResult;

void Motor(void)
{
    if (KeyData == 0x80)
    {
        TRISCbits.TRISC2=0;
        ConvertADC();
        while(BusyADC());
        ADResult= ReadADC();
        SetDCPWM1(ADResult);
    }
    else
        TRISCbits.TRISC2=1 ;
}
```

## 變數的類別 (三) 靜態變數 (Static)

- ◆ 靜態變數的視野與區域變數是一樣的，只能在函數內部；但生命週期並不會隨函數執行結束而消失(保留下來)。
- ◆ 靜態變數的值存在於固定的記憶位址。
- ◆ 當該函數再次執行時，上次保留之變數值會繼續使用。
- ◆ 靜態變數若有初值，只有程式開始後會被設定一次。

```
Void main (void)
{
    unsigned char i;

    for (i;i<10;i++)
        AddOne( );
}

/** AddOne Function */
void AddOne(void)
{
    static unsigned char count=0;
    count++;
}
```

## 變數的類別 (四) 暫存器變數 (Register)

- ◆ 暫存器變數的視野數及生命週期和區域變數相同，暫存器變數的優點是運算速度快。
- ◆ MPLAB C18 本身並不提供暫存器變數的宣告功能，但 PIC18C452的記憶體均可視為一暫存器使用，所以此時全域變數相當於暫存器變數。



## 變數的類別 (五) 揮發性變數 (Volatile)

- ◆ 若變數的值不一定要經由程式來改變，變數本身會自行或隨外在因素而改變。則稱有 volatile (易揮發) 的性質
- ◆ 揮發性變數一般就是指某些特殊暫存器(詳細請參考 p18f4520.h)：
  - TMR0 , TMR1 ...
  - PC , PCL ...
  - EEDATA , ADCON0 ...
  - PORTA , PORTB ...

```
Unsigned char x,y;;
volatile unsigned char TMR0;

x=55;
y=x;

TMR0=0x00;
/*-----
   The compiler must read TMR0
   and can't use the 0x00 in
   its temporary variable
   since TMR0 increments with
   execution
   -----*/
y=TMR0;
```



## 變數的類別 (六) 自定型別(Typedef)

- ◆ Typedef 可用來創造自己的型別名稱或特定型別。
- ◆ 主要的目的：
  - 使用更符合應用的資料型別名稱或組合來讓程式更容易維護或閱讀
  - 可以將特定的 struct, union 設定為自定資料型別

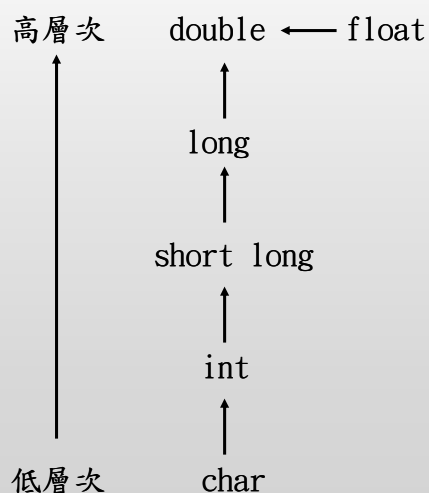
```
typedef unsigned char Byte;
typedef unsigned int Word;

void main(void)
{
    Word j[10];
    Byte i;

    for (i=0;i<10;i++)
        j[i] = (Word)i;
}
```



## 資料型別的轉換



- ◆ 兩不同層次的運算元相互運算時，層次低的會轉成較高層次的資料型態
- ◆ 在指定敘述中(變數=運算式)，運算式的結果會轉換成變數的資料型態後再執行指定運算(=)
- ◆ 強迫資料型態轉換
  - Char x;
  - Int Var;
  - Var = (int) x ;



## 運算元

1. 算數運算(Arithmetic)
2. 關係比較運算(Relational)
3. 邏輯比較運算(Logical)
4. 位元運算(Bitwise)
5. 指定運算(Assignment)
6. 加一、減一運算(Inc & Dec)



## 算數運算(Arithmetic)

運算符號	範例	說明
+	$a + b$	a 變數與 b 變數相加
-	$a - b$	a 變數減去 b 變數
*	$a * b$	a 變數與 b 變數相乘
/	$a / b$	a 變數除以 b 變數
%	$a \% b$	取 a 變數除以 b 變數的餘數

注意: +, -, \*, / 可使用在各種資料型態中  
% 只能用在整數型態中





## 關係比較運算(Relational)

運算符號	範例	說明
==	(a==b)	測試 a 是否等於 b
!=	(a!=b)	測試 a 是否不等於 b
>=	(a>=b)	測試 a 是否大於或等於 b
<=	(a<=b)	測試 a 是否小於或等於 b
>	(a>b)	測試 a 是否大於 b
<	(a<b)	測試 a 是否小於 b

注意：測試的結果是以 真(=1) 或 假(=0) 回傳



## 邏輯判斷運算(Logical)

運算符號	範例	說明
&&	a&& b	a 與 b 作邏輯AND運算，兩個為 1，結果為真(1)
	a    b	a 與 b 作邏輯OR運算，其中只要有1，結果為真(1)
!	! A	將 a 的結果反向 ( 0變1，1變0)

注意：在邏輯比較運算中，均是以真或假為判斷條件，並以真或假回覆判斷的結果。

何謂真：結果不等於零，回傳“1”  
何謂假：結果等於零，回傳“0”



## 位元運算(Bitwise)

運算符號	範例	說明
~	~a	將 a 的每一位元反相 (取 1' 補數)
&	a&b	a 和 b 的相對應位元作 AND 運算
	a   b	a 和 b 的相對應位元作 OR 運算
^	a^b	a 和 b 的相對應位元作 XOR 運算
>>	a>>b	將 a 向右移 b 位元
<<	a<<b	將 a 向左移 b 位元

注意：位元運算只能用在整數型態中



## 指定運算(Assignment)

運算符號	範例	說明
=	a=b	<b>a=b</b>
+=	a+=b	<b>a=a+b</b>
-=	a-=b	<b>a=a-b</b>
*=	a*=b	<b>a=a*b</b>
/=	a/=b	<b>a=a/b</b>
%=	a%=b	<b>a=a%b</b>
=	a =b	<b>a=a   b</b>
&=	a&=b	<b>a=a&amp;b</b>
^=	a^=b	<b>a=a^b</b>
>>=	a>>=b	<b>a=a&gt;&gt;b</b>
<<=	a<<=b	<b>a=a&lt;&lt;b</b>

例：

a \*= b + c

same as

a = a \* (b + c)



## 加一、減一運算 (Increment & Decrement)

◆ ++ 或 -- 可將變數  
自動加一或減一

- 運算後再加、減一
  - ✓ a++, b--
- 先加、減一後，再執行變數的運算
  - ✓ ++a, --b

