MICROCHIP

Regional Training Centers

W401 C18 教育訓練

第一天

版本 : W401 Rev. 3
日期 : Oct. 25 2010

MICROCHIP
Regional Training Centers

MPLAB C18 教育訓練 (Rev.3)

- 使用軟體工具
  - MPLAB IDE v8.53 (或更新版本)
  - MPLAB C18 v3.30 Lite Mode (或更新版本)
  - MPASM , MPLINK , MPLIB
- 使用硬體工具
  - MPLAB ICD3
  - Microchip APP001 Rev. 3 實驗版 (PIC18F4520)
- 參考書籍
  - MPLAB C18 Compiler User's Guide & Library
  - MPASM User's Guide with MPLINK and MPLIB
  - MPLAB IDE User's Guide
  - APP001 Vre. 3 使用手冊

MICROCHIP
Regional Training Centers

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

MICROCHIP
Regional Training Centers

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**

MICROCHIP
Regional Training Centers

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？

MICROCHIP
Regional Training Centers

爲什麼要用 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
  - 除錯較爲困難
  - 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

MICROCHIP
Regional Training Centers

C 面對組合語言的各項優勢

- 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 最大的程式空間爲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

MICROCHIP
Regional Training Centers

如果有一設計案是這樣 …

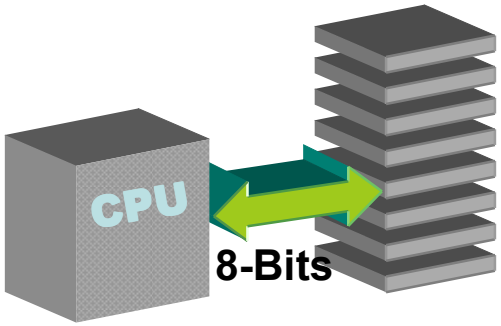
C or 組合語言比較適合呢？

- 有一工業控制溫度爲 **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)**
- 輸出爲五個控溫繼電器
  - 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

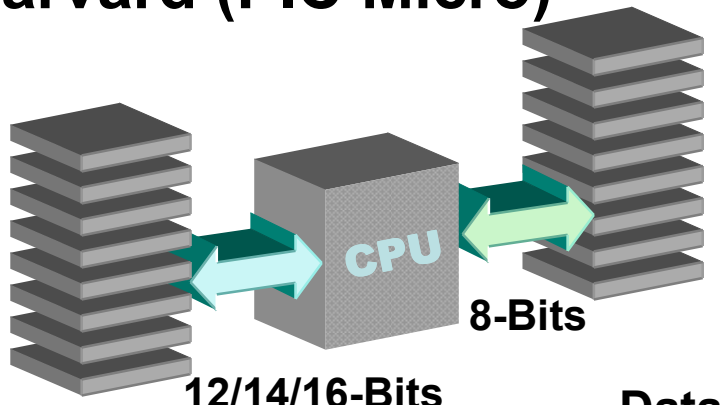
PICmicro® 的架構

Von Neumann (一般MCU)



Program
以及 Data
Memory

Harvard (PIC Micro)



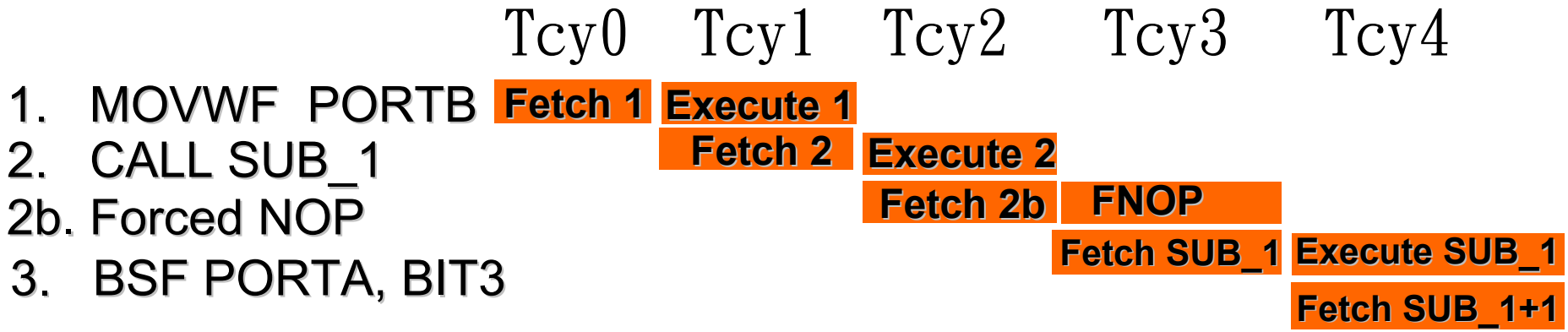
Program
Memory

Data
Memory

- 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
- 使用兩個不同的記憶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

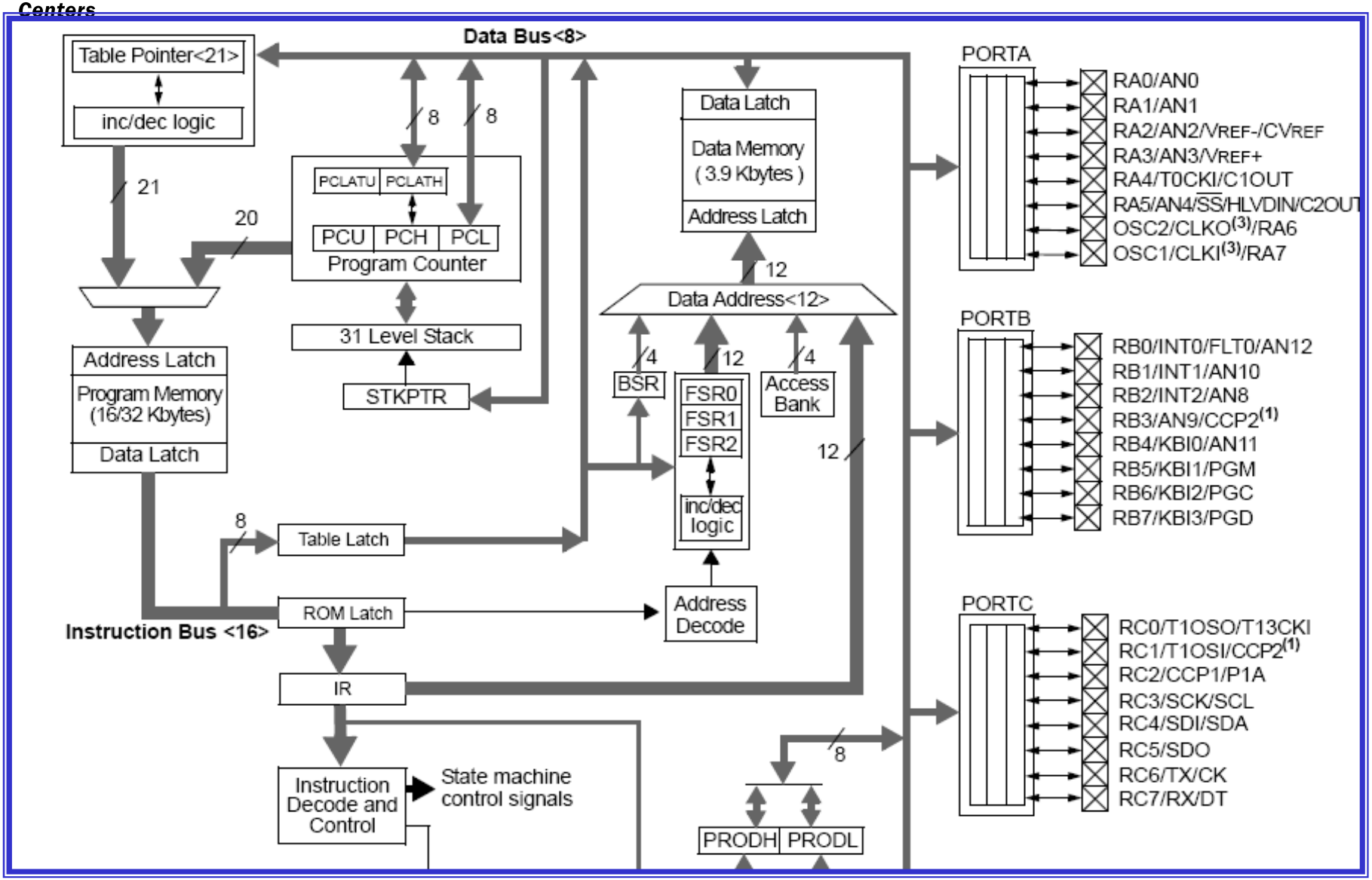
PIC18F 管狀式存取方式

- 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。
- PIC18F 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。

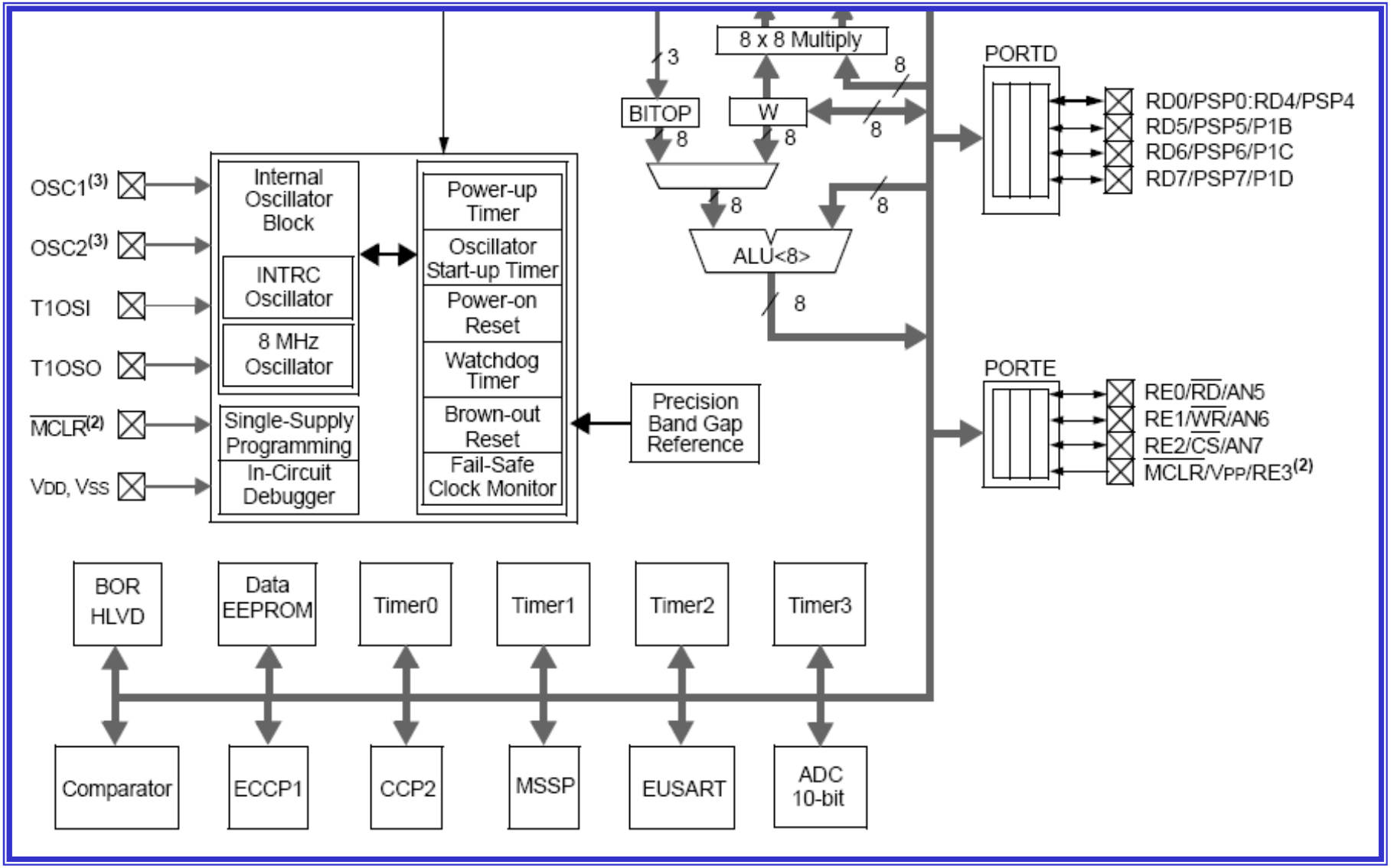


MICROCHIP
Regional Training Centers

PIC18F4520 架構圖(一)



PIC18F4520 架構圖(二)



MICROCHIP
Regional Training Centers

PIC18F4520 主要周邊 (一)

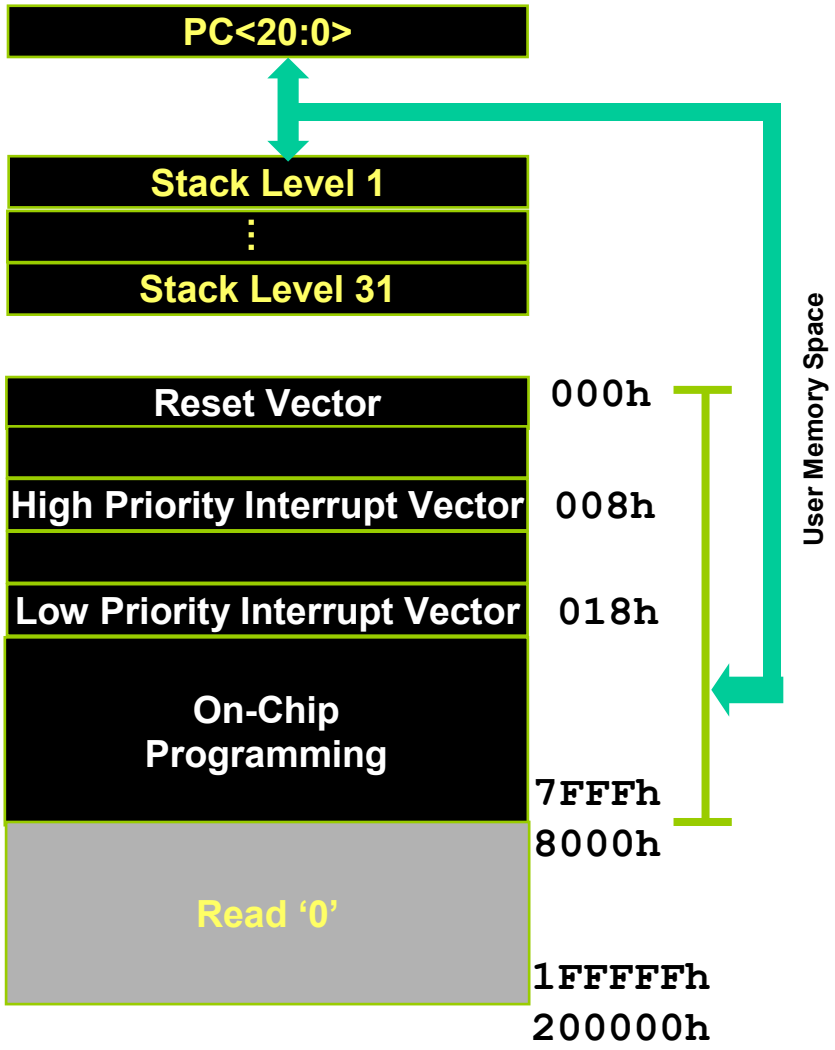
- Flash Memory : 32K Bytes (10 萬次的重寫次數)
- Data RAM : 1536 Bytes
- EEPROM : 256 Bytes (100 萬次重寫能耐)
- Timer : 4 組
- CCP Module : x 1
- Enhanced CCP : x 1
- 串列通訊 : EUSART，MI²C，SPI
- 電壓比較器 : x 2
- AD轉換器 : 10-bit / 13CH
- Reset : POR，BOR，WDT，MCLR，Stack，Reset 指令
- I/O : 最多 36

PIC18F4520 主要周邊(二)

- 振盪模式
  - HS、XT、LP、HSPLL、EC 及 External RC
  - 內建 RC 震盪 31KHz ~ 8MHz (具8種頻率選擇)
    - 可使用 x4 PLL
- Clock Failure Monitoring
- Extended WDT : 4ms ~131Sec.
- Programmable BOR (Software Enable)
- Power Done Mode
  - Run、Idle、Sleep Mode

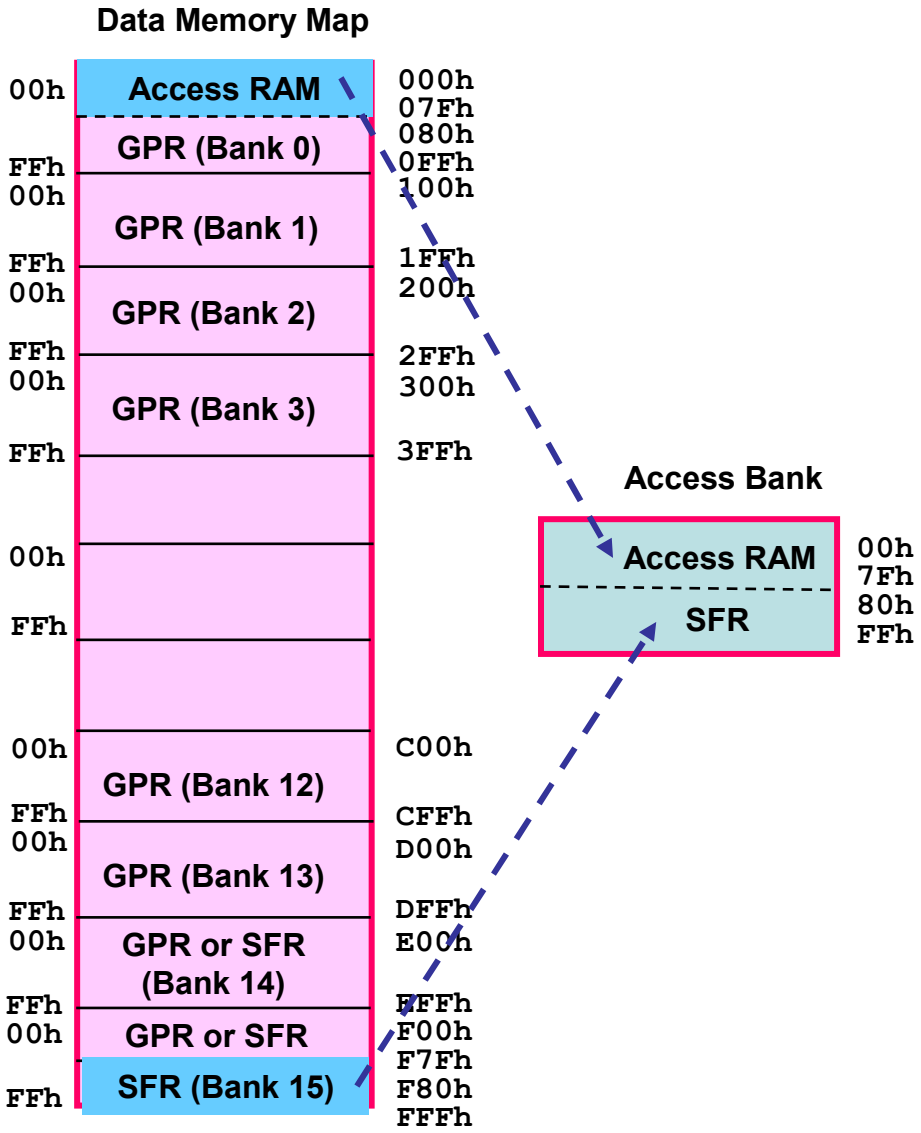
PIC18F4520 程式位置配置

- 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 的程式存取空間爲 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