```
Void main (void)
{
    unsigned char a=0;
    unsigned char b,c;

    a++;          /* same as ++a */
                  /* a=1 */

    b = 5 + a++;
    /* b=6 , a=2 */

    c = 6 + --a;
    /* c=7 , a=1 */
}
```

## 程序控制

1. 迴圈控制(Looping)
2. 條件判斷(Conditional)
3. 多向敘述(Switch Statement)
4. (Break Statement)
5. (Continue Statement)

## 迴圈控制(Looping)

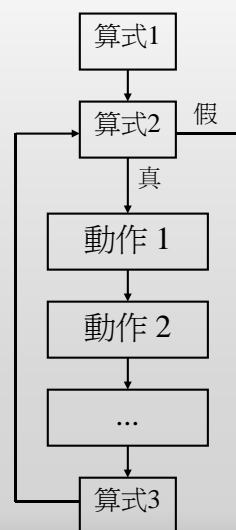
- ◆ 在C語言裡，共有三種迴圈控制
  - for (算式1;算式2;算式3)
  - while
  - do while
- ◆ 迴圈控制使用在重覆事件中，一般會加入條件判斷以離開此迴圈
- ◆ 迴圈控制可以讓程式更簡潔，易讀



## for 迴圈

### ◆ 格式:

```
for (算式1 ;算式2 ;算式3)
{
    動作 1;
    動作 2;
    :
    :
    :
}
```



## for 迴圈 - 範例

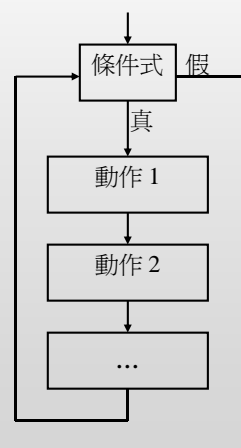
```
for( ; ; )  
for(i=0;1;i++)  
for(i=0;i<16;i++)  
for(y=1;(x=y)<10;Y++)  
for(i=1,j=6;i<25;i++,j--)  
for(x=0,y=1000;y>1;x++,y/=10)
```



## While 迴圈

### ◆ 格式:

```
while(條件式)  
{  
    動作 1;  
    動作 2;  
    :  
    :  
    :  
}
```



## while 迴圈 - 範例

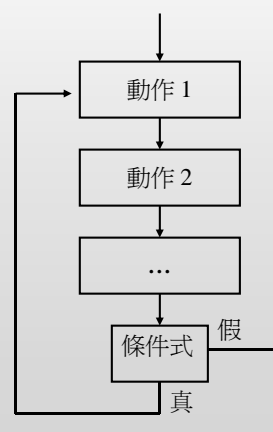
```
while(1)
while(i<0x34)
while(c != 'x')
while(BusyADC())
while(Rec_Data=='1')
```



## do-while 迴圈

### ◆ 格式:

```
do
{
    動作 1 ;
    動作 2 ;
    :
    :
} while (條件式);
```



## do While 迴圈 - 範例

```
Err=0;
i=0;

do
{
    while (Rec_Data!=0x00)
    {
        if(Temp_Code[i]!=Rec_Data) Err=1;

        Rec_Data=0x00;
        i++;
    }
} while (i<4);

if (Err!=1) Save_Password();
else {
    Display_Terminal(Error_Message);
    while(1);
}
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

125

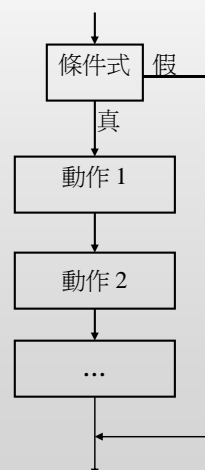
MPLAB-C18 Workshop



## if 敘述

### ◆ 格式:

```
if (條件式)
{
    動作 1;
    動作 2;
    :
    :
    :
}
```



© 2002 Microchip Technology Taiwan. All Rights Reserved.

126

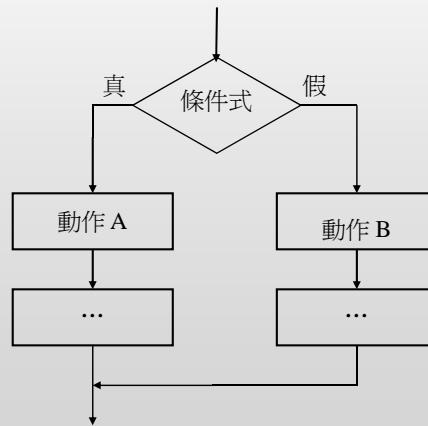
MPLAB-C18 Workshop



## if else 敘述

### ◆ 格式：

```
if (條件式)
{
    動作 A ;
    :
}
else
{
    動作 B ;
    :
}
```



## if else 敘述 - 範例

### ◆ 格式：

```
if (條件式1)
    if (條件式2)
        if (條件式3)
            {動作 A}
        else
            {動作 B}
    else
        {動作 C}
else
    {動作 D}
```

- ◆ 動作A：必須是條件1、2、3都成立時才會執行
- ◆ 動作B：必須是條件1、2成立，但條件3不成立時才會執行
- ◆ 動作C：必須是條件1成立，但條件2不成立時才會執行與條件3無關
- ◆ 動作D：必須是條件1不成立執行與條件2、3無關





## if else if 敘述 - 範例

### ◆ 格式:

```
if (條件式1)
    {動作 A}
else if (條件式2)
    {動作 B}
else if (條件式3)
    {動作 C}
else
    {動作 D}
```

- ◆ 動作A: 是條件1成立時執行
- ◆ 動作B: 必須是條件1不成立，但條件2成立時才會執行
- ◆ 動作C: 必須是條件1、2不成立，但條件3成立時才會執行
- ◆ 動作D: 必須是條件1、2、3都不成立時才會執行



## switch 敘述

### ◆ 格式:

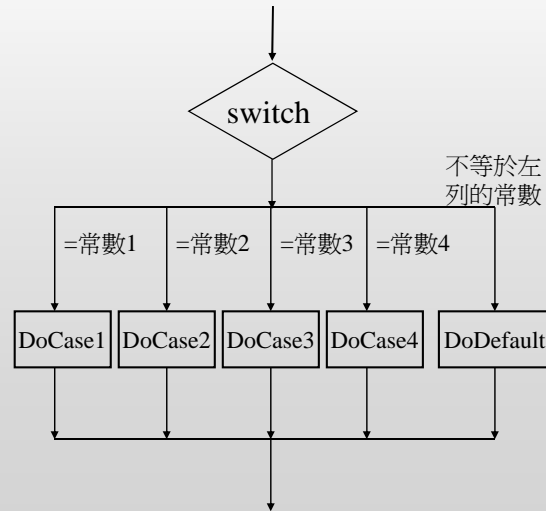
```
switch (判斷物)
{
    case 條件值1:
        {動作 1}
        break;
    case 條件值2:
        {動作 2}
        break;
    case 條件值3:
        {動作 3}
        break;
    default:
        {動作 4}
        break;
}
```

- ◆ switch 內的判斷物其結果必須為整數或字元
- ◆ switch是以判斷物來與各case的條件值比對；如果與某一case的條件值相等，則執行該case的動作
- ◆ 如果條件式與各case的條件值比對都不相等，則執行該default的動作
- ◆ 動作執行完畢可以用break跳出或繼續執行
- ◆ case後的條件值必須是常數，不可用變數，也不可重覆



## Switch + break 敘述 - 範例

```
switch (i)
{
    case 1:
        DoCase1();
        break;
    case 2:
        DoCase2();
        break;
    case 3:
        DoCase3();
        break;
    case 4:
        DoCase3();
        break
    default:
        DoDefault();
        break
}
```



© 2002 Microchip Technology Taiwan. All Rights Reserved.

131

MPLAB-C18 Workshop

## Switch 敘述 - 範例

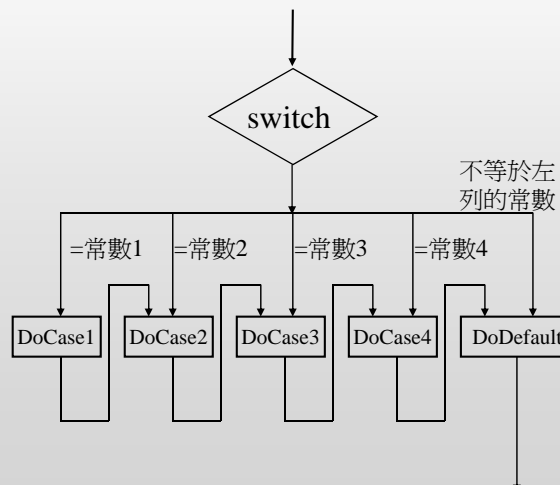
```
switch (i)
{
    case 1:
        DoCase1();

    case 2:
        DoCase2();

    case 3:
        DoCase3();

    case 4:
        DoCase3();

    default:
        DoDefault();
}
```



© 2002 Microchip Technology Taiwan. All Rights Reserved.