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



MPLAB IDE & MPLAB C18

MICROCHIP
Regional Training Centers

Microchip整合式的研發平台
MPLAB™ IDE

系統專案管理模式
即時線上程式編輯
原始程式偵錯功能

語言工具	軟體模擬	硬體模擬器	程式燒錄器
MPASM 組譯器	MPLAB SIM 軟體模擬器	ICE-2000 ICE-4000	PICStart + ICD3
MPLINK 連結器	16-bit 軟體模擬器	PICkit2 Debug PICkit3 Debug	PICkit3 Exp. PM3
MPLAB C18 MPLAB C30 MPLAB C32		Real ICE ICD 3	

MICROCHIP
Regional Training Centers

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

MICROCHIP
Regional Training Centers

MPLAB C18 概述 (2)

- 支援軟體堆疊
  - 使用 RAM 空間做爲軟體堆疊
  - 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程式
- 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

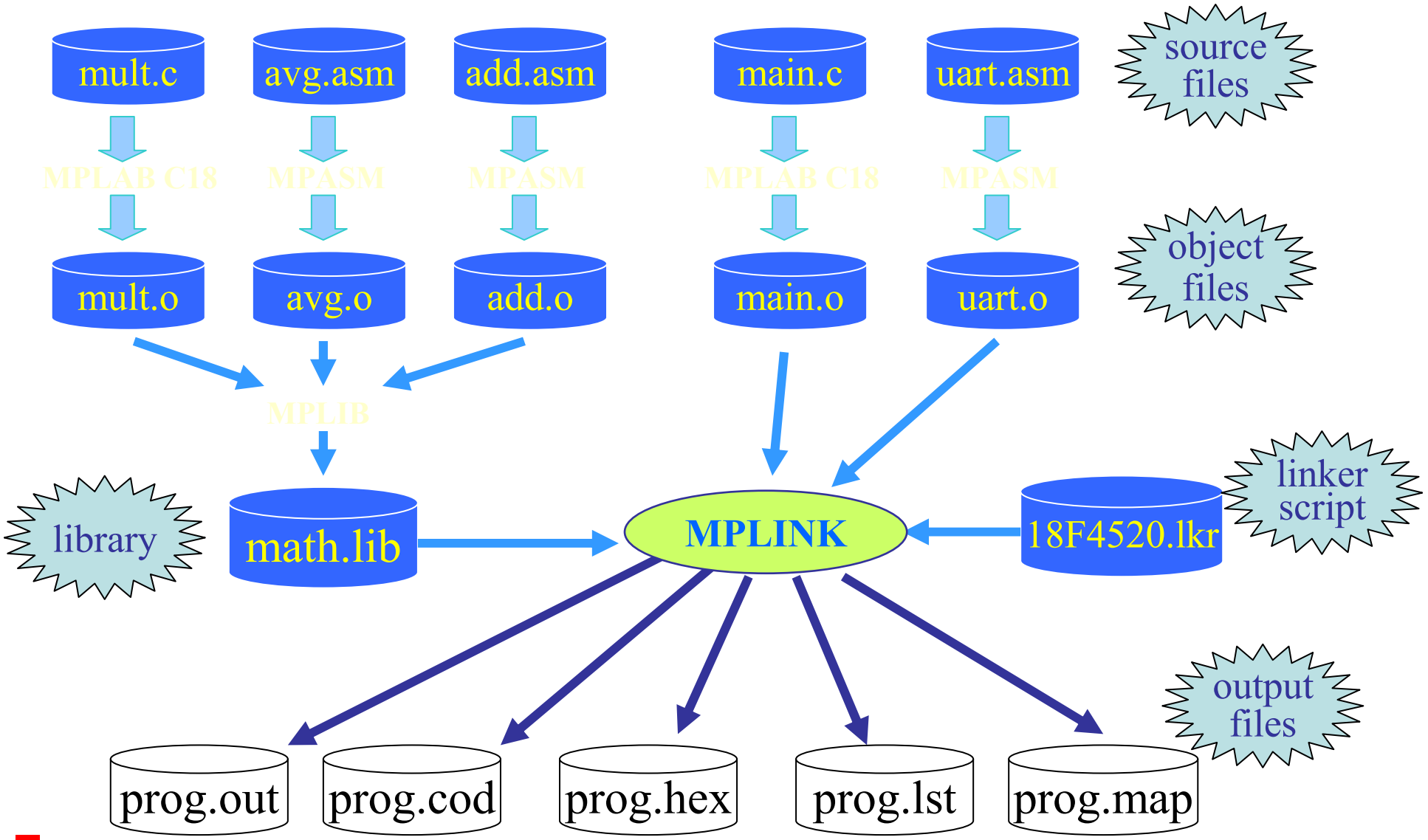
MICROCHIP
Regional Training Centers

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)
  - 產生可執行擋 (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


mult.c
avg.asm
add.asm
main.c
uart.asm
source files
MPLAB C18
MPASM
MPASM
MPLAB C18
MPASM
mult.o
avg.o
add.o
main.o
uart.o
object files
MPLIB
library
math.lib
MPLINK
18F4520.lkr
linker script
output files
prog.out
prog.cod
prog.hex
prog.lst
prog.map

練習 1

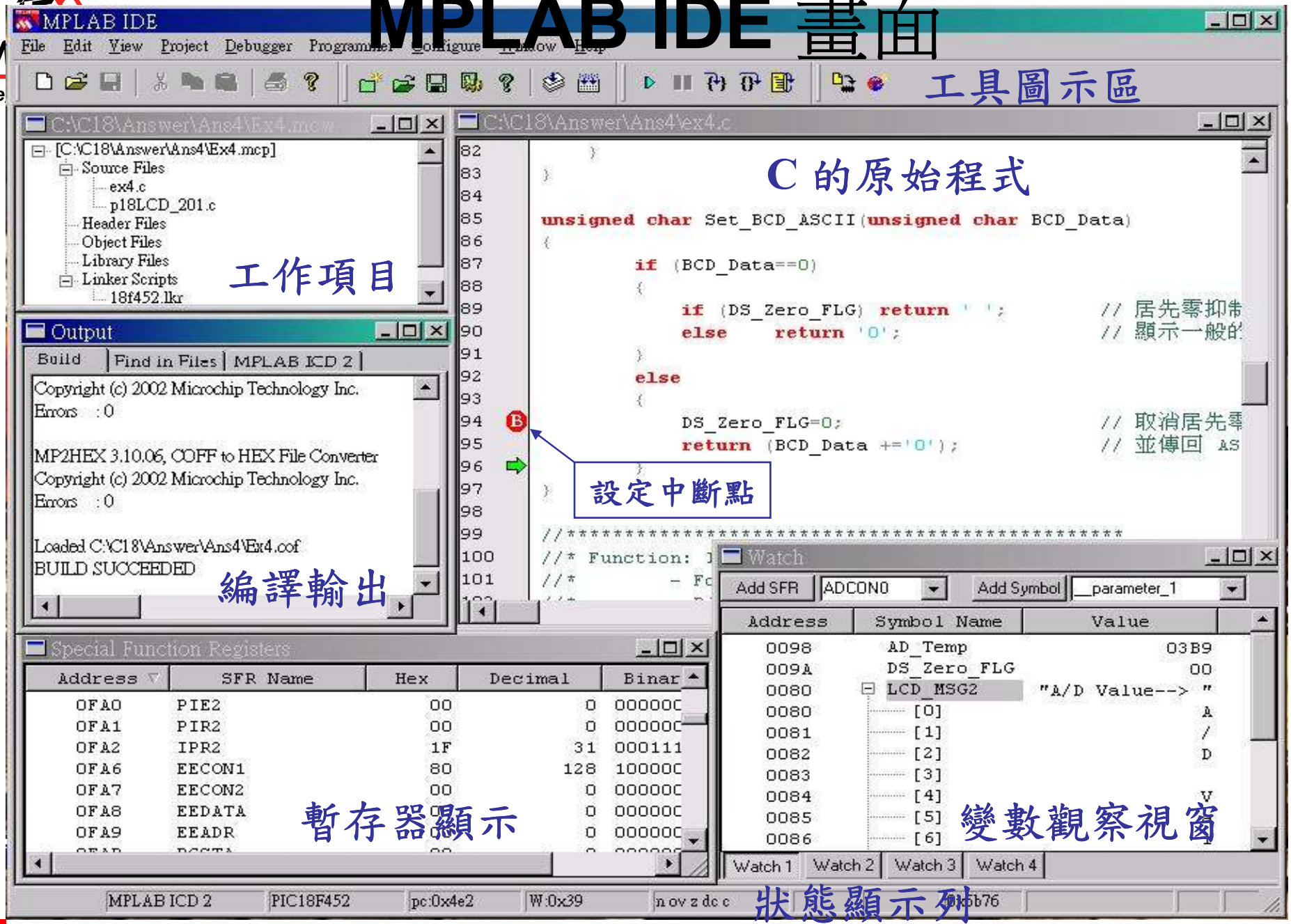
1. 認識 MPLAB IDE
2. 編輯 C 原始程式
3. 設定使用環境
4. 編譯程式與除錯

MICROCHIP
Regional Training Centers

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。

MPLAB IDE 畫面


MPLAB IDE
File Edit View Project Debugger Programmer Configure Window Help
工具圖示區
C:\C18\Answer\Ans4\Ex4.mcw
[C:\C18\Answer\Ans4\Ex4.mcp]
Source Files
ex4.c
p18LCD_201.c
Header Files
Object Files
Library Files
Linker Scripts
18f452.lkr
工作項目
C:\C18\Answer\Ans4\ex4.c
C 的原始程式
unsigned char Set_BCD_ASCII(unsigned char BCD_Data)
if (BCD_Data==0)
if (DS_Zero_FLG) return ' ';
else return '0';
DS_Zero_FLG=0;
return (BCD_Data +='0');
設定中斷點
Output
Build
Find in Files
MPLAB ICD 2
Copyright (c) 2002 Microchip Technology Inc.
Errors : 0
MP2HEX 3.10.06, COFF to HEX File Converter
Copyright (c) 2002 Microchip Technology Inc.
Errors : 0
Loaded C:\C18\Answer\Ans4\Ex4.cof
BUILD SUCCEEDED
編譯輸出
Special Function Registers
Address SFR Name Hex Decimal Binar
OFA0 PIE2 00 0 00000C
OFA1 PIR2 00 0 00000C
OFA2 IPR2 1F 31 000111
OFA6 EECON1 80 128 10000C
OFA7 EECON2 00 0 00000C
OFA8 EEDATA 0 0 00000C
OFA9 EEADR 0 0 00000C
暫存器顯示
Watch
Add SFR ADCON0 Add Symbol __parameter_1
Address Symbol Name Value
0098 AD_Temp 03B9
009A DS_Zero_FLG 00
0080 LCD_MSG2 "A/D Value--> "
0080 [0] A
0081 [1] /
0082 [2] D
0083 [3]
0084 [4] V
0085 [5]
0086 [6]
Watch 1 Watch 2 Watch 3 Watch 4
變數觀察視窗
MPLAB ICD 2 PIC18F452 pc:0x4e2 W:0x39 n ov z dc c
狀態顯示列

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 + Lift - 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 <p18cxxx.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。

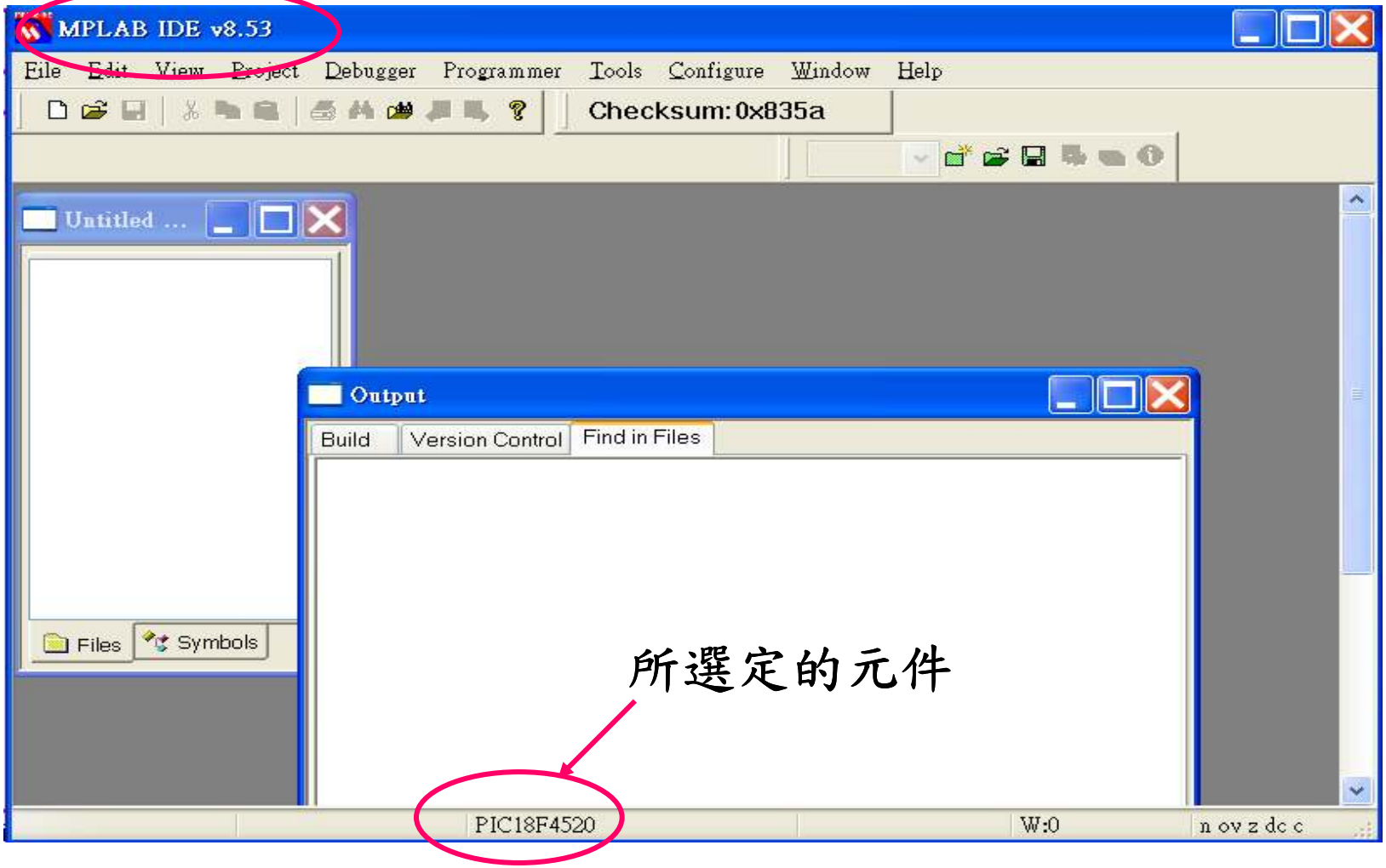
3. 輸入完成後，點選File”後再點選“Save As”並輸入如下所示之檔名並儲存。