132

MPLAB-C18 Workshop

## break & continue 敘述

### ◆ Break

- 用來跳離本層所圍起來的區域，或是用來中斷並跳離迴圈(如：for、while、do ... while)
- 也可以用來中斷某種敘述的執行(如：switch)
- break只能用來跳離本層迴圈，如想跳離更多層的迴圈可以用goto

### ◆ Continue

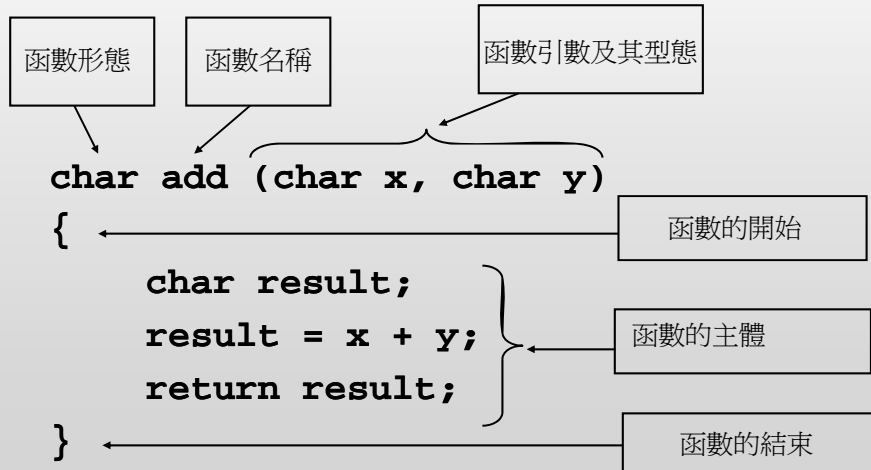
- 用來跳開迴圈主體的剩餘部份，而進行下一迴圈執行
- 對 while，do..while 而言，continue 敘述促使程式進入條件的測試，以決定是否再一次執行迴圈主體部份
- 對 for 敘述而言，continue 敘述促使程式進入迴圈運算式的執行，接著執行迴圈條件的測試，以決定是否再執行迴圈主體



## 模組化設計



## 函數的說明



© 2002 Microchip Technology Taiwan. All Rights Reserved.

135

MPLAB-C18 Workshop

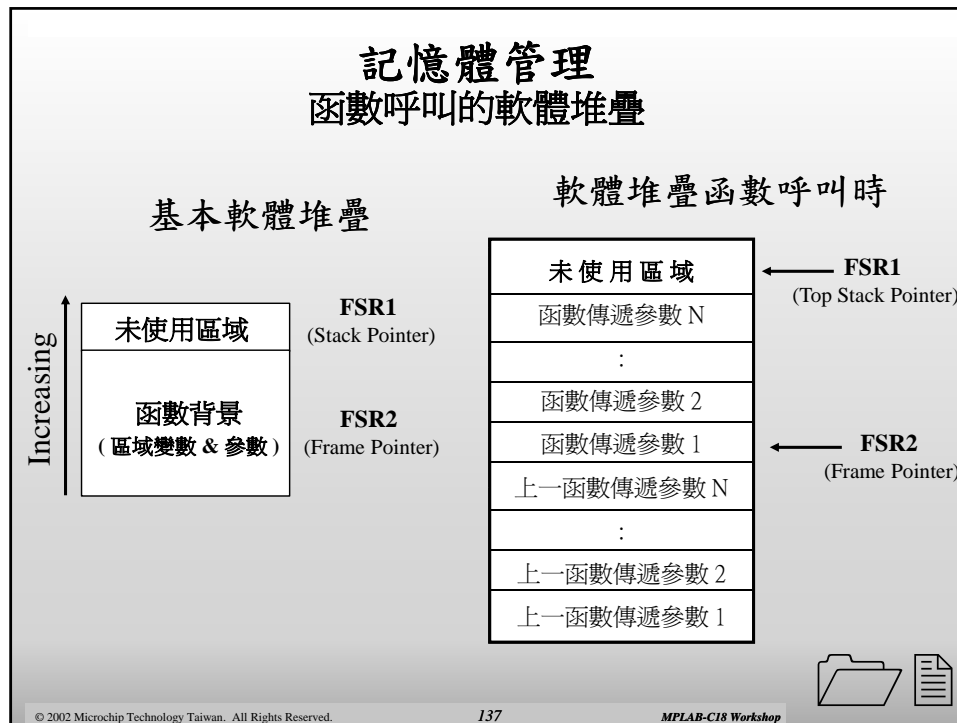
## 記憶體管理 指標(FSR)

- 資料指標(Pointers to RAM)
  - ✓ FSR0 (只有較低的12 bits 有效)
  - ✓ 使用在一般通用型指標(例:陣列、字串...)
- 堆疊指標(Stack pointer)
  - ✓ FSR1
  - ✓ 指標所指的位置是下一個有效的堆疊位置
  - ✓ 最好不要動它
- 框架指標(Frame pointer)
  - ✓ FSR2
  - ✓ 指標所指的位置是上一層參數傳遞最後位置
  - ✓ 使用前其值必須存到軟體堆疊

© 2002 Microchip Technology Taiwan. All Rights Reserved.

136

MPLAB-C18 Workshop

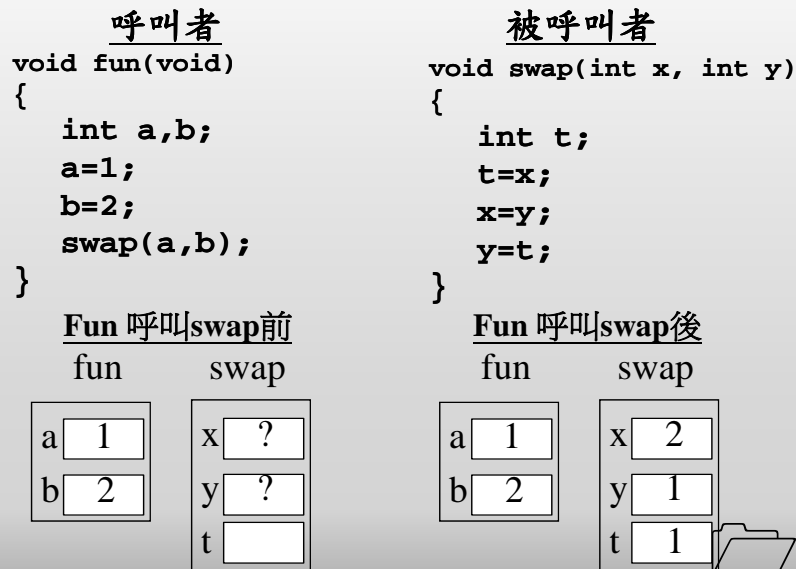


## 何謂 Call-by-Value

- ◆ 呼叫者(Caller)可將引數傳給被呼叫的函數
  - `void add (char x, char y)`
  - `void` 是指本函數無回傳值(Non-return)
- ◆ 呼叫者與被呼叫者之間各有獨立的變數運作空間 (RAM)，即被呼叫者程式執行完畢，其結果並不會影響到呼叫者本身的變數
- ◆ 引數的傳遞
  - 傳固定的常數值
  - 傳變數值

© 2002 Microchip Technology Taiwan. All Rights Reserved. 138 MPLAB-C18 Workshop

## Call-by-Value



© 2002 Microchip Technology Taiwan. All Rights Reserved.

139

MPLAB-C18 Workshop

## 何謂 Call-by-Reference

- ◆ 以變數的位址作為引數的傳遞
- ◆ 變數的前面加一 & 符號即表示要傳遞的是位址
  - 陣列及字串的名稱本身即為一指標
- ◆ 被叫用的函數要將引數宣告為指標型態
  - void swap ( int \*Var1 , int \*Var2 )
- ◆ 使用指標，被呼叫者內相關變數的改變會直接的影響到呼叫者的變數
- ◆ 使用指標，可輕易的傳遞整個陣列或字串至函數中操作

© 2002 Microchip Technology Taiwan. All Rights Reserved.

140

MPLAB-C18 Workshop

## Call-by-Reference (Call-by-Address)

呼叫者

```
void fun(void)
{
    int a[3]={1,2,3};
    int i=1;
    swap(&i,&a[i]);
}
```

執行前

i=1  
a 陣列內容 = 1, 2, 3

被呼叫者

```
void swap(int *x, int *y)
{
    int t;
    t=*x;
    *x=*y;
    *y=t;
}
```

執行後

i=2  
a 陣列內容 = 1, 1, 3

© 2002 Microchip Technology Taiwan. All Rights Reserved.

141

MPLAB-C18 Workshop



## Call-by-Address (範例)

```
const rom char Msg[]="Enter your password:"; //設定ROM固定字串表
unsigned char DispBuf[20];                    // 設定陣列在 RAM 記憶體

void main (void)                             // main 函數
{
    demo(Msg);                                // 呼叫 demo 函數
}

void demo (const rom char *data)             // demo 函數 及引數型別
{
    unsigned char i=0;

    while((*data)!=0x00)                      // 字串結束?
    {
        DispBuf[i++]=*data;                  // 將ROM 的字串複製到陣列裡
        data++;                              //將ROM 的字串指標指到下一個
    }
}
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

142

MPLAB-C18 Workshop



## 函式回傳值 (Returning Values from Functions)

- ◆ 函式可以利用 return 敘述把特定的值傳回給呼叫者(Caller)
- ◆ 傳回值的宣告
  - 傳回值必須在被呼叫的函式外宣告
  - 函式必須冠上傳回值的資料型態



## 函式回傳值 (範例)

```
Void main (void)
{
    unsigned char c=0;

    c = sum(5,10);
    c = sum(15,20);
    c = sum(25,30);
}

unsigned char sum(unsigned char a,unsigned char b)
{
    unsigned char x;
    unsigned char y=2;
    x = y * (a+b);
    return(x);
}
```





## 陣列 (Arrays) 指標 (Pointers)



## 陣列 (Array)

- ◆ 陣列是由一群具相同資料型態且相鄰位置的元素所組成的集合體
  - 格式: 資料型態 陣列名稱[N];
  - 其中N為該陣列的元素個數，實際從0到(N-1)
    - ✓ int Month[12];
    - ✓ char a[10];
- ◆ 陣列的大小最好可放在同一個Bank內
- ◆ 陣列中的元素表示，必須用陣列名稱 + 索引值
  - 例如：陣列名稱 a[10]

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
------	------	------	------	------	------	------	------	------	------



## 陣列初值的設定

- ◆ Static char a[5] = {2, 4, 6, 8, 10}; // 常數設定

a[0]	a[1]	a[2]	a[3]	a[4]
2	4	6	8	10

- ◆ Static char a[5] = {'x', 'y', 'z', 'u'}; // ASCII字元設定

a[0]	a[1]	a[2]	a[3]	a[4]
x	y	z	u	\0

- ◆ Static char a[7] = {"Hello"}; // ASCII字串設定

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
H	e	l	l	o	\0	\0



## 陣列 和 字串

- ◆ 字串是多個字元的組合，最後是以0x00（通稱null character \0）為結束字元

- ◆ 如果陣列的值是常數，可將該陣列設定到ROM區域