4. ..\RTC\Ans1\ex1.c

啓動 MPLAB® IDE

步驟 1

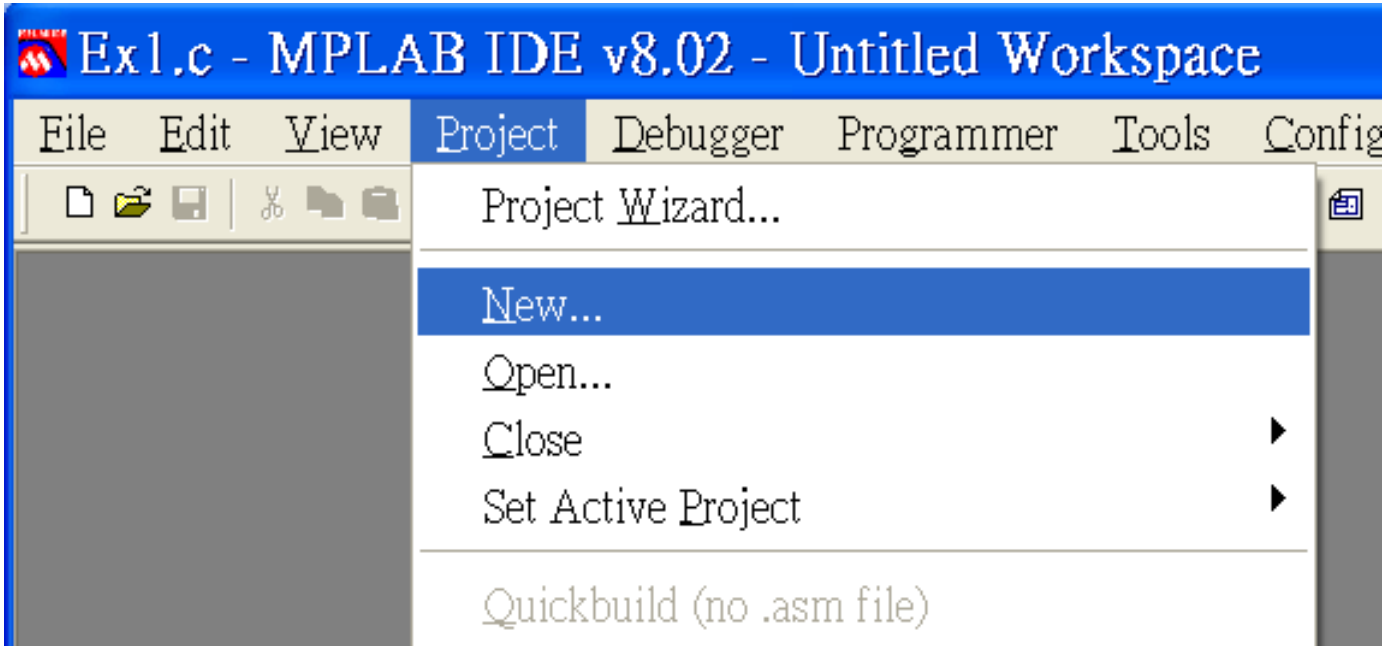
版本顯示



建立一個 New Project

步驟 2

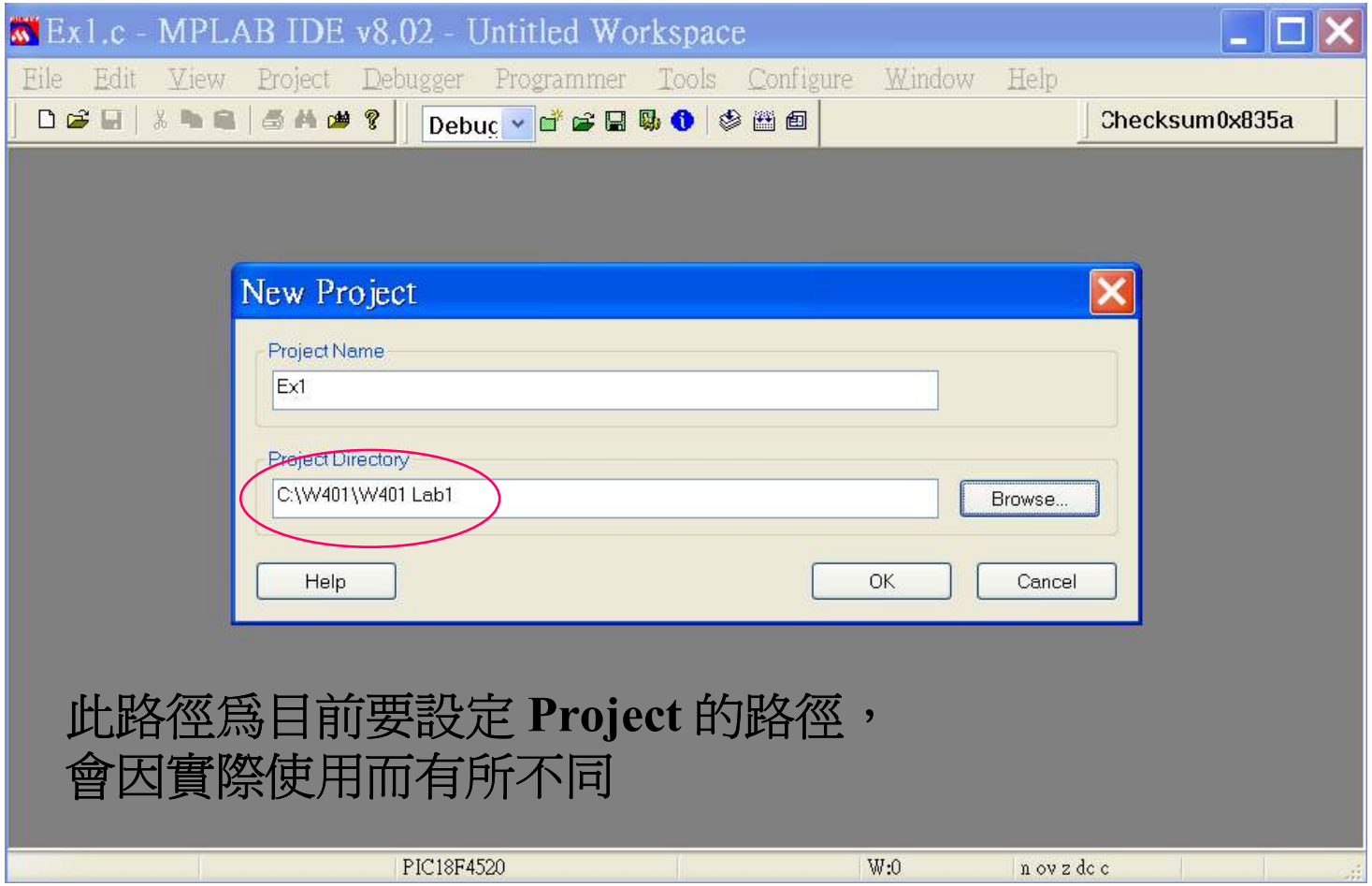
- **Project** 的建立使用 **Project Wizerd** 比較簡單快速 (建議使用)
- 這裏使用手動建置方式建立一個 **New Project** 的步驟：(一步一步慢慢來了解步驟)



輸入 New Project

步驟 3

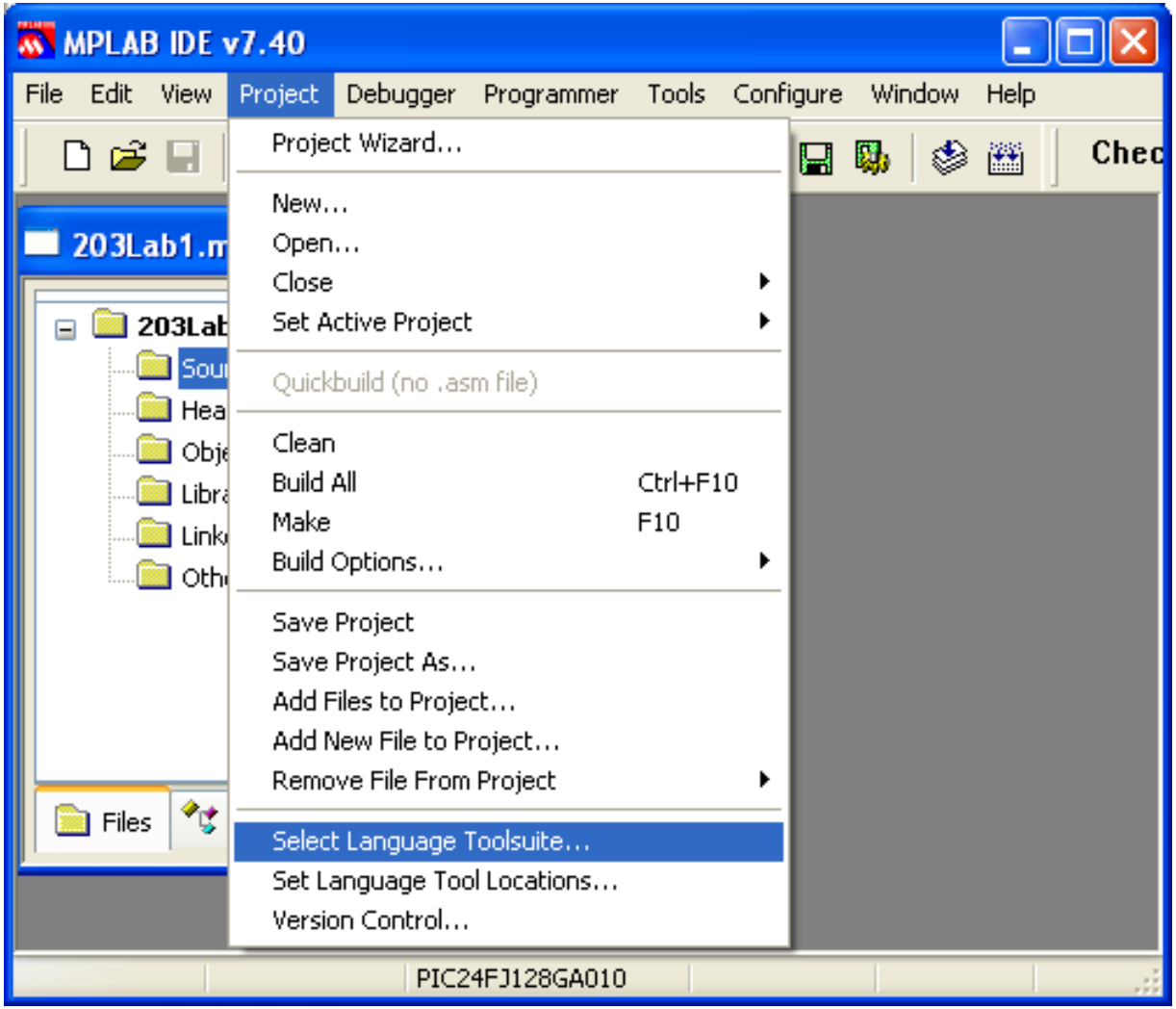
- Project Name 爲 : Ex1
- 路徑爲 : ..\W401\RTC\Ans1



設定語言工具
C18, MPASM, 或其它

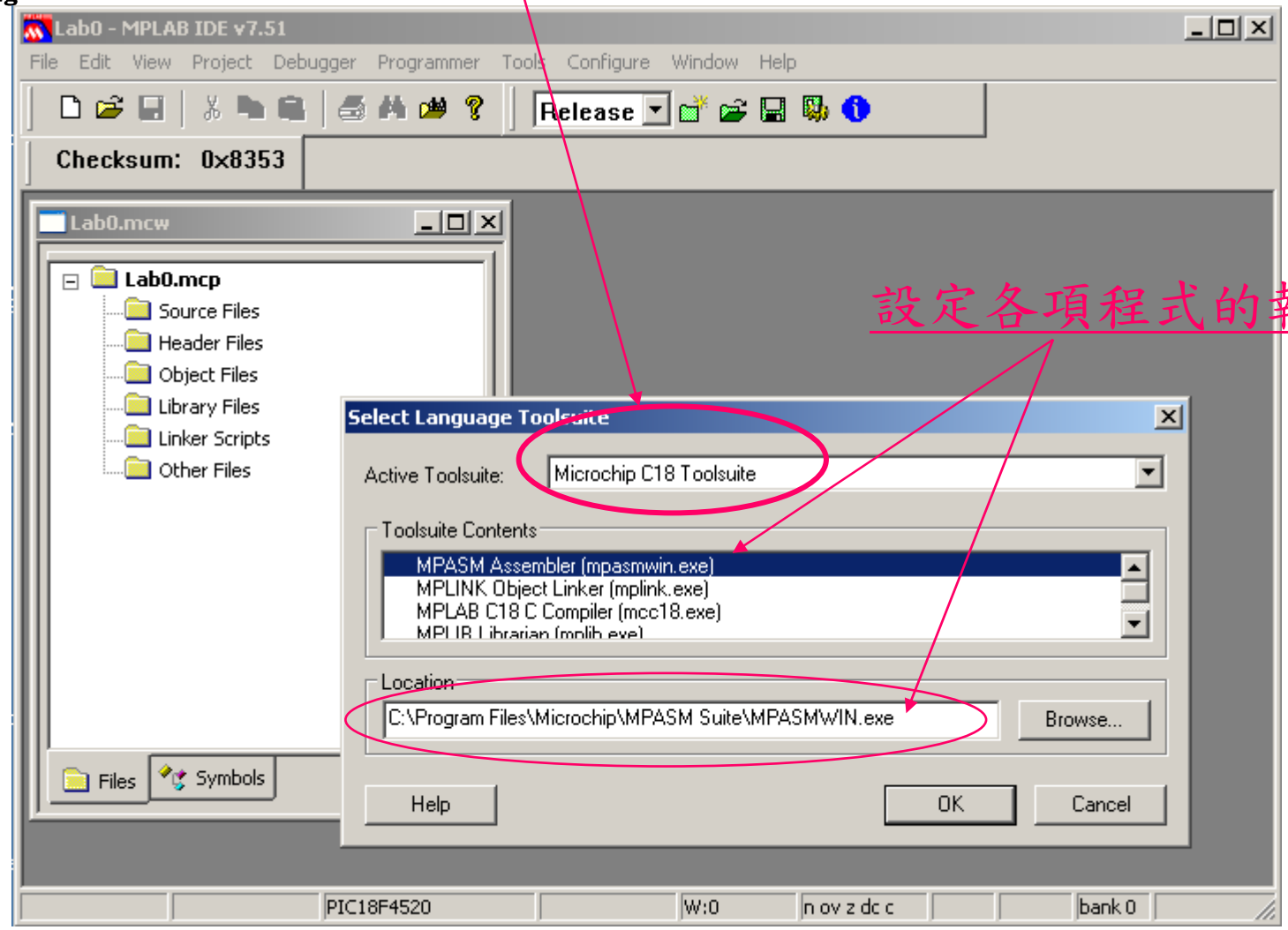
步驟 4

MPLAB IDE 是不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爲何？及其編譯程式的路徑與程式名稱。



選擇 C18 為語言工具

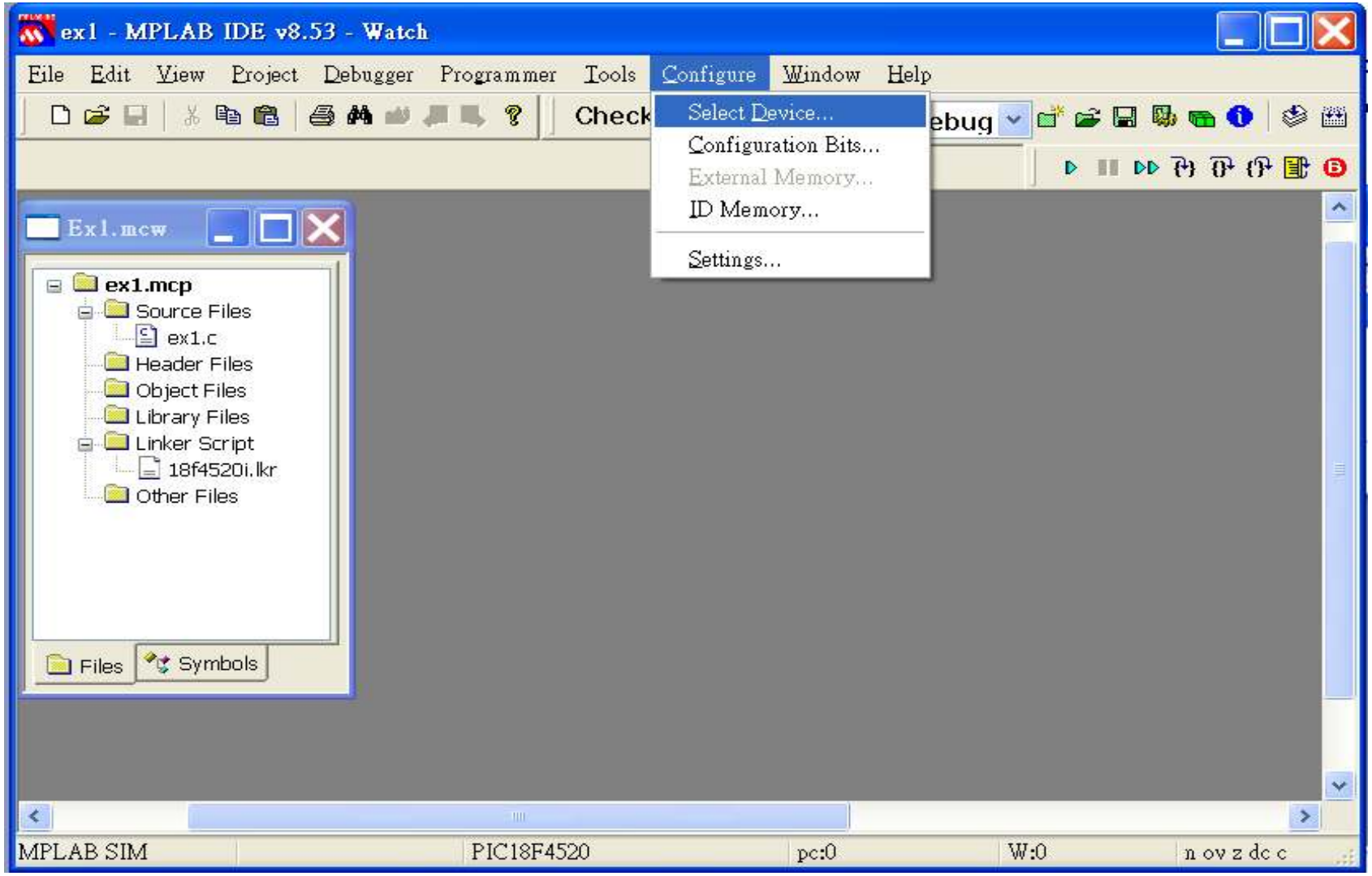
步驟 5



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

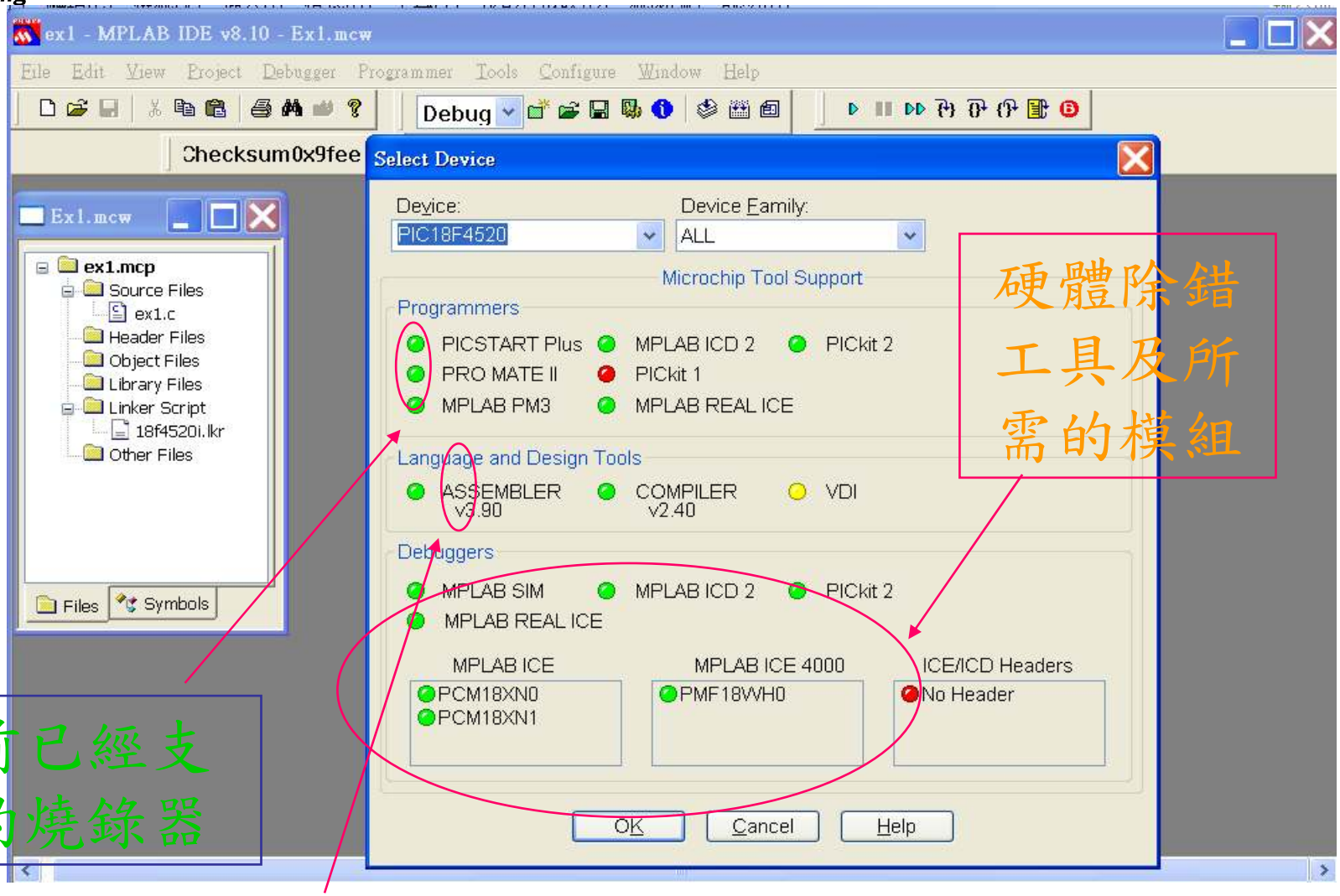
選擇使用的 PIC 型號

步驟 6



使用 PIC18F4520

步驟 7



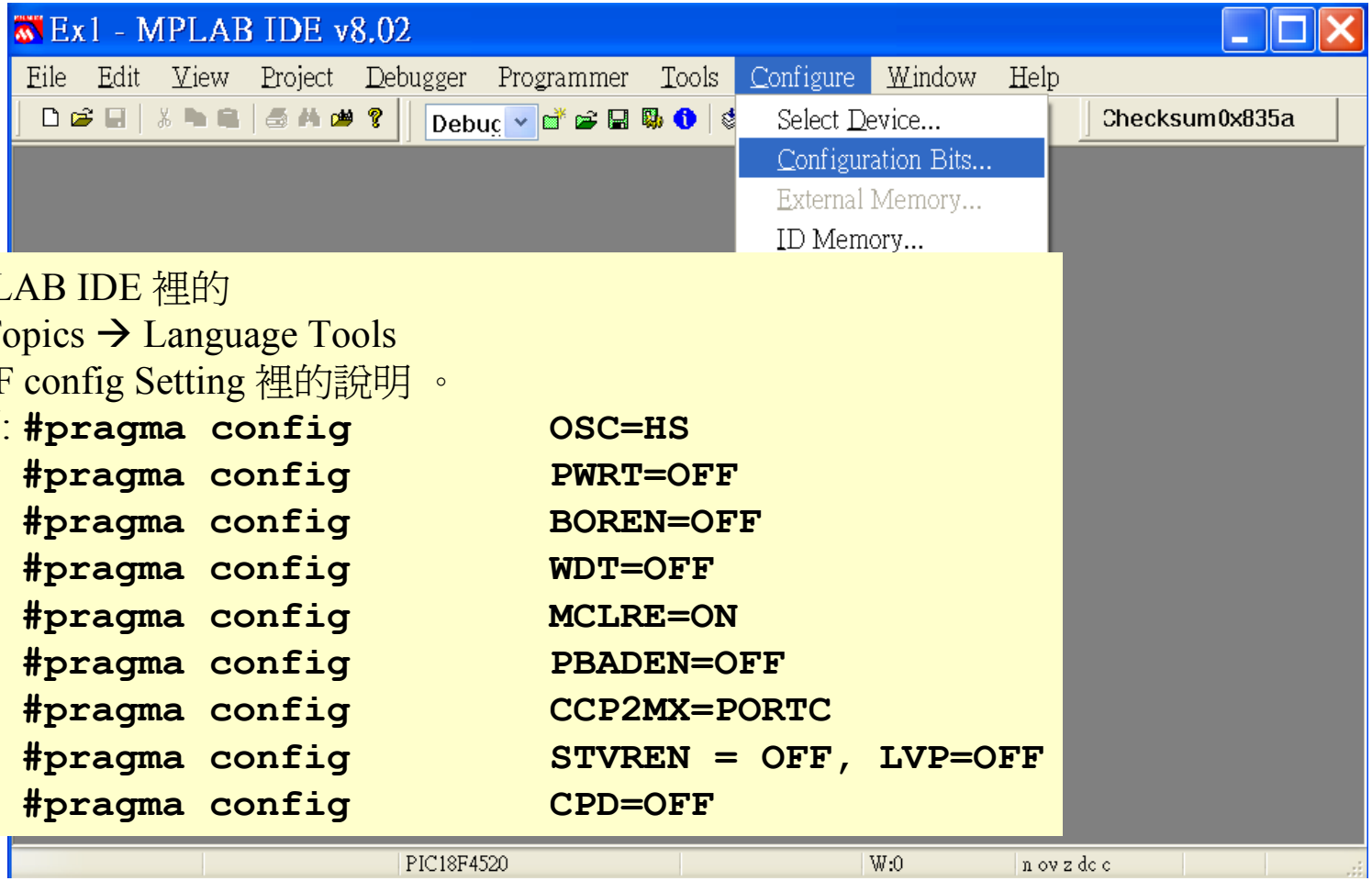
硬體除錯工具及所需的模組

目前已經支援的燒錄器

語言工具的支援

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

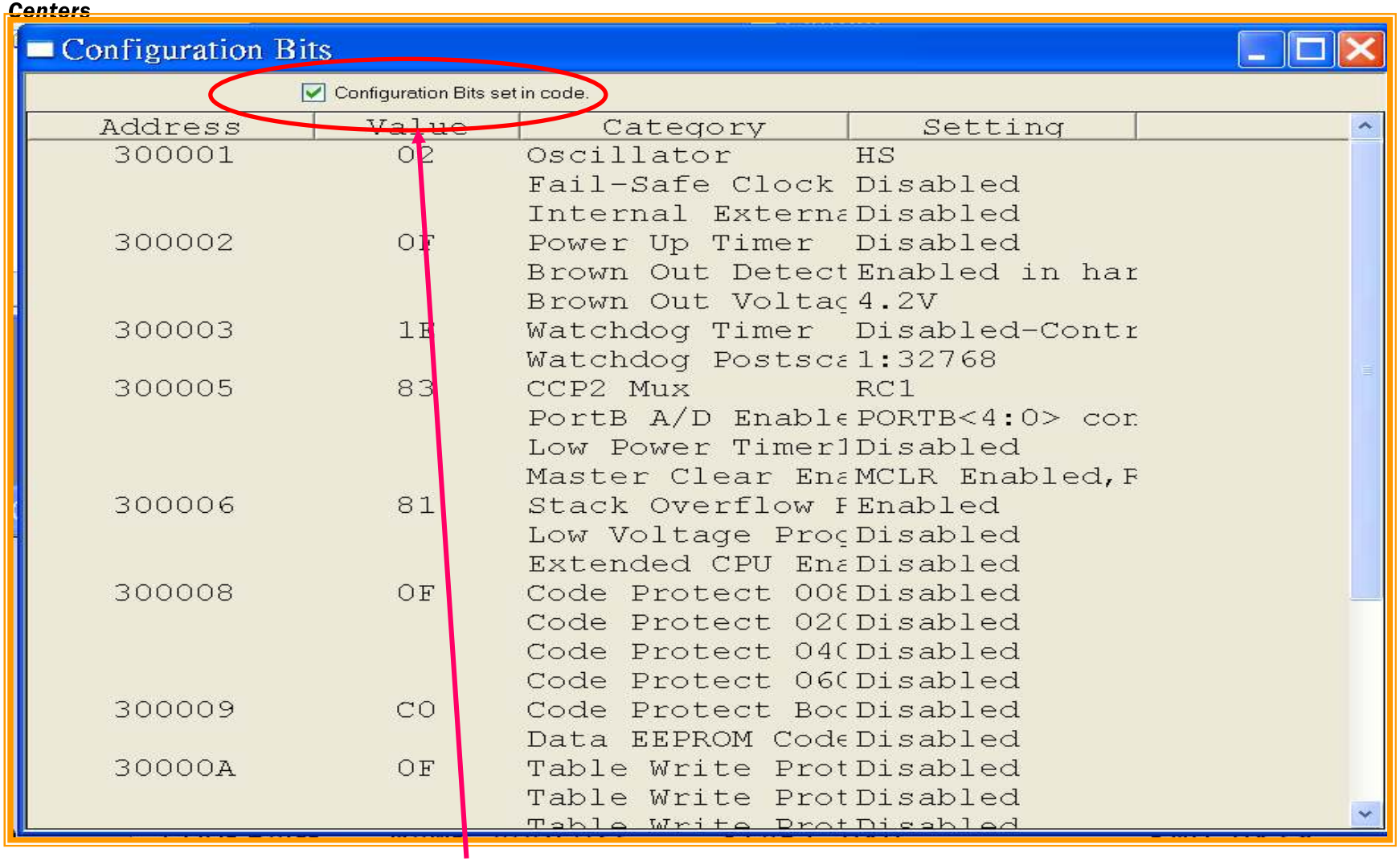


參考 MPLAB IDE 裡的
Help →Topics → Language Tools
→ PIC18F config Setting 裡的說明 。

使用範例:

```
#pragma config          OSC=HS
#pragma config          PWRT=OFF
#pragma config          BOREN=OFF
#pragma config          WDT=OFF
#pragma config          MCLRE=ON
#pragma config          PBADEN=OFF
#pragma config          CCP2MX=PORTC
#pragma config          STVREN = OFF, LVP=OFF
#pragma config          CPD=OFF
```

先檢查一下 config 的設定 步驟 9

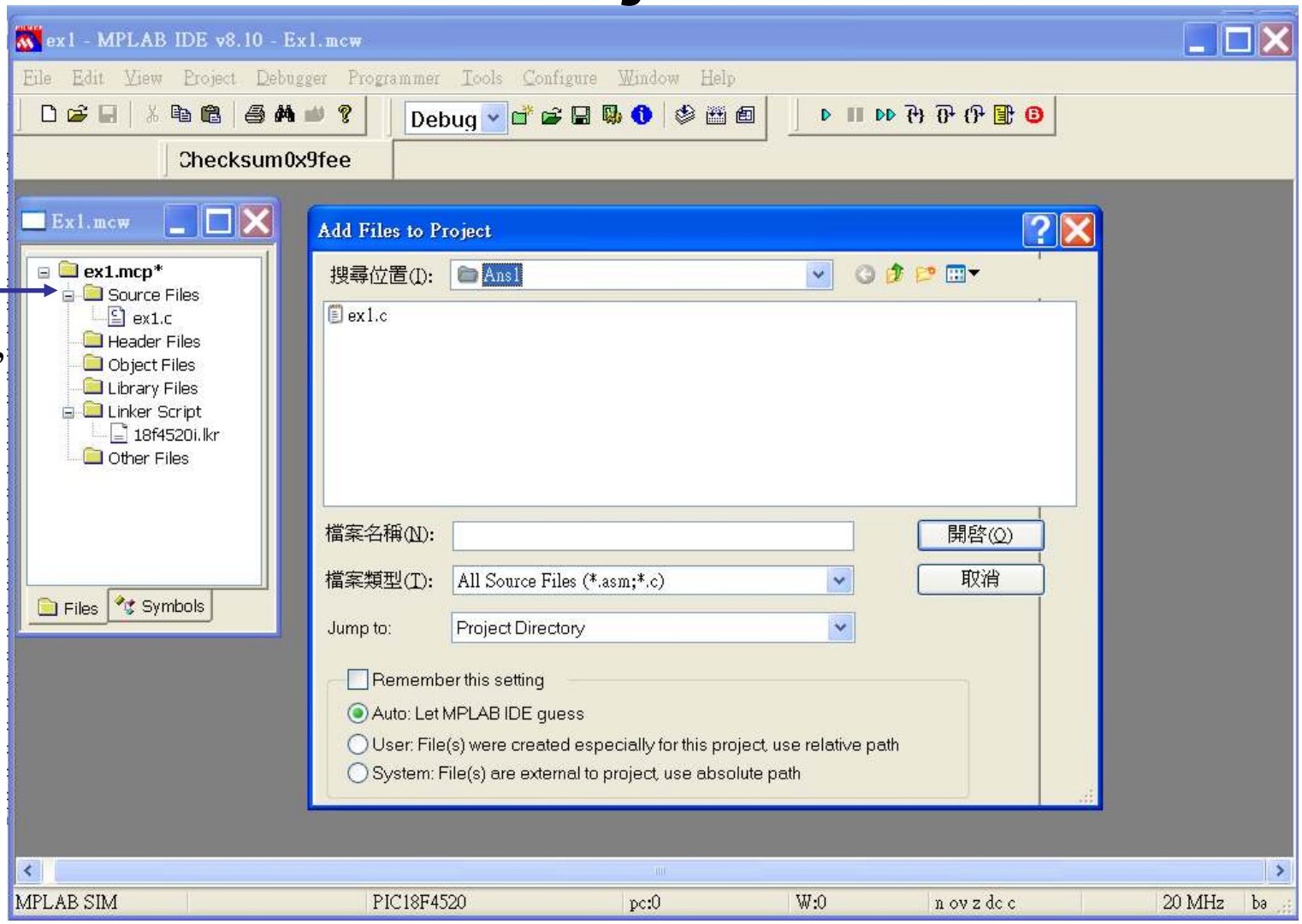


打勾拿掉後才可以用手動方式修改

將原始檔案加入目前的 Project

步驟 10

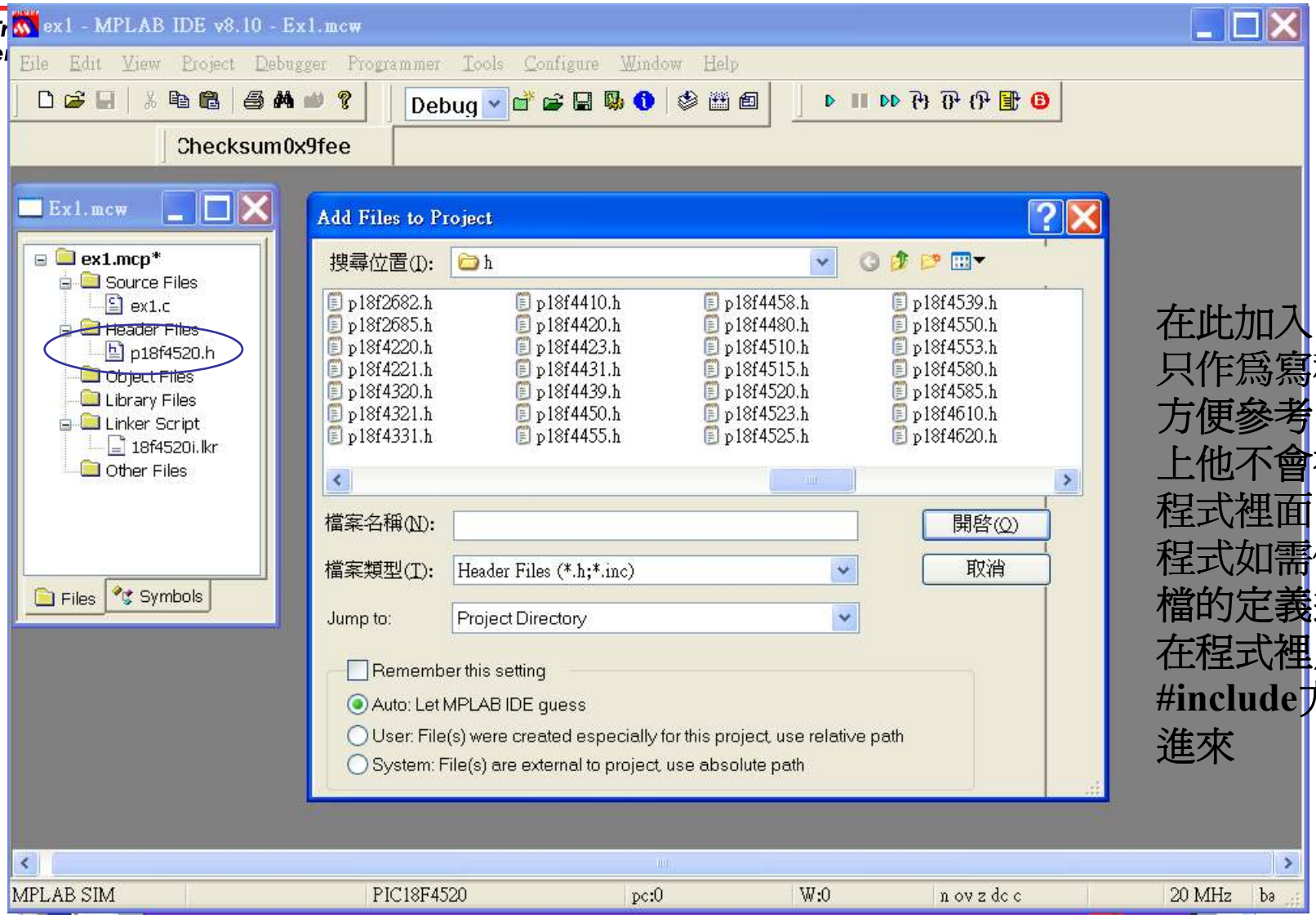
在 “Source Files” 按下老鼠右鍵 加入原始程式



Source file: **`..\Ans1\ex1.c`**

加入該零件的 H 檔案

步驟 11



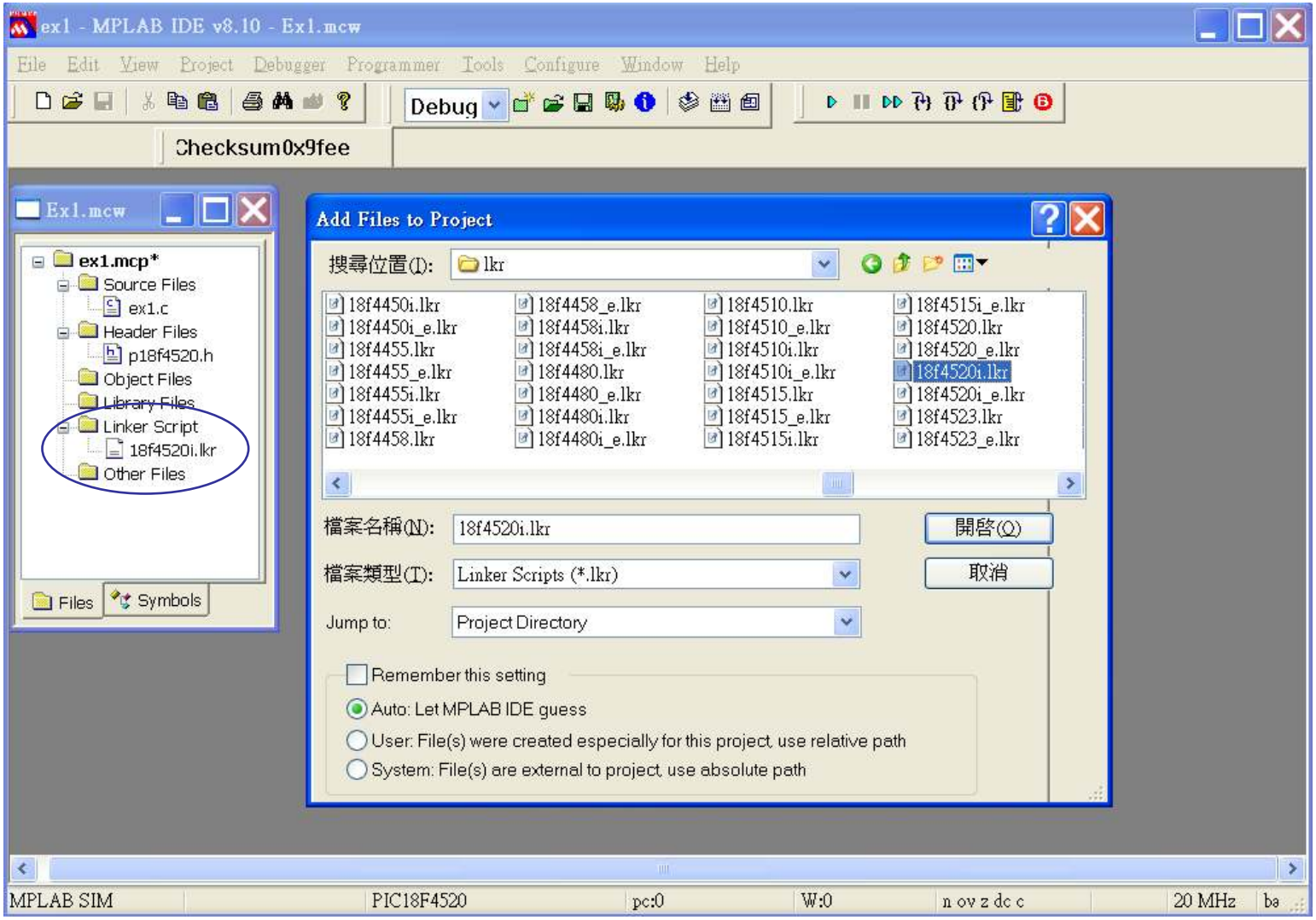
在此加入 H 檔只作爲寫程式時方便參考，實際上他不會被加到程式裡面。
程式如需使用H檔的定義還是要在程式裡用 **#include**方式加進來