```
const rom char ch[7]={'H','e','l','l','o','\0'};
```

```
const rom char ch[]={ "Hello" };
```

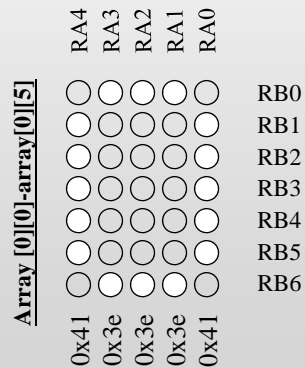
//其中 “const rom”的宣告會將陣列擺在 ROM 裡



## 二維陣列(範例)

```
const rom char Display_Table[4][5]=
{
    {0x41,0x3e,0x3e,0x3e,0x41}, // Display "0"
    {0x31,0x3d,0x00,0x3f,0x3f}, // Display "1"
    {0x39,0x1e,0x2e,0x36,0x39}, // Display "2"
    {0x5d,0x36,0x36,0x36,0x49}, // Display "3"
};

void Display(unsigned char Status)
{
    unsigned char x;
    if (Status<4)
    {
        for (x=0;x<5;x++)
        {
            PORTB=Display_Table[Status][x];
            PORTA=0x01 << x;
            Delay_1ms(3);
        }
    }
}
```



## 記憶體管理 資料陣列的擴展

- ◆ 陣列的存取一般是被限定在同一個 “Bank” 內的資料最多為 256 Bytes。
  - 巨大陣列的存取在MPLAB-C18是可以被允許的
  - 修改連結敘述檔(Linker Script)即可達到巨大陣列的存取：
    - ✓ 檔案位置 “C:\mcc18\lkr\18f4520.lkr”。
    - ✓ 陣列的擴展必須是相鄰的banks。
    - ✓ 此區域必須加以保護以避免被其它變數所使用，使用 “PROTECTED” 的屬性參數。

## 記憶體管理

資料陣列的擴展範例 - 先建立一個夠大的連續資料節區

### Large arrays - linker script

```
CODEPAGE      NAME=vectors  START=0x0      END=0x27      PROTECTED
CODEPAGE      NAME=page     START=0x28     END=0x7FFF
CODEPAGE      NAME=config   START=0x200000 END=0x2001FF  PROTECTED

ACCESSBANK    NAME=accessram START=0x0      END=0x7F
DATABANK      NAME=gpr0     START=0x80     END=0xFF
DATABANK      NAME=gpr1     START=0x100    END=0x1FF
//DATABANK    NAME=gpr2     START=0x200    END=0x2FF
//DATABANK    NAME=gpr3     START=0x300    END=0x3FF
DATABANK      NAME=gpr4     START=0x400    END=0x4FF
DATABANK      NAME=gpr5     START=0x500    END=0x5FF
ACCESSBANK    NAME=accesssfr START=0xF80    END=0xFFF

DATABANK      NAME=hugestuff START=0x200    END=0x3FF     PROTECTED
SECTION NAME=HugeObj  RAM=hugestuff
```

在程式中用 `#pragma udata` 指定

© 2002 Microchip Technology Taiwan. All Rights Reserved.

151

MPLAB-C18 Workshop

## 記憶體管理

資料陣列的擴展範例 - 在 C Program 需宣告使用特定節區

### Large arrays - C source

```
#pragma udata HugeObj           //select HugeObj section
static unsigned char Array1[0x200]; //define array
#pragma udata                   //return to default udata section

unsigned char *Ptr1;             //define array pointer (16-bits)

void main(void)
{
    unsigned int Count;          //define counter variable

    Ptr1 = Array1;               //initialize pointer

    for (Count=0x00; Count<0x200; Count++)
    {
        *Ptr1=0xFF;              //set array element to 0xFF
        Ptr1++;
    }
}
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

152

MPLAB-C18 Workshop

## 指標 (Pointers)

- ◆ 如果有一個變數內存放的是“變數A”的位址
  - 則這個變數就稱之為“變數A”的指標
  - 這個變數稱為指標變數
  - 指標，類似組合語言的間接定址方式(INDF, FSR)
- ◆ 指標變數的宣告
  - “\*” 符號放在變數前面，就是宣告為指標型態
  - `char *x;` // x 是字元指標(存的是位址)
  - `int *y,*z;`
- ◆ 取得變數的指標(位址)
  - “&” 符號放在變數前面，就是取得該變數的位址
  - `ptr = &i;` //將變數“i”的位址傳給ptr