Header file: **`c:\mcc18\h\p18f4520.h`**

加入連結描述檔 (LKR)

步驟 12

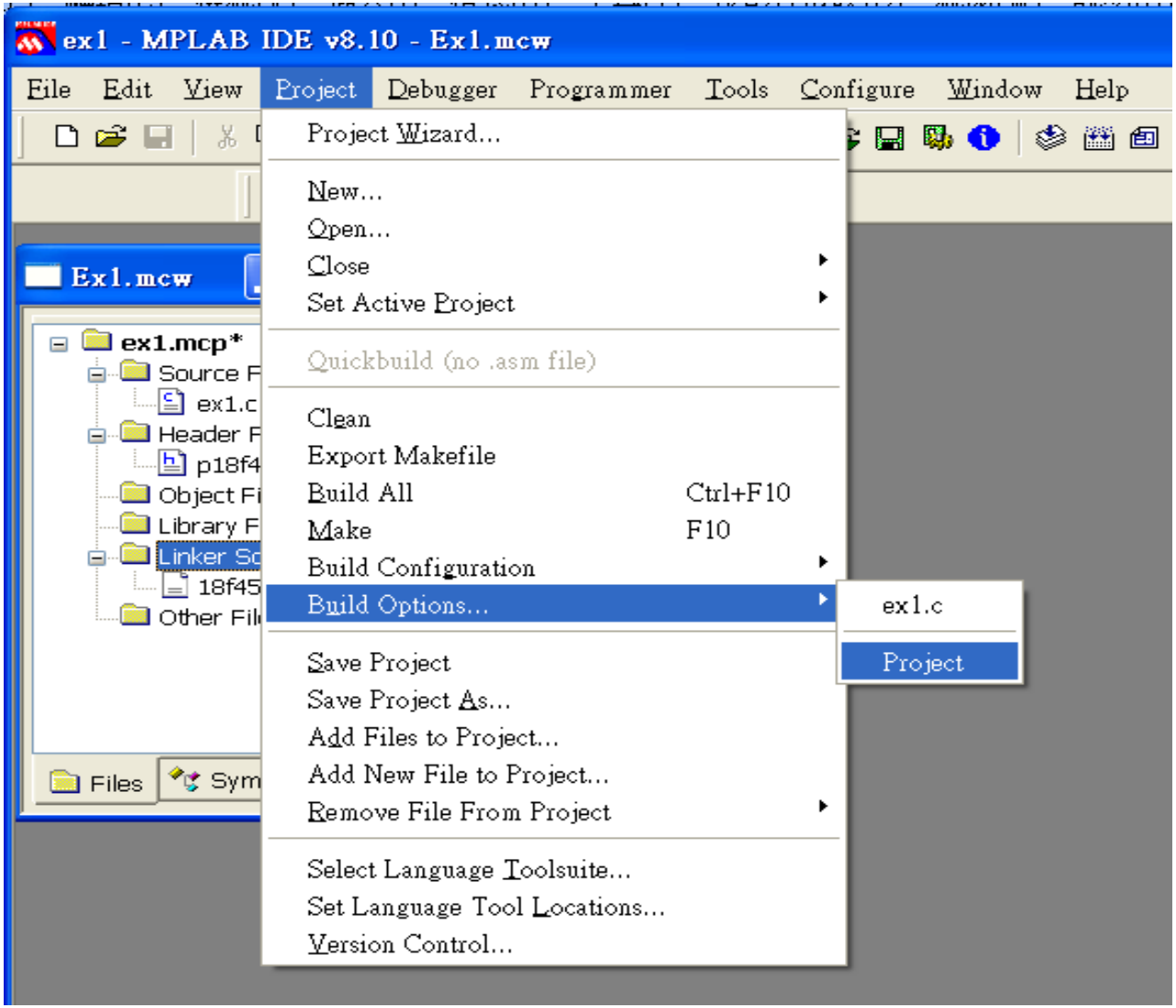
LKR 有數種不同的檔案使用時需注意不要弄錯：
18f4520.lkr
18f4520i.lkr
18f4520_e.lkr



Linker File: **`c:\mcc18\lkr\18f4520i.lkr`**

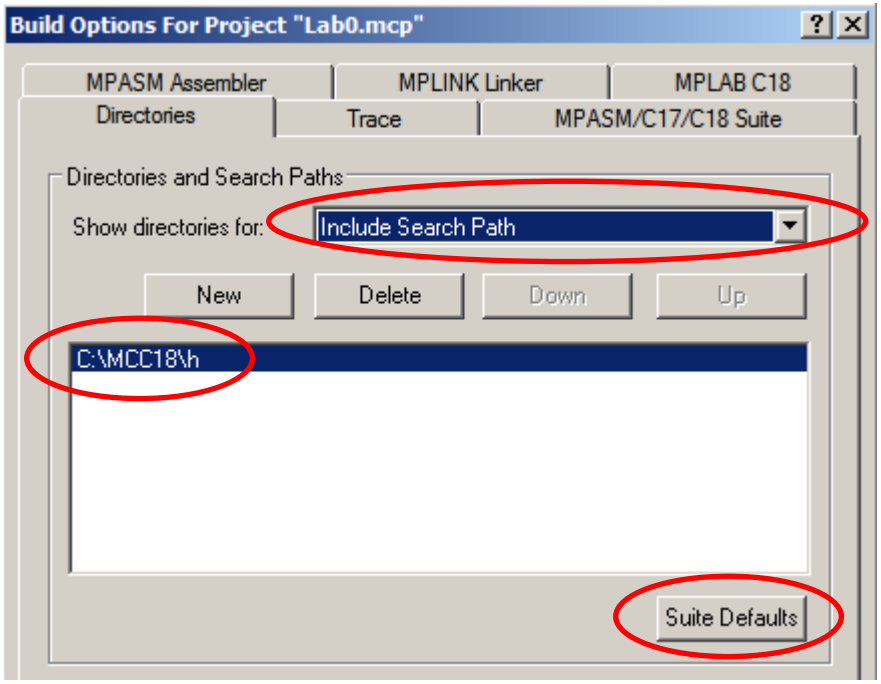
設定 Project 裡的 ‘Build Options ’

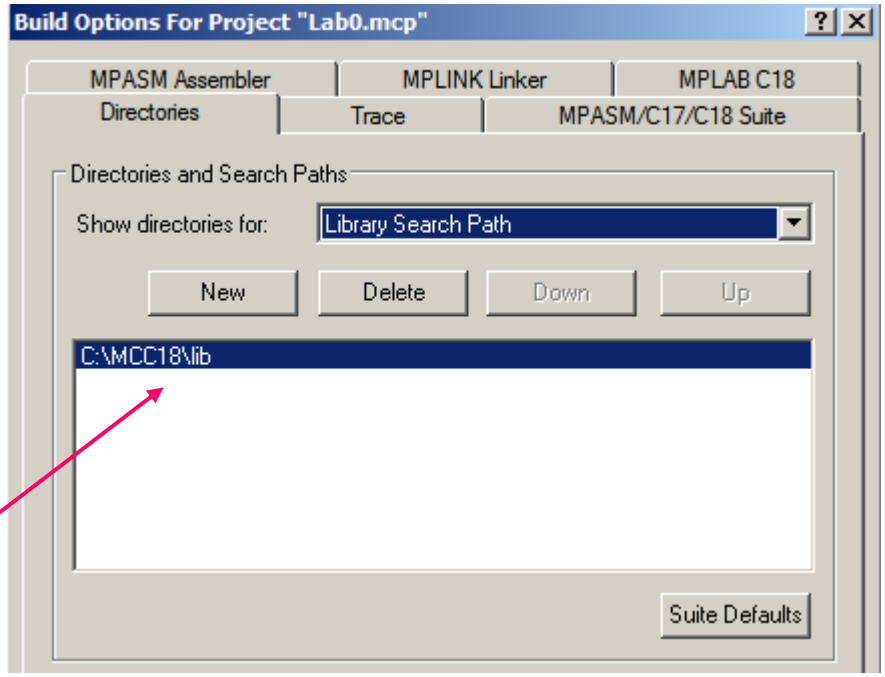
步驟 13



設定 Include & LIB 尋找路徑

步驟 14





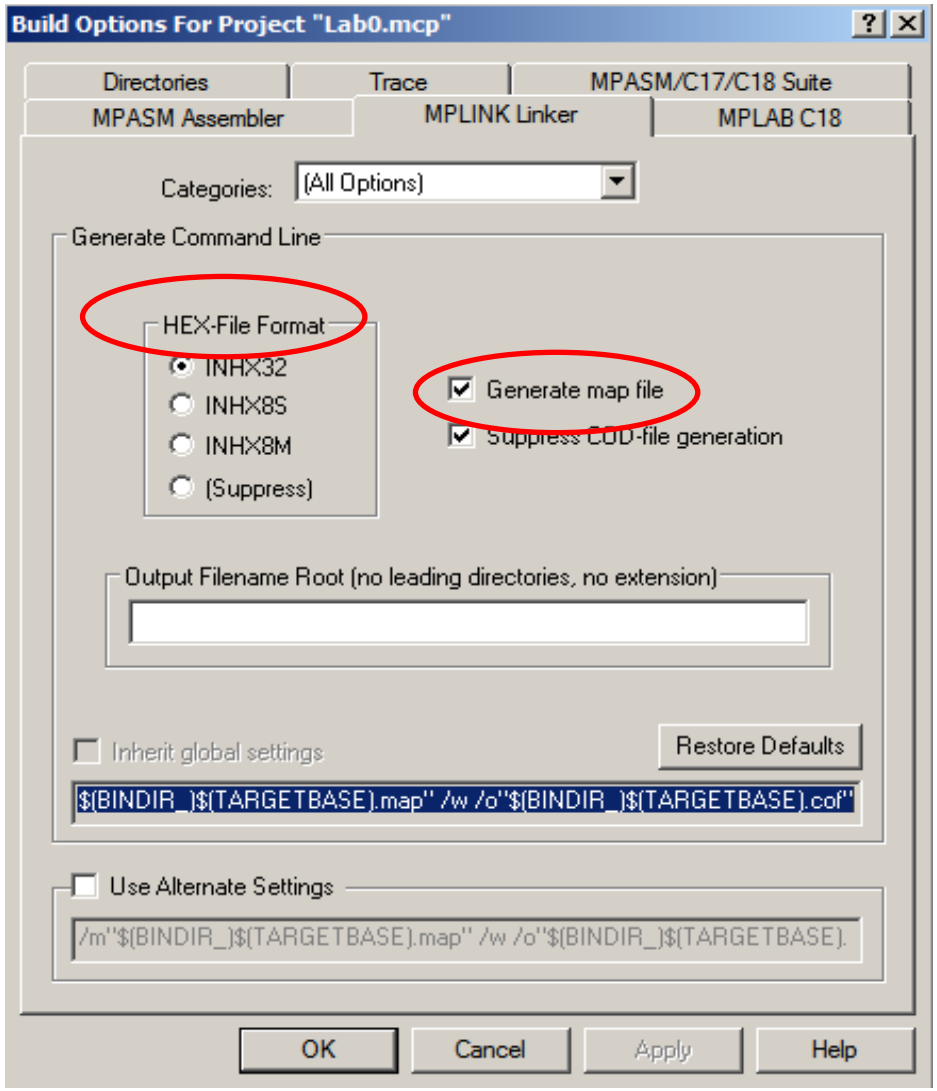
例：編譯時出現了找不到 **c018i.o** 的錯誤

```
MPLINK 4.02, Linker
Copyright (c) 2006 Microchip Technology Inc.
Error - could not find file 'c018i.o'
Errors : 1
```

請設定 c018i.o 的搜尋路徑

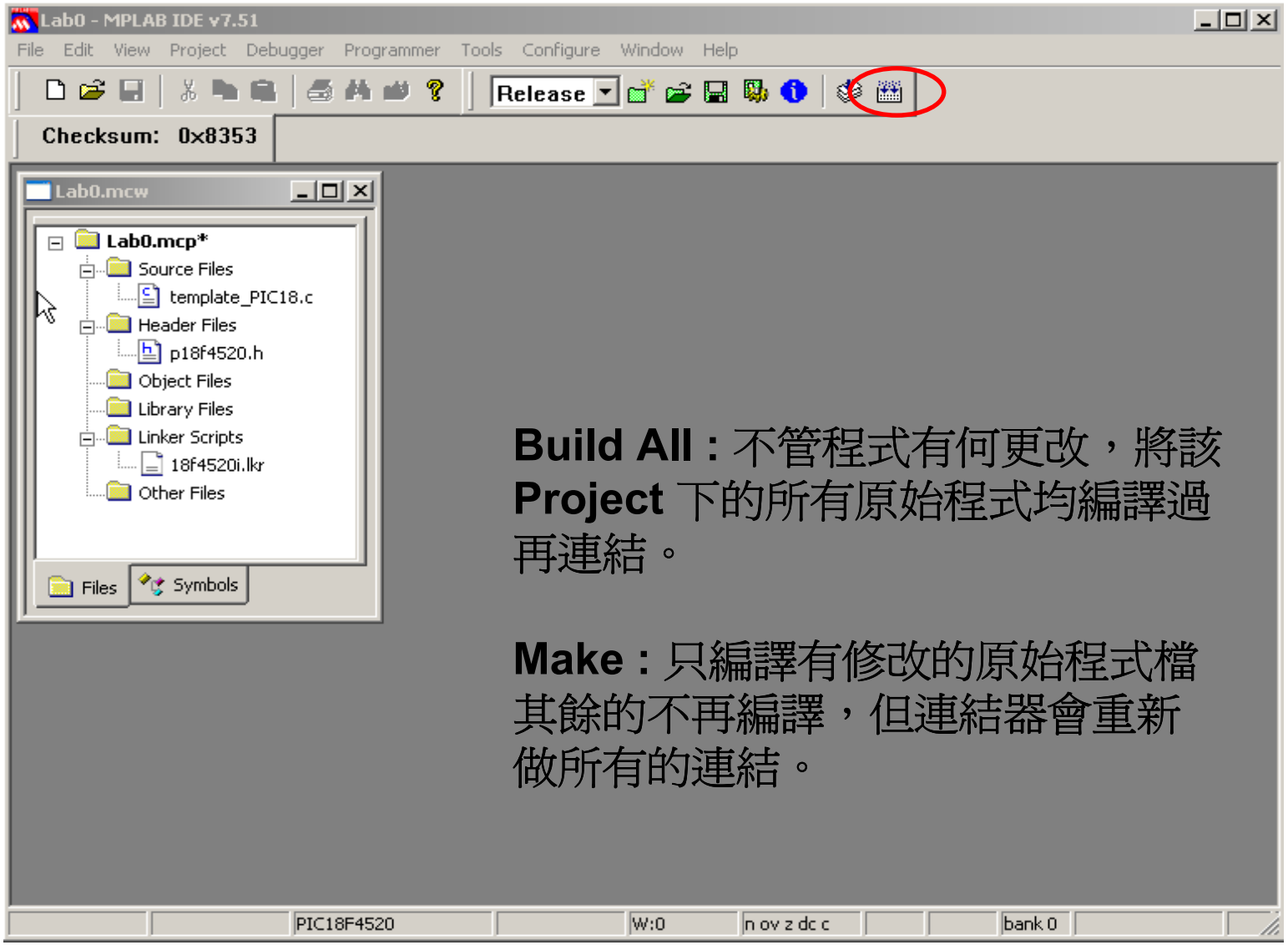
產生 .map 及 HEX 輸出格式

步驟 15



編譯程式 (使用 Build All)

步驟 16

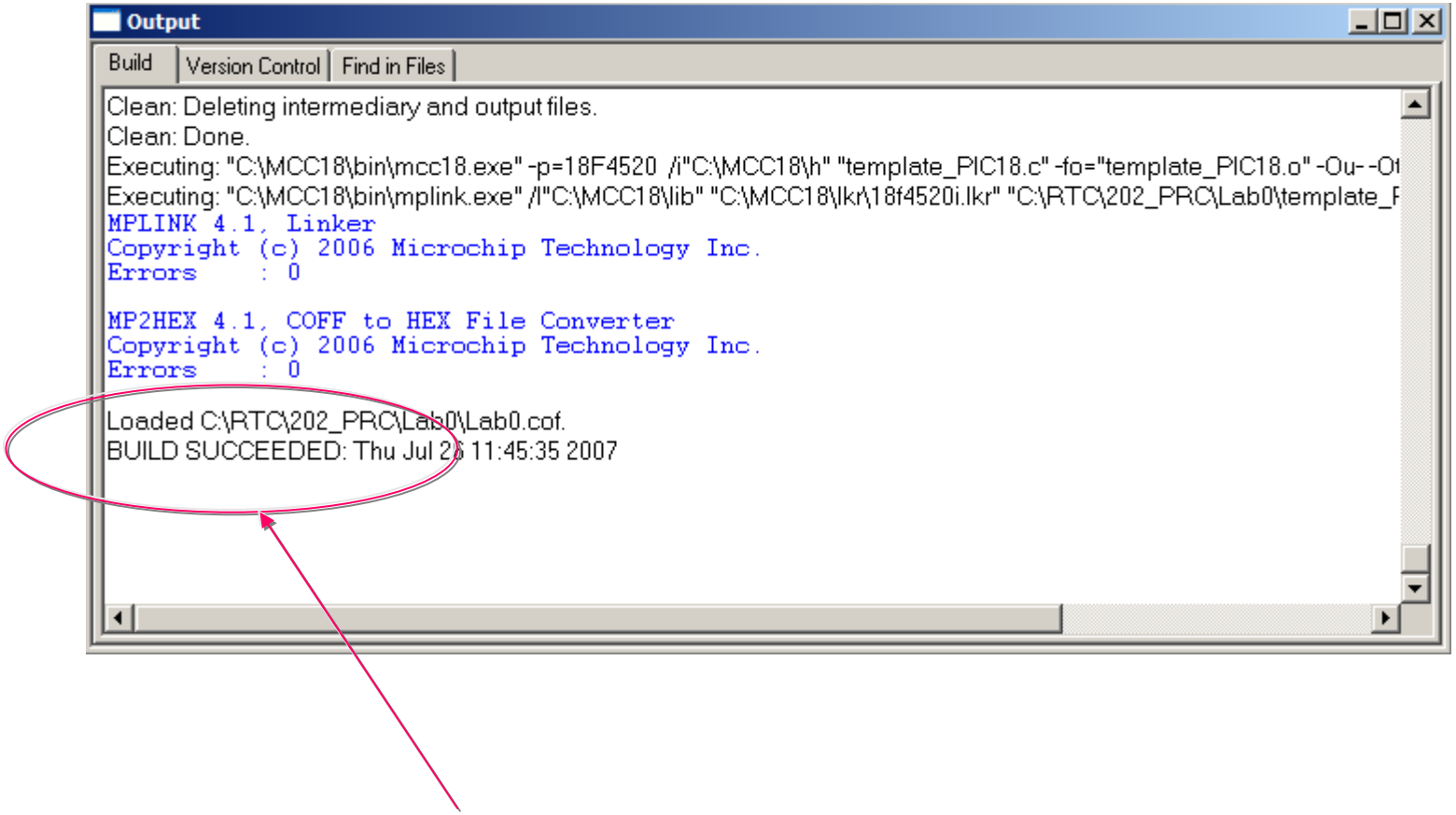


Build All : 不管程式有何更改，將該 **Project** 下的所有原始程式均編譯過再連結。

Make : 只編譯有修改的原始程式檔其餘的不再編譯，但連結器會重新做所有的連結。

確定編譯成功

步驟 17



只有 BUILD SUCCEDED (編譯成功) 出現才會產生Hex 的檔案。

MICROCHIP
Regional Training Centers

如果編譯不成功

- 找不到 **c018i.o**
  - 到 Project → Build Options → Project 下在 Directories 設定路徑 :
    - Include Search Path : C:\MCC18\h
    - Library Search Path : C:\MCC18\lib
    - Linker-Script Search Path : C:\MCC18\lkr

MICROCHIP
Regional Training Centers

安裝 ICD3 USB 驅動程式

(使用 64-bit OS 參考下一頁)

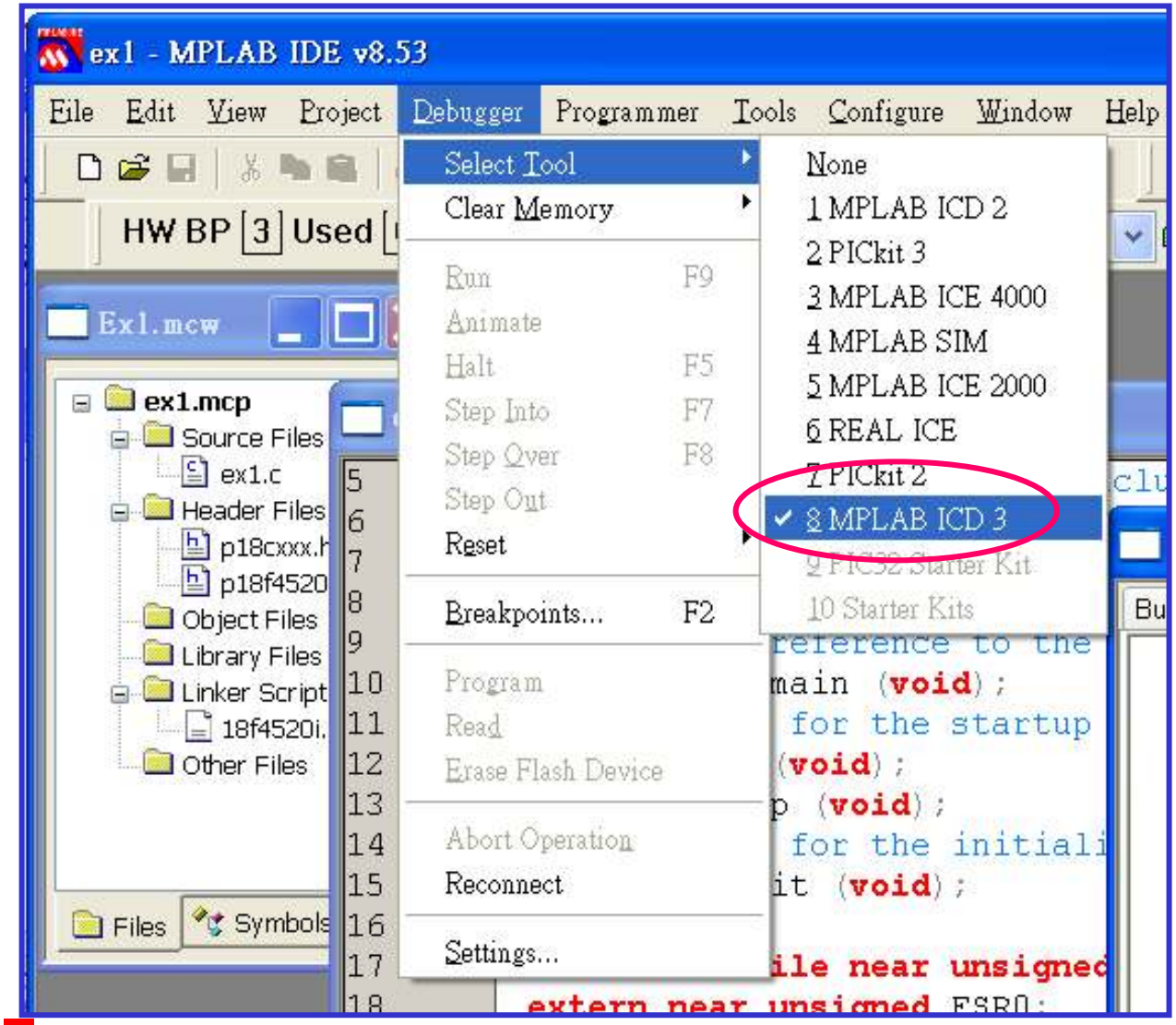
- 初次使用 **ICD3** 時，**MPLAB IDE** 會要求安裝 **USB** 驅動程式 **(PICKit3** 使用 **HID Class** 所以無須特別安裝**)**
- **ICD3** 換到別的 **USB Port** 時也會要求安裝 **USB** 驅動程式
- **ICD3 USB** 驅動程式如何安裝
  - XP 會自動安裝：ICD3 接上 USB 時會有找到新硬體按照 Windows 的指示安裝 (安裝驅動程式目錄如下)
  - Vista & Win7 手動安裝的目錄
    - C:\Program Files\Microchip\ MPLAB IDE\ICD3\Drivers
- 安裝檔案的說明：
  - C:\Program Files\Microchip\ MPLAB IDE\ICD3\Drivers\ddicd3.htm

如果使用 XP 64 / Win Vista 64 / Win 7 64

- 使用 **XP 64 :**
  - C:\Program Files (x86) \Microchip\MPLAB IDE\Drives64\XP64\ddxp64.htm
- 使用 **Vista 64 :**
  - C:\Program Files (x86) \Microchip\MPLAB IDE\Drives64\Vista64\ddvista64.htm
- 使用 **Win 7 64-bit**
  - C:\Program Files (x86) \Microchip\MPLAB IDE\Drives64\Win7_64\ddwin764.htm
- **USB** 硬體搜尋自動安裝的指定路徑
  - C:\Program Files (x86) \Microchip\MPLAB IDE\Drives64

選擇 ICD3 作爲除錯工具

步驟 18



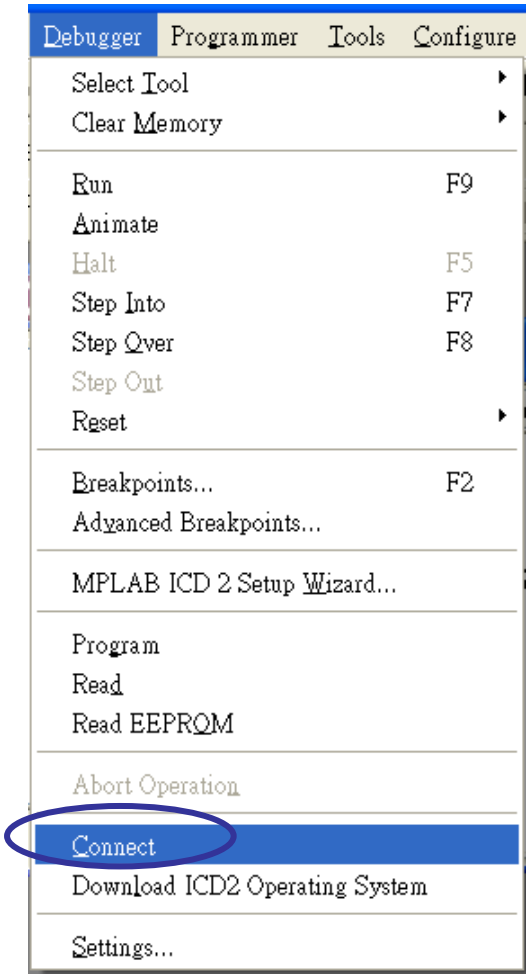
請先確定 ICD3 USB的驅動程式已經正確安裝在 MPLAB IDE 下，安裝方法及路徑請參考:
C:\Program Files\Microchip\MPLAB IDE\ICD3\Drivers\ ddicd3.htm

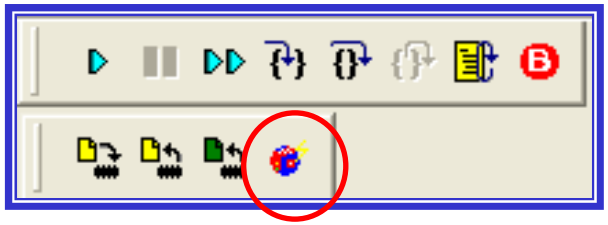
MPLAB ICD3 所支援的元件有數百種之多，當選擇 ICD3 時，MPLAB IDE 會自動檢查 F/W 的版本自行更新軟體。

ICD3 軟體在更新時不可斷線與斷電

連接 ICD3 到 APP001

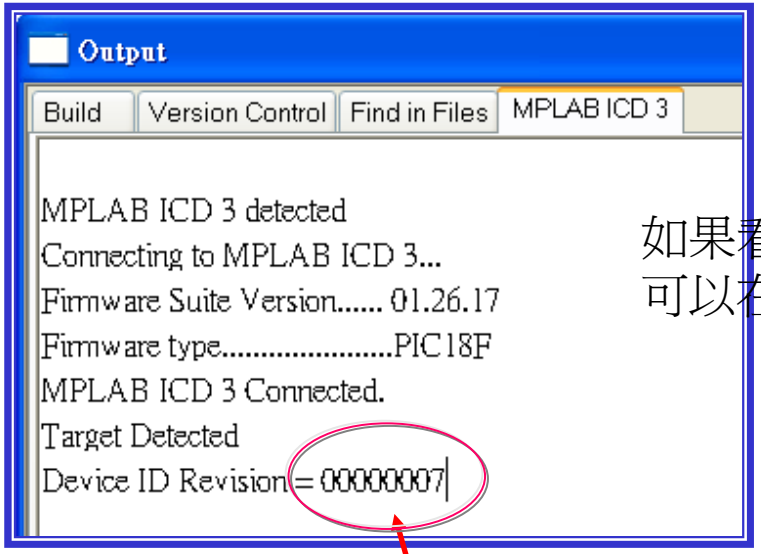
步驟 19





ICD3 除錯選項

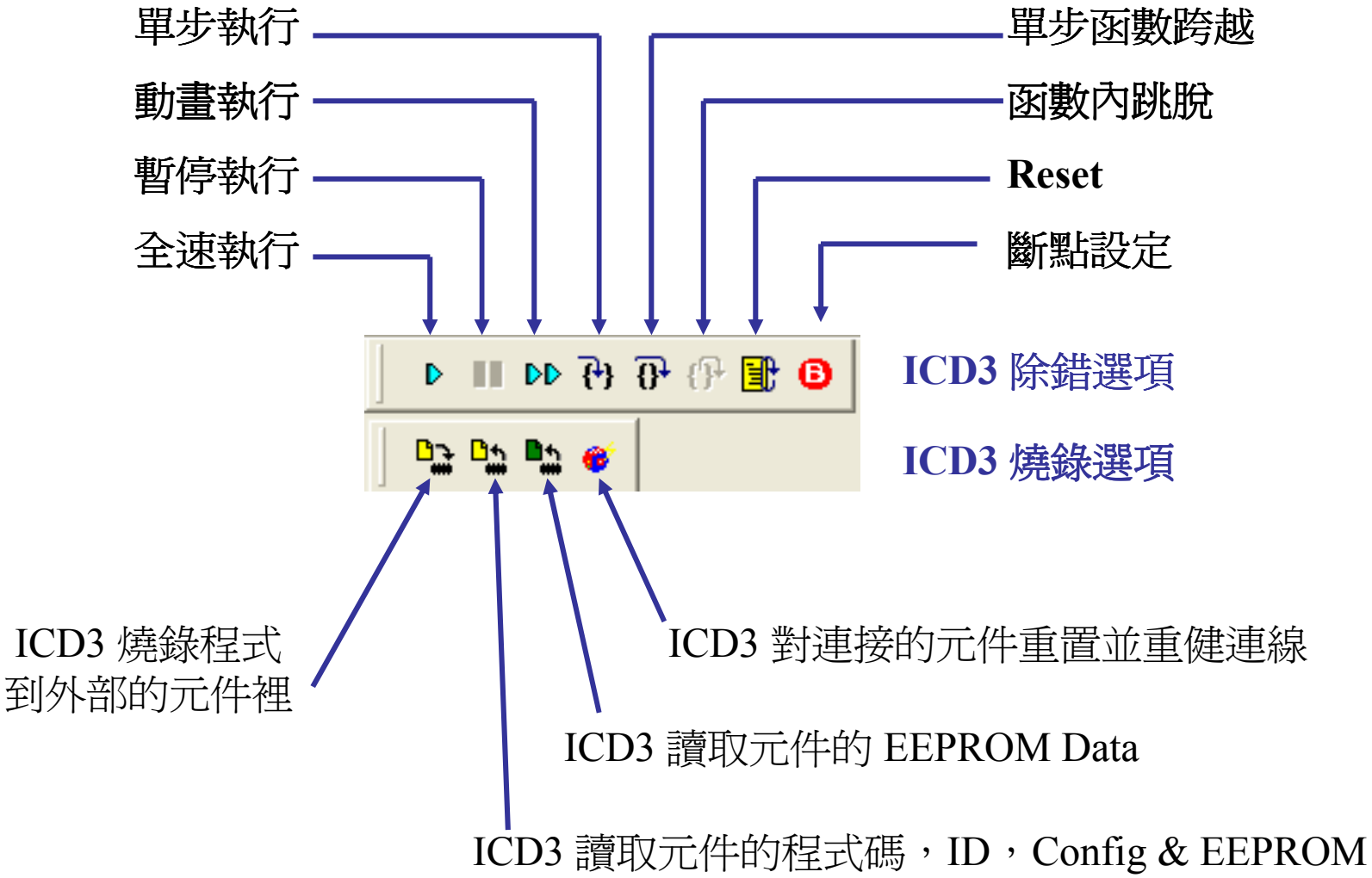
ICD3 燒錄選項



如果看不到 Output 視窗
可以在 view 選項裡開啓

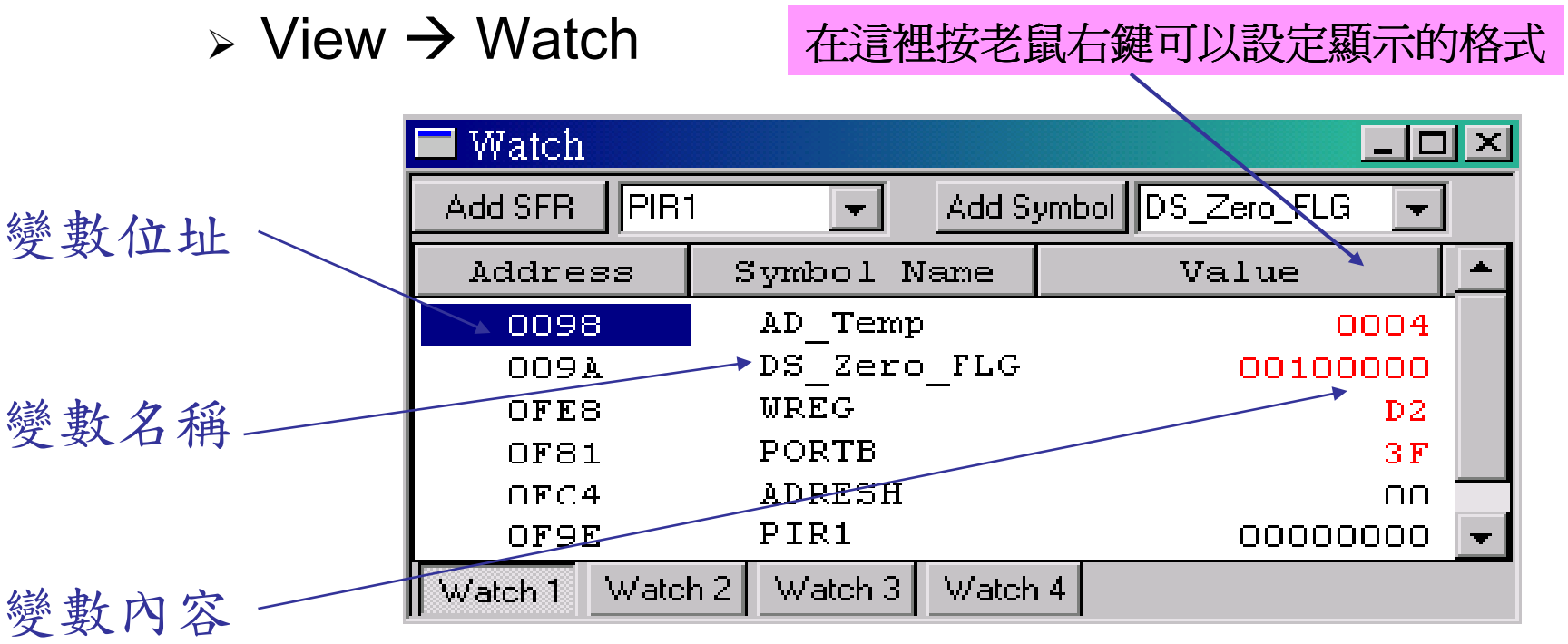
正確的連接到 ICD3 並確定所連到元件爲 PIC18F4520

燒錄 HEX 檔到元件裡除錯 步驟 20

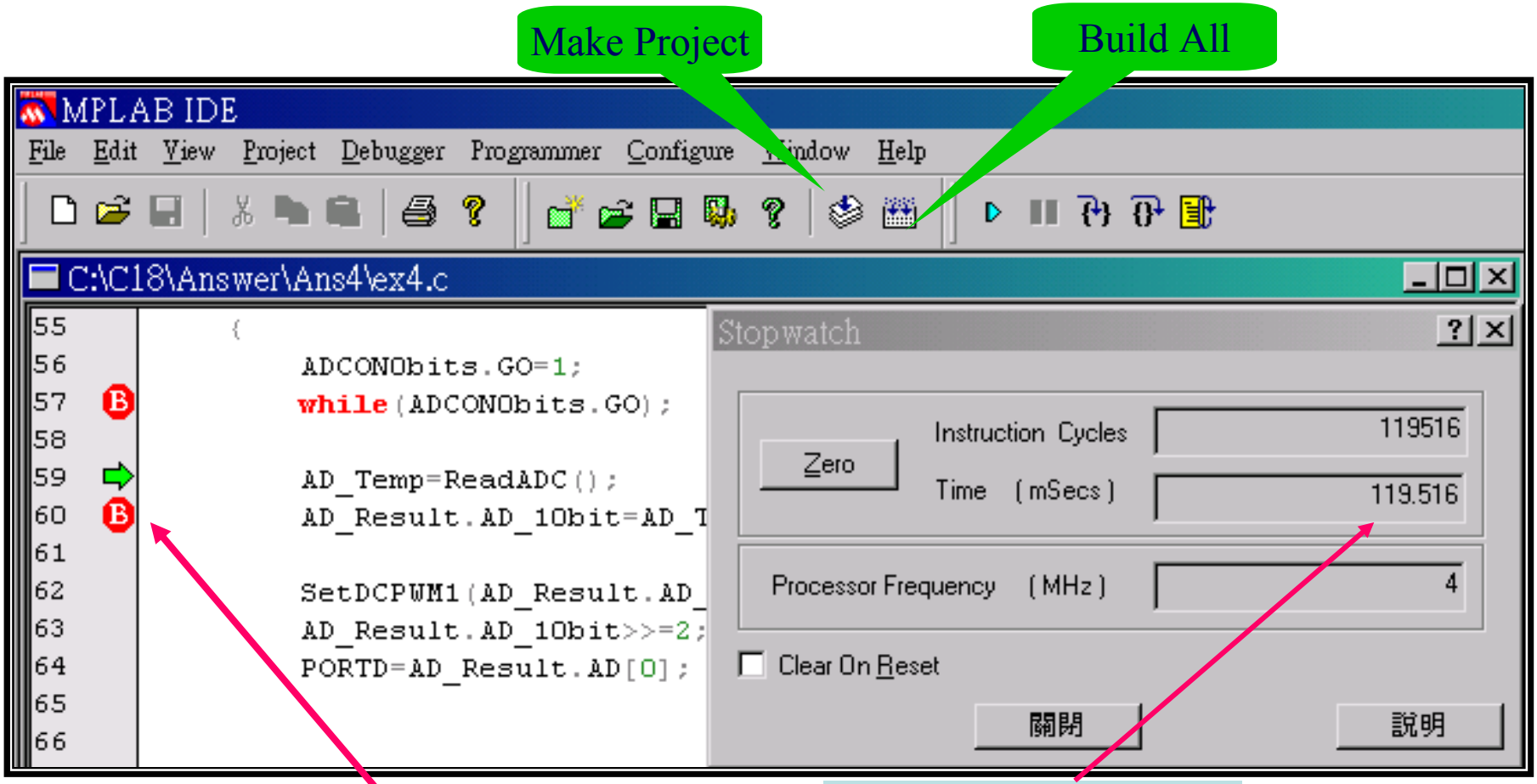


Watch Window 的重要性

- 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



斷點設定及使用軟體模擬的計時器



利用mouse的右鍵來設定斷點

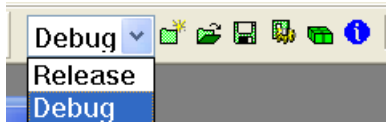
利用Stopwatch視窗來量測程式執行所需的時間

MICROCHIP
Regional Training Centers

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？

Debug 與 Release 模式



- 程式的編譯模式有兩種:
  - Debug Build
    - 一般情況建議都使用此編譯模式
    - 編譯器會保留部分 RAM & Flash 讓除錯軟體使用
    - 設定後須重新編譯過才有作用
  - Release Build
    - 只能使用在 Programmer Mode
    - 在此模式下無法進行除錯