## 指標的操作

- ◆ 假設指標為：`char *Ptr1;`
  - `Ptr1=0x20;` 將指標位址設為0x20；(危險舉動最好不要)
  - `*Ptr1=0x20;` 將常數0x20載入到指標所指的位址(目前位址為0x20)
  - `Ptr1 += 5;` 將指標位址加5, 成為0x25
  - `*Ptr1 += 5;` 將常數5與指標所指的位址內容相加
- ◆ 指標運算
  - 指標本身可以被遞增或遞減(一個單位，看指向的資料大小)
  - 指標可以用計算的方式加、減指定個單位(查表)

```
int *IntPtr;
char *ChPtr;
IntPtr++;           //pointer incremented by 2 bytes
ChPtr++;            //pointer incremented by 1 byte
```



## 指標初始設定

- ◆ 記著！指標的內容就是位址

```
int Var;                // 宣告變數 Var
int *VPtr;              // 宣告 VPtr 為指標變數
char Array1[] = "Hello"; // 宣告陣列 Array1[]
char *APtr;             // 宣告 APtr 為指標變數

VPtr = &Var;            // 將 Var 變數的位址傳給 VPtr
APtr = Array1;          // 將陣列 Array1 的起始位址傳給 APtr
APtr = &Array1[0];      // 作用與上一行相同
                        // 注意：APtr為16-bit的指標變數
*APtr = 'C' ;           // 將 Hello 改為 Cello
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

155

MPLAB-C18 Workshop



## 指標 VS 陣列

- ◆ 範例：

```
void Add(char *Ptr1, char Length)
{
    while(Length != 0x00)
    {
        *Ptr1 += 0x05;
        Length--;
        Ptr1++;
    }
}

void main(void)
{
    char Array[] = {0,1,2,3,4}; //define and initialize array
    char *APtr = Array;         //define and init pointer to array

    Add(Array,5);               //call function using array variable
    Add(APtr,5);                //call function using pointer
    Add(&Array[0],5);           //call function using array element
}
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

156

MPLAB-C18 Workshop



## 綜合問題？

請說明程式中

```
Add(Array,5);
Add(APtr,5);
Add(&Array[0],5);
```

三種函數的執行結果有何不同？

## 結構型態 (Structures)

## 共用型態 (union)



## 結構型態 (Structures)

- ◆ 結構是可以把不同型態的資料收集在一起當作一個整體，可以用“結構變數名稱. 成員”的名字來指定結構中的某一成員
- ◆ 當結構的名字在程式中單獨被提及時，所代表的是整個結構，而不是結構的位址
- ◆ 結構變數的位址可以用 & 運算子取得
- ◆ 使用結構的場合
  - 成員之中具有各種不同的資料型態（型態相同可用陣列）
  - 多樣化變數宣告

```
struct struct-name ← 結構名稱
{
    type member1;
    type member2;
    . . .
} variable-name; ← 結構變數
```

變數成員



## 使用結構變數

- ◆ 欲使用結構內的成員，可使用“.”來組成：

**variable-name.memberx** (結構變數.成員)

```
struct Comm_protocol
{
    char ID[6];
    char Data[10];
    char Message[20];
    unsigned int CRC;
    unsigned char Repeat;
} Rec_Fram;

:
:
unsigned char j;
for(j=0;j<20;j++)
{
    writeUSART(Rec_Fram.Message[j]);
}
```

Comm\_protocol 在 RAM 的排列

Rec_Fram	
成員名稱	資料長度
ID	6 Bytes
Data	10 Bytes
Message	20 Bytes
CRC	2 Bytes
Repeat	1 Bytes



## 使用位元結構 (bit)

- ◆ 位元結構是集合獨立位元的一種特殊結構
- ◆ 該結構的組成，最大值為一個位元組(byte)
- ◆ 成員為一個個單獨的位元(bit)或由數個位元(group of bits)組合而成的結構
  - 位元成員的存取、標記和一般的結構變數相同

```
struct struct-name
{
    type member1 : 3;
    type member2 : 1;
    . . .
} variable-name;
```

← 結構名稱

← 變數成員，3和1是指佔bit數

← 結構變數





## 定義位元結構（範例）

```
extern volatile near unsigned char PORTB; // 宣告PORTB是一個Byte
extern volatile near union {             // 宣告PORTBbits為一位元結構的共用
    struct {                             // 形態(union),位址在Access RAM
        unsigned RB0:1;                  // 定義PORTB的標準功能
        unsigned RB1:1;
        unsigned RB2:1;
        unsigned RB3:1;
        unsigned RB4:1;
        unsigned RB5:1;
        unsigned RB6:1;
        unsigned RB7:1;
    };
    struct {                             // 定義PORTB的另外功能
        unsigned INT0:1;
        unsigned INT1:1;
        unsigned INT2:1;
        unsigned CCP2:1;
    };
} PORTBbits ;
```

摘錄自 “P18F4520.H”檔

© 2002 Microchip Technology Taiwan. All Rights Reserved.

161

MPLAB-C18 Workshop

## 使用位元結構（範例）

```
PORTB=0x34;                               /* 記著！ PORTB & PORTBbits
                                           的位址定義是在 p18C452.asm中*/
:
PORTBbits.RB7=1;                          // RB7 輸出Hi
PORTBbits.RB6=!PORTBbits.RB6;            // RB6 輸出轉態
:
:
if (PORTBbits.INT0)                      // 測試INT0腳電壓？
{
    PORTAbits.RA0=1;
    Nop();
    PORTA >>= 1;
}
else PORTA=0;
```

© 2002 Microchip Technology Taiwan. All Rights Reserved.

162

MPLAB-C18 Workshop

## 共用型態 (union)

- ◆ 共用型態(union)，可使幾種不同的資料型態的變數共用一塊記憶空間
  - 共用型態(union)使用方式類似結構型態(Structure)
  - 共用型態(union)內的變數稱為共用元素
  - 共用型態常使用於資料轉換
- ◆ 編譯器會根據共用元素中佔記憶空間的最大者來分配記憶空間
  - 可同時宣告各種不同型態的變數

```
union union-name
{
    type member1;
    type member2;
    . . .
} variable-name;
```

共用型態名稱

共用元素成員

共用型態變數

© 2002 Microchip Technology Taiwan. All Rights Reserved.

163

MPLAB-C18 Workshop

## 共用型態的資料架構

```
union u_tag
{
    int l;
    char c[4];
} temp;
```



l			
c[0]	c[1]	c[2]	c[3]
RAM0	RAM1	RAM2	RAM3

u-tag 佔四個 Bytes

例：

```
union
{
    int l;
    char c[4];
} EE_Read_Data;

for (i=0; i<5; i++)
{
    EE_Addr=i;
    EE_Read_Data.l = EERandomRead(0xA0, EE_Addr);
    Secu_Code[i]= EE_Read_Data.c[0];
}
```

EERandRead()讀進來的資料為“int”型態，利用“union”拆成兩個“char”後，只擷取其中的低位元組“Low-byte”

© 2002 Microchip Technology Taiwan. All Rights Reserved.

164

MPLAB-C18 Workshop

## 合併使用 結構型態 & 共用型態

- ◆ 一個結構型態或共用型態的宣告內，也可含有其它的共用型態或結構型態

```
union FPvar
{
    float FPNum;                //floating point access
    struct
    {
        unsigned char Arg0;    //argument byte 0 access
        unsigned char Arg1;    //argument byte 1 access
        unsigned char Arg2;    //argument byte 2 access
        unsigned char Exp;     //exponent byte access
    } ByByte;
} Foo;

Foo.FPNum = 3.14159;
Exponent = Foo.ByByte.Exp - 0x7F;
```



## 列舉型態 (enum)

- ◆ 依據需求來宣告‘資料的型態’與其‘數值’
- ◆ 格式：

```
enum 列舉名稱 { 識別字 1,  
                識別字 2,  
                識別字 3,  
                .....  
                識別字 n,  
} 列舉變數;
```

```
enum def_m {Jan=1, Feb, Mar,  
            Apr, May, Jun,  
            Jul, Aug, Sep,  
            Oct, Nov, Dec  
} month ;
```



## 使用列舉型態

```
enum week {sun,mon,tue,wed,thu,fri,sat};

enum week day;

day=mon;
printf("%d",day);

for (day=sun;day<=sat;day++)
{
    printf ("%d",day)
}
```



## 練習四的準備

### 如何使用LCD函式

- ◆ 使用LCD函式庫時，須同時使用定義檔  
“p18F\_LCD.h”和 “delays.h”
- ◆ p18F\_LDC.h 定義LCD函數的雛型宣告
- ◆ 原始程式 “WAP\_LCD.c” 必須加入Project  
的項目中



## 使用 LCD 函數

- ◆ void OpenLCD : 開啟LCD
  - ✓ OpenLCD ( ); (範例)
- ◆ putsLCD & putsLCD : 寫一字串到LCD
  - ✓ void putsLCD (char \*ptr) ---- from RAM
  - ✓ void putsLCD (const rom far char \*ptr) ---- from ROM
- ◆ WriteDataLCD : 寫一字元到LCD
  - ✓ void WriteDataLCD (unsigned char data)
- ◆ WriteCmdLCD : 寫一控制命令到LCD
  - ✓ void WriteCmdLCD (DISP\_ON)
- ◆ LCD\_Set\_Cursor (unsigned char Y, unsigned char X) : 設定顯示的位置
  - ✓ LCD\_Set\_Cursor( 1, 0 ); 將Cursor設定在第二行第一個位置

© 2002 Microchip Technology Taiwan. All Rights Reserved.

169

MPLAB-C18 Workshop

## 練習 4

- ◆ 開啟” c:\RTC\C18\_V2\Answer\_APP001V3\ex4. mcp”
- ◆ 設定常數字串 “ C18 Workshop “在ROM區
- ◆ 設定變數字串 “A/D Result =“在RAM區
- ◆ 利用LCD函數來撰寫，開啟LCD並設定為4-bit模式、5x7雙行模式、CURSOR OFF
- ◆ 在LCD的第一行顯示 “ C18 Workshop “
- ◆ 在LCD的第二行顯示 “A/D Result =“
- ◆ 調整VR以改變AN0的輸入電壓，並將其A/D轉換結果
  - 直接控制馬達轉速，
  - A/D最高的八位元顯示在LED
  - 10-bit 的A/D結果以ASCII碼顯示在LCD上

**A/D Result = 0F6**

© 2002 Microchip Technology Taiwan. All Rights Reserved.

170

MPLAB-C18 Workshop