MICROCHIP
Regional Training Centers

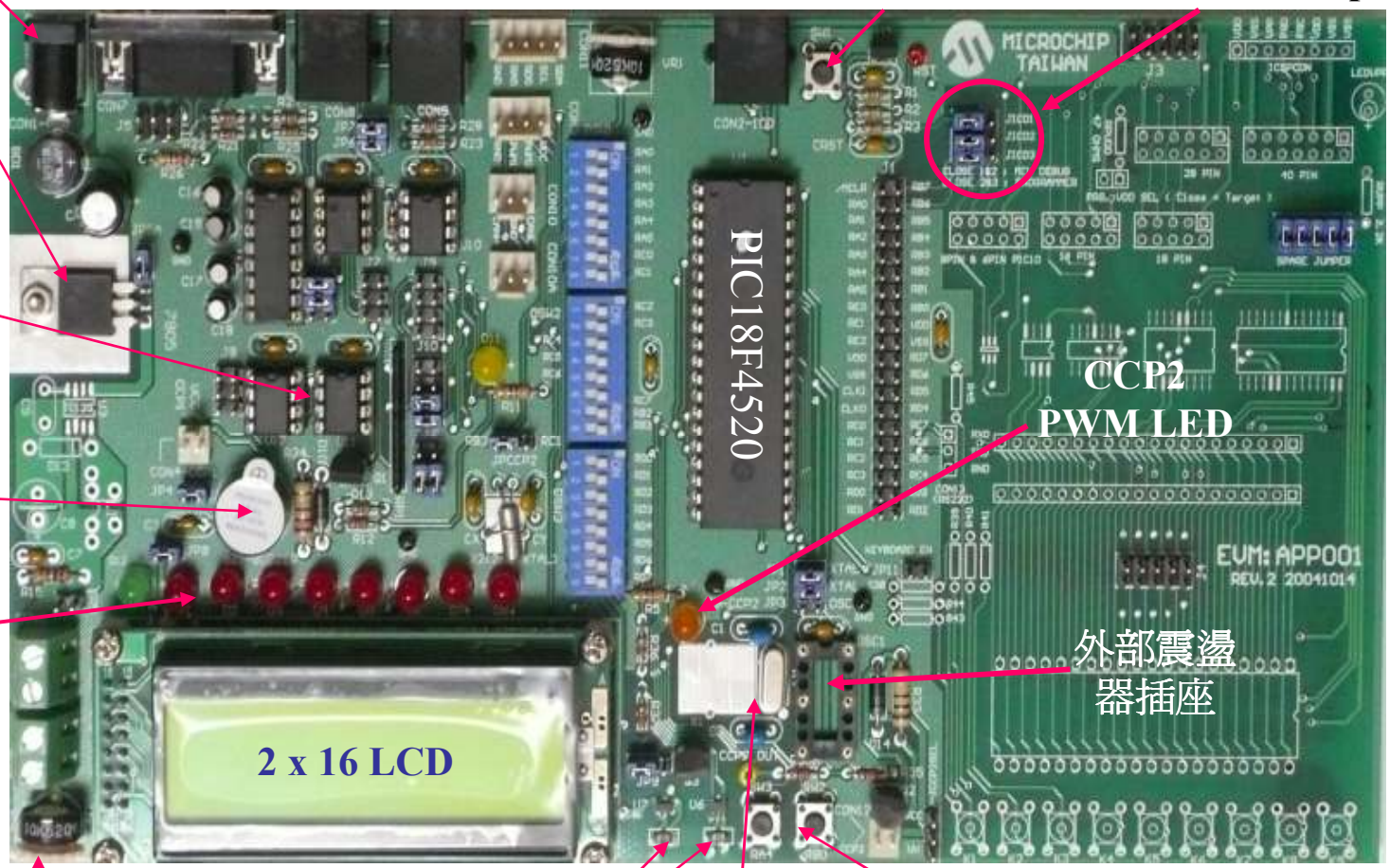
Debugger & Programmer

- **Debugger**
  - 進入除錯模式，程式除錯時使用
  - 燒錄時會將 Debug F/W 一起燒到 PIC 裡
  - 程式的執行由 MPLAB IDE 控管
- **Programmer**
  - 進入燒錄模式
  - 只燒錄程式而已，燒完後即可獨立單機執行，不需 MPLAB IDE 控制
  - 可當作一般的燒錄功能使用

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)
{
 /* User source code here */
}
```
主程式函數

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

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**200mS**

```
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，直到條件式不成立爲止。

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)              // 判斷位元 RB0=1?
       PORTD=0xff ;               // RB0=1,
       PORTD=0xFF
else
       PORTD=0x00 ;               // RB0=0, PORTD=0x00
```

MICROCHIP
Regional Training Centers

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

p18f4520.h 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”
- 位元結構型態可視爲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;
```

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 ;
```

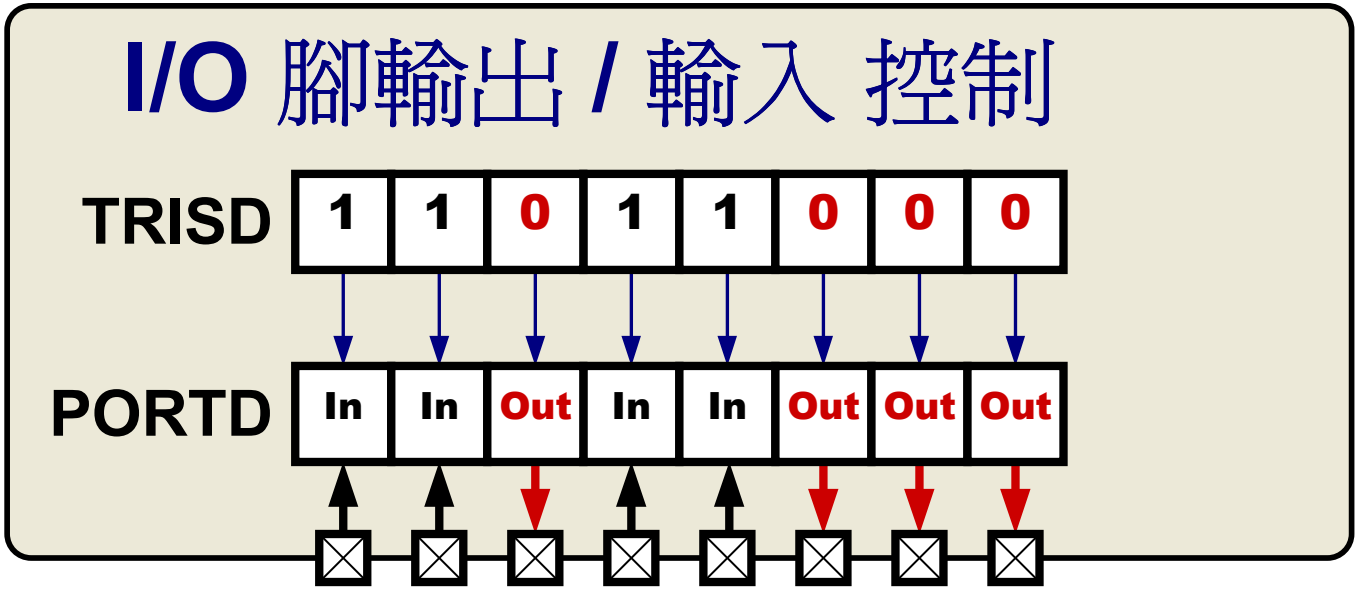
MICROCHIP
Regional Training Centers

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>
  - 因爲 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>
..
```

I/O Port 基本輸出、入設定



- 利用 **TRISD** 直接做 **8-bit I/O** 控制
- 使用 **TRISDbits.TRISDx** 單一 **I/O** 腳控制
  - 1 = Input, 0 = Output

C18 的位元定址 (p18f4520.h)

- 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。對應的位元結構爲暫存器名稱加上 **“bits”**

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

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

2. 將 **PORTD** 的 **bit3 & bit5** 設定爲 **1**：

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

P18F4520.lib 周邊函數庫檔

- 定義**18F4520**所有的特殊周邊暫存器**(SFR)**的位址
  - P18F4520.o 是以 obj 的型態存在於 P18F452.LIB
  - 原始程式爲組合語言型態放在
    - C:\MCC18\src\traditional\proc
- **p18f4520.lib** 提供各周邊控制函數
  - 相關的周邊控制函數，各自分門別類的以 obj 型態存在於 p18f4520.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_common 的目錄下

PIC18F4520 三個元件支援檔

- **p18f4520.o (asm)**
  - 定義周邊暫存器的位址給 C18 使用
- **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

有關特殊暫存器之位址定義

```
  LIST P=18F4520
       NOLIST
;-----------------------------------------------------
; MPLAB-Cxx  PIC18F4520 processor definition module
;
; (c) Copyright 1999-2007 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
LATC
LATCbits            RES 1     ; 0xF8B
LATD
LATDbits            RES 1     ; 0xF8C
LATE
LATEbits            RES 1     ; 0xF8D
```

MICROCHIP
Regional Training Centers

MPLAB-C18 啓動模組

- 啓動模組的功能(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
- 啓動模組有三個，放在 **clib.lib** 函數庫裡
  - c018.o - 無須爲變數規劃初始值時使用
  - c018i.o - 須爲變數規劃初始值時使用
  - c018iz.o –先將RAM清零，再爲變數規劃初始值
  - 原始程式位置：C:\MCC18\src\traditional\startup

C018i.c 啓動模組程式

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

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

  _do_cinit ( );

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

RESET 位址:
0x000000

給予 STACK 及
TBLPTRU 初使值

設定初始變數值

將控制權交至 main( )

在 18f4520i.lkr 描述檔

```
// File: 18f4520i.lkr
// Sample ICD2 linker script for the PIC18F4520 processor

LIBPATH  .

FILES c018i.o
FILES clib.lib
FILES p18f4520.lib

CODEPAGE   NAME=page       START=0x0          END=0x7DBF
CODEPAGE   NAME=debug      START=0x7DC0       END=0x7FFF        PROTECTED
CODEPAGE   NAME=idlocs     START=0x200000     END=0x200007      PROTECTED
CODEPAGE   NAME=config     START=0x300000     END=0x30000D      PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE     END=0x3FFFFF      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=0x5F3
DATABANK   NAME=dbgspr     START=0x5F4        END=0x5FF         PROTECTED
ACCESSBANK NAME=accesssfr  START=0xF80        END=0xFFF         PROTECTED

SECTION    NAME=CONFIG     ROM=config

STACK SIZE=0x100 RAM=gpr4
```

連結起動模組

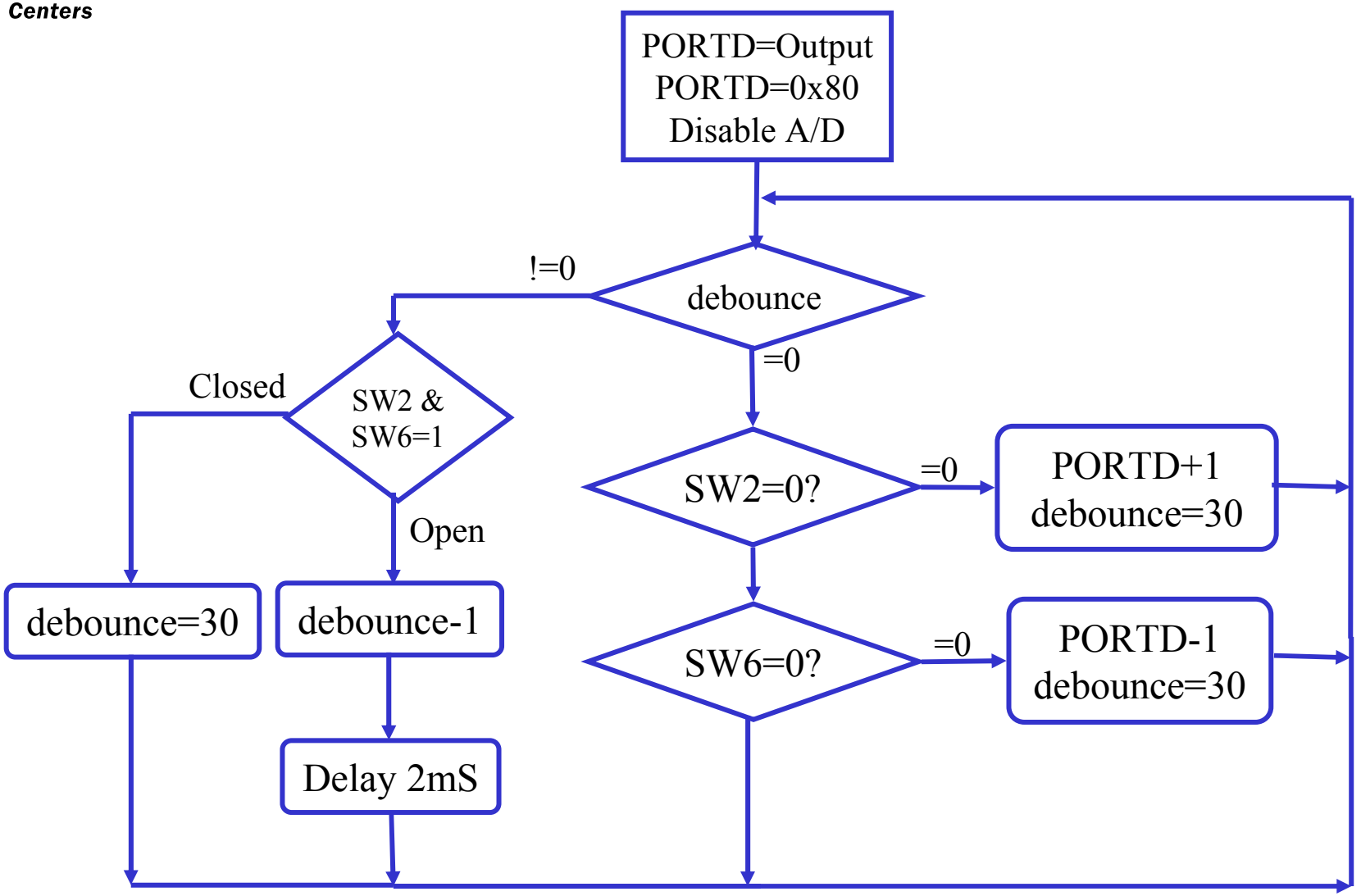
連結標準函數庫

連結18F4520周邊函數庫

練習 2
PORT 的輸出與按鍵輸入

- 開啓 **"\RTC\W401\Ans2\Ex2.mcp"**
- 將 **PORTD (LED)** 設定爲輸出模式
- 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濾此種現象
- 參考下頁之流程圖

練習二程式流程說明


PORTD=Output
PORTD=0x80
Disable A/D
debounce
!=0
=0
SW2 &
SW6=1
Closed
Open
debounce=30
debounce-1
Delay 2mS
SW2=0?
=0
PORTD+1
debounce=30
SW6=0?
=0
PORTD-1
debounce=30

第三章

基本周邊功能操作

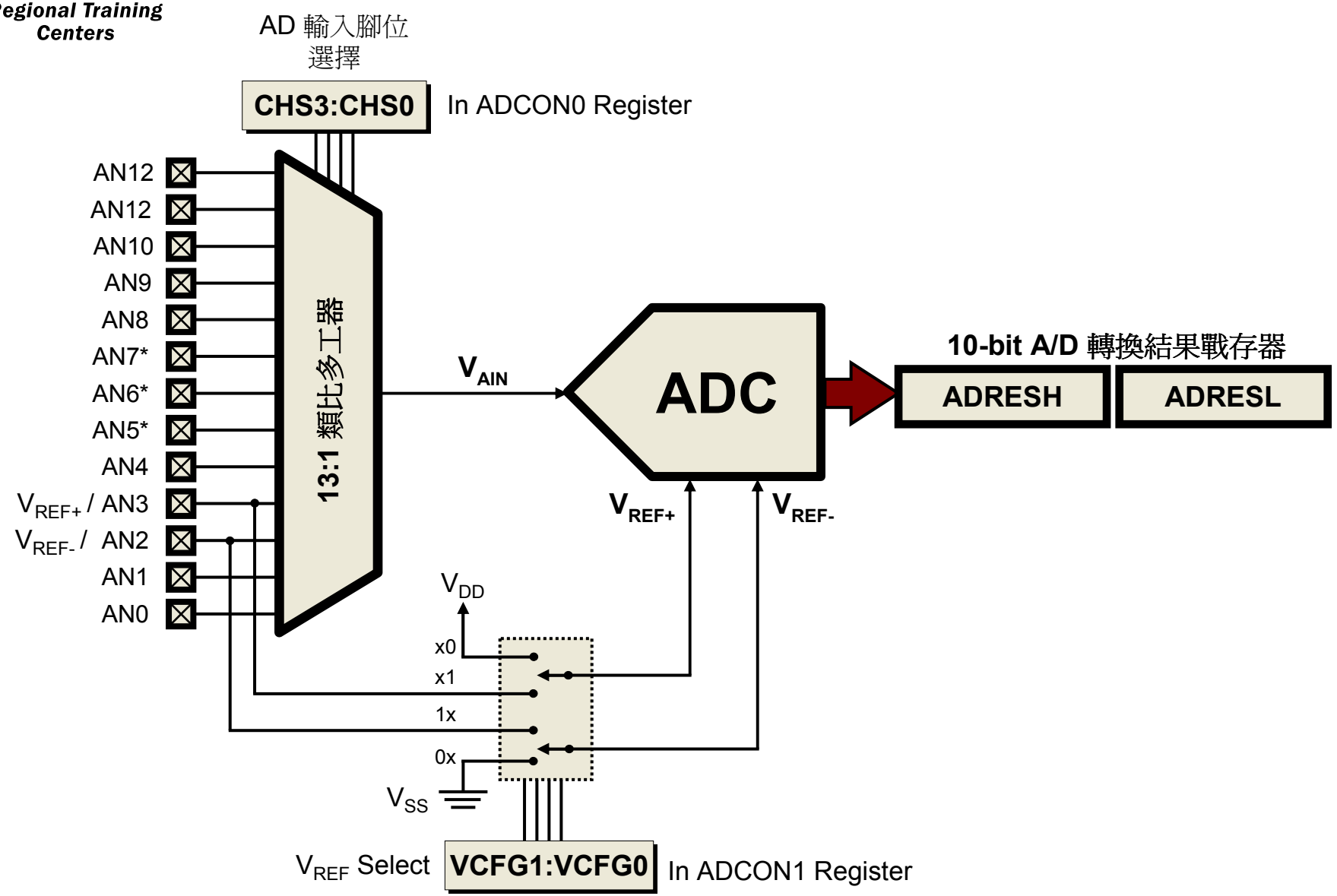
1. 基本10位元A/D轉換
2. 10位元解析度的PWM控制
3. 調整馬達轉速

MICROCHIP
Regional Training Centers

10-bit A/D 轉換器

- **13** 組類比轉換多工輸入選擇，**10 bits** 解析度
- **AD** 最小時脈週期 **(T_{AD}) : 0.7uS** (外部震盪器)
- 類比輸入取樣時間：**1.4 µS** (輸入阻抗**<10K**)
- 類比輸入轉換時間：**8.4 µS (12 T_{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 **9.8 µs**

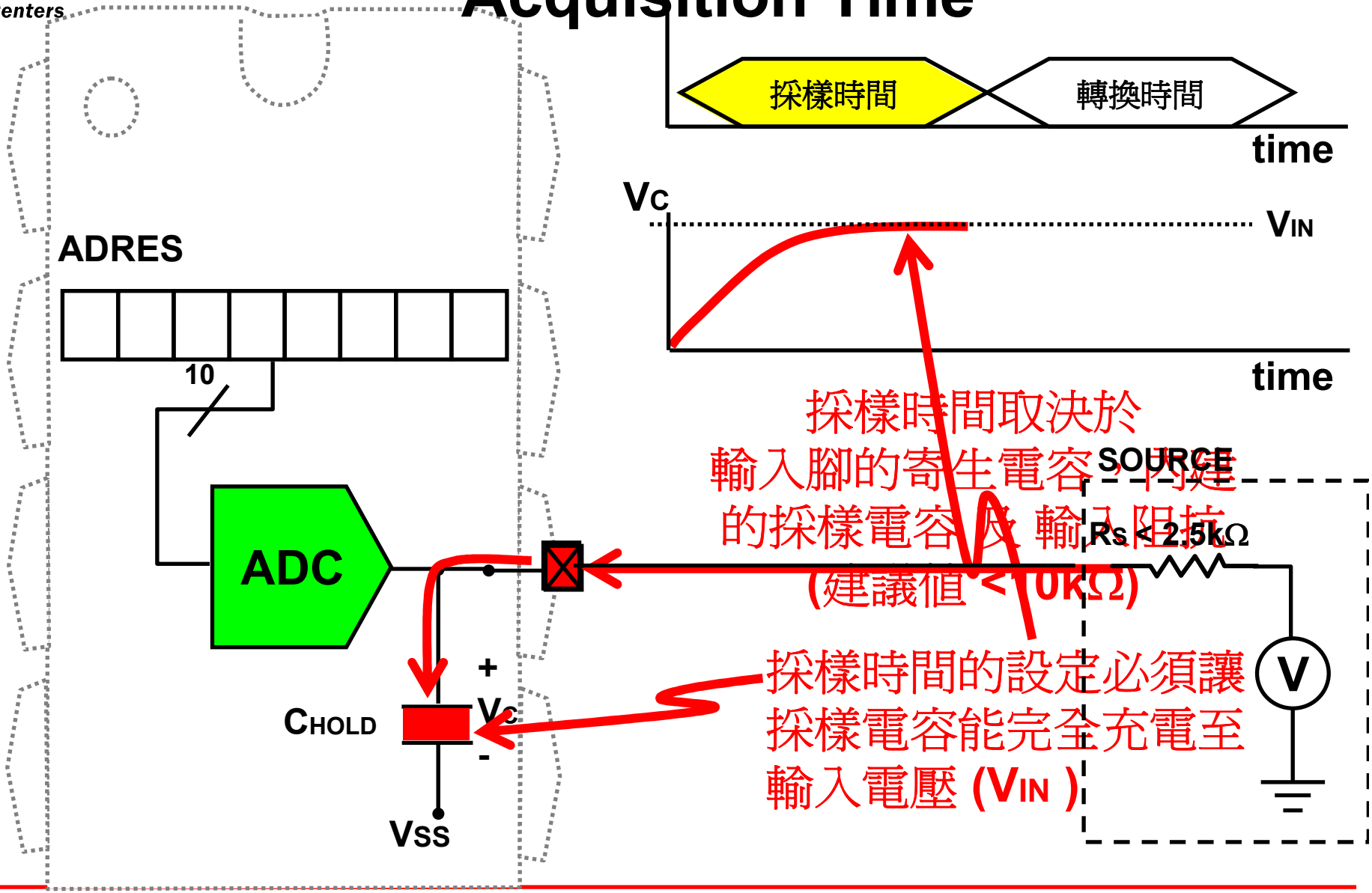
10-bit A/D 轉換器



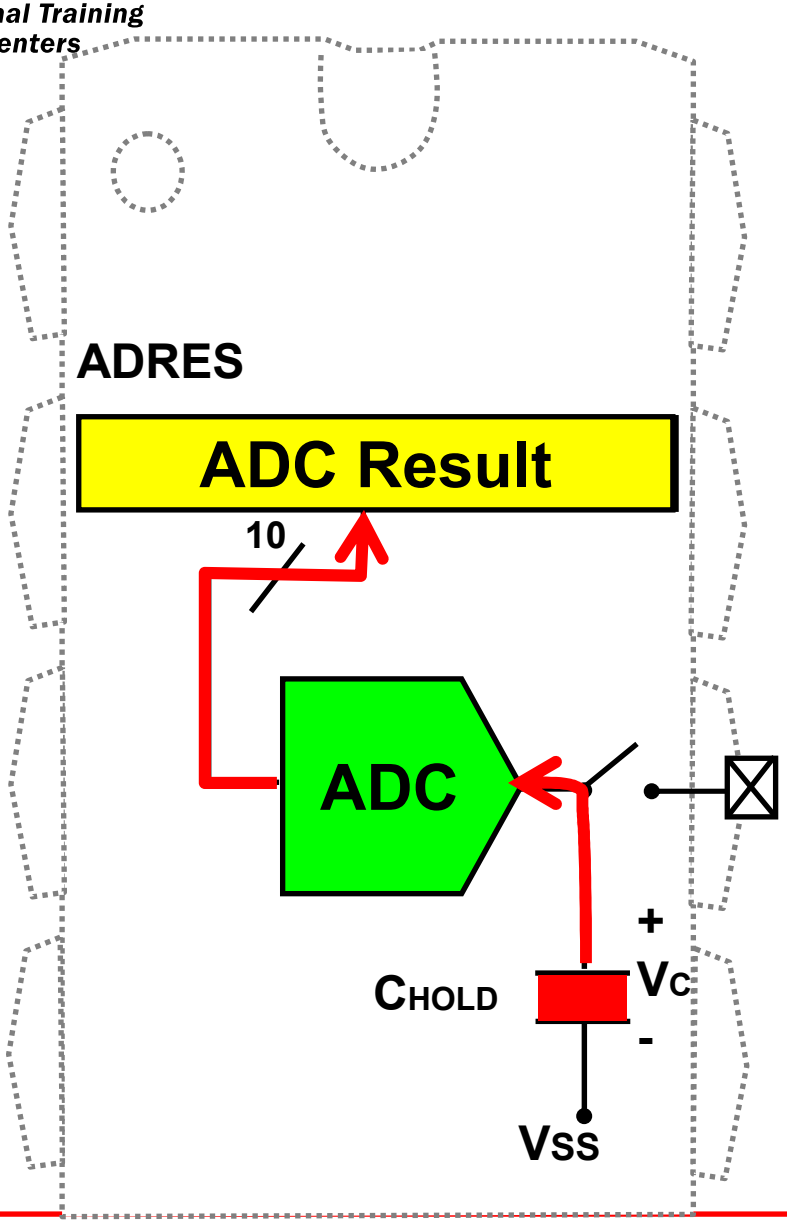
*AN5-AN7 not available on 28-pin devices

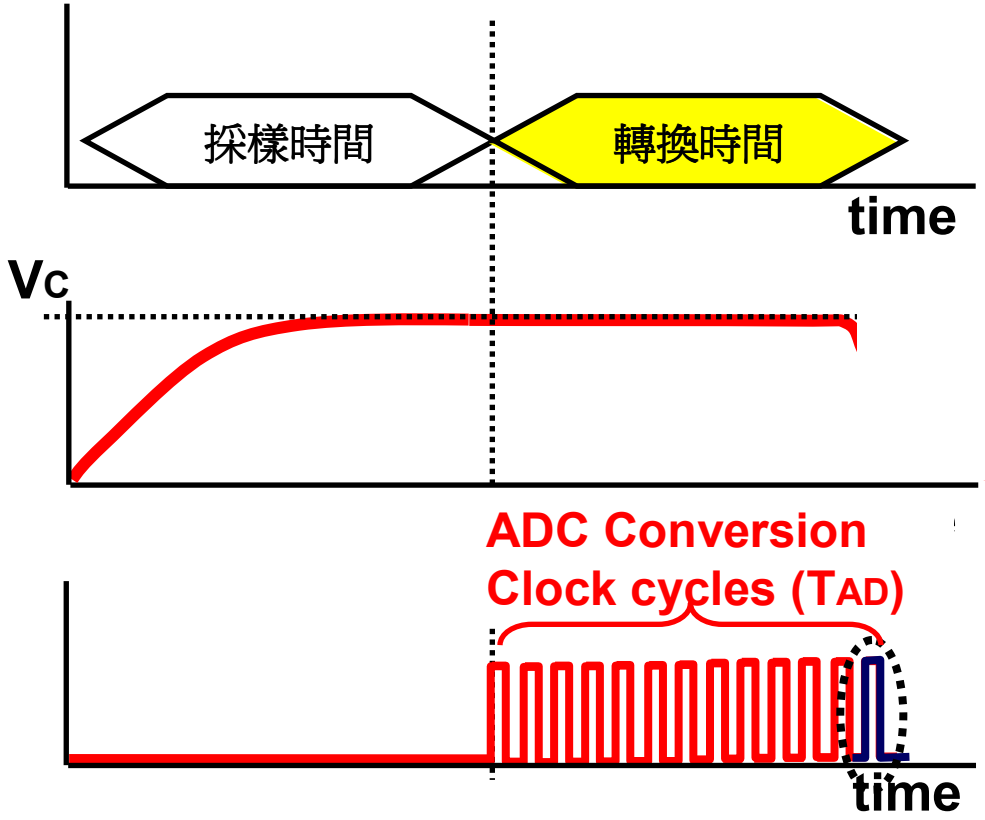
輸入訊號採樣時間

Acquisition Time



ADC 轉換時間


ADRES
ADC Result
10
ADC
+
VC
-
CHOLD
VSS


採樣時間
轉換時間
time
VC
ADC Conversion
Clock cycles (TAD)
time

18F 系列 ADC 的主要差異

- 舊式的 **ADC (V1)** 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** 爲單位**),** 使用者只要切換 **Channel** 後啓動轉換**, ADC** 會先取樣 **nTAD** 後再轉換 **(ADC_V5** 版本**)**
  - 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 的轉換時間爲 19.2 uS
  - PIC18F4520 的轉換時間只要 8.4 uS

10-bit A/D 轉換器
ADCON0 暫存器設定

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/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

bit 5-2 **CHS3:CHS0**
類比輸入通道選擇
0000 = Channel 0 (AN0)
0001 = Channel 1 (AN1)
0010 = Channel 2 (AN2)
0011 = Channel 3 (AN3)
0100 = Channel 4 (AN4)
0101 = Channel 5 (AN5)
0110 = Channel 6 (AN6)
0111 = Channel 7 (AN7)
1000 = Channel 8 (AN8)
1001 = Channel 9 (AN9)
1010 = Channel 10 (AN10)
1011 = Channel 11 (AN11)
1100 = Channel 12 (AN12)

bit 1 **GO/DONE:** A/D 轉換狀態位元
1 = A/D 正在做轉換動作
0 = A/D 轉換器閒置

bit 0 **ADON:** A/D On bit
1 = A/D 開啓 ADC 模組
0 = A/D 關閉 ADC 模組

10-bit A/D 轉換器

ADCON1 暫存器設定

ADCON1 Register

U-0	U-0	R/W-0	R/W-0	R/W-0*	R/W*	R/W*	R/W*
-	-	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 5 **VCFG1:** 負參考電壓 (V_{REF-}) 設定位元
1 = V_{REF-} (AN2)
0 = V_{SS}

bit 4 **VCFG0:** 正參考電壓 (V_{REF+}) 設定位元
1 = V_{REF+} (AN3)
0 = V_{DD}

10-bit A/D 轉換器

ADCON1 暫存器設定

ADCON1 Register

U-0	U-0	R/W-0	R/W-0	R/W-0*	R/W*	R/W*	R/W*
-	-	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 3-0 **PCFG3:PCFG0**

數位腳位 或 A/D 類比輸入腳位設定

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0000	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A =類比輸入腳位
D =數位腳位

10-bit A/D 轉換器
ADCON2 暫存器設定

ADCON2 Register

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

設定 T_{AD} 的來源及時間 (轉換每一位元所需的基本時間)

bit 2-0 **ADCS2:ADCS0:** A/D Conversion Clock Selection bits

111 = F_{RC} (Clock derived from A/D RC oscillator)

110 = $F_{OSC}/64$

101 = $F_{OSC}/16$

100 = $F_{OSC}/4$

011 = F_{RC} (Clock derived from A/D RC oscillator)

010 = $F_{OSC}/32$

001 = $F_{OSC}/8$

000 = $F_{OSC}/2$

使用內部 **RC** 震盪器 T_{AD} 時間固定爲 **1.2us**

使用外部震盪器 最小的 T_{AD} 設定不可少於 **0.7us**

10-bit A/D 轉換器
ADCON2 暫存器設定

ADCON2 Register

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

bit 5-3 **ACQT2:ACQT0:** A/D 採樣時間設定位元

111 = 20 T_{AD}

110 = 16 T_{AD}

101 = 12 T_{AD}

100 = 8 T_{AD}

011 = 6 T_{AD}

010 = 4 T_{AD}

001 = 2 T_{AD}

000 = 0 T_{AD}

如果需求:
T_{ACQ} = 4.2 µs
T_{AD} = 1uS

所以設定爲:
ACQT2:0 = 011 = 6.0 µs

10-bit A/D 轉換器

ADCON2 暫存器設定

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

bit 7 **ADFM:** A/D 轉換結果輸出格式設定位元

1 = 向右靠齊

ADRESH

						b_9	b_8

ADRESL

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

0 = 向左靠齊

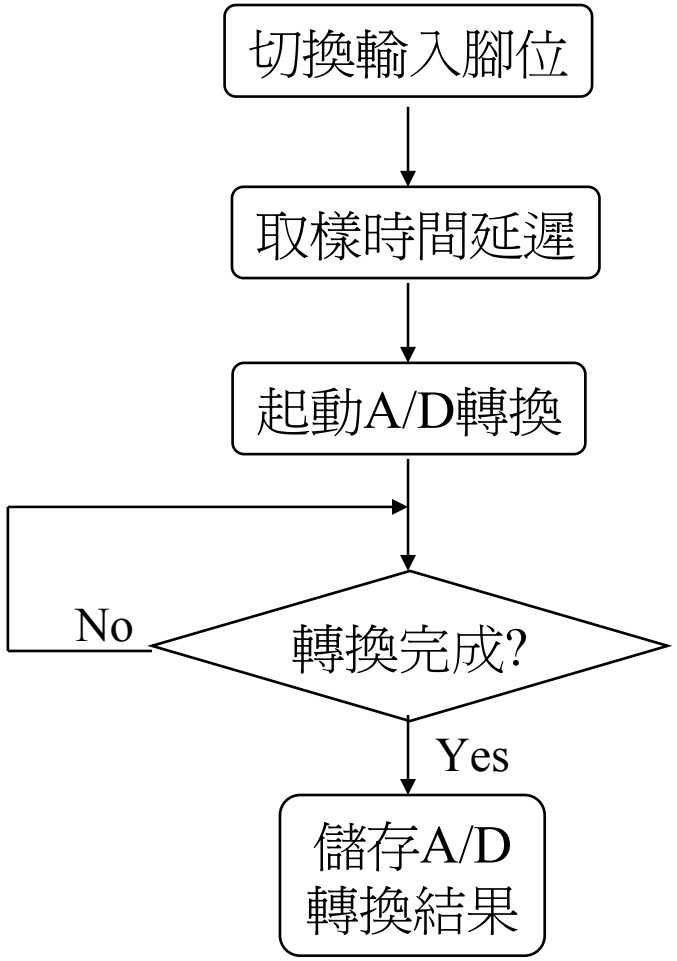
ADRESH

b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2

ADRESL

b_1	b_0						

A/D 轉換基本流程



SetChanADC( )

Delay 20uS

ConvertADC( )
or
ADCON0bits.GO=1

while(BusyADC( ))
or
While(ADCON0bits.GO)

A_D=ReadADC( )

A/D 轉換程式比較

組合語言

```
        list   p=18f452
        include <p18f452.inc>
main    clrf   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   $-2
        movf   ADRESH,W
        return
```

C 語言

```
#include <p18f452.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;
}
```

使用C18的 A/D 轉換函數

- 使用 **C18** 提供的周邊函式庫時, 要參考 **Help File** 中的說明
- 以 **ADC** 爲例, 可以在 **C:\mcc18\doc** 找到 **AD Converter.htm**
- 參考其中的內容, 可以知到使用的 **ADC** 屬於那一個 **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
ADC_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

PIC18F4520 爲 V5

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>
- ADC_V5 版本下的 OpenADC 的 Prototype 宣告

```
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( );` // AD 開始轉換
  - `While(BusyADC( );` // 等待 AD 完成轉換

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

  - `SetChanADC(ADC_CH0);`

```
int ReadADC (void)
```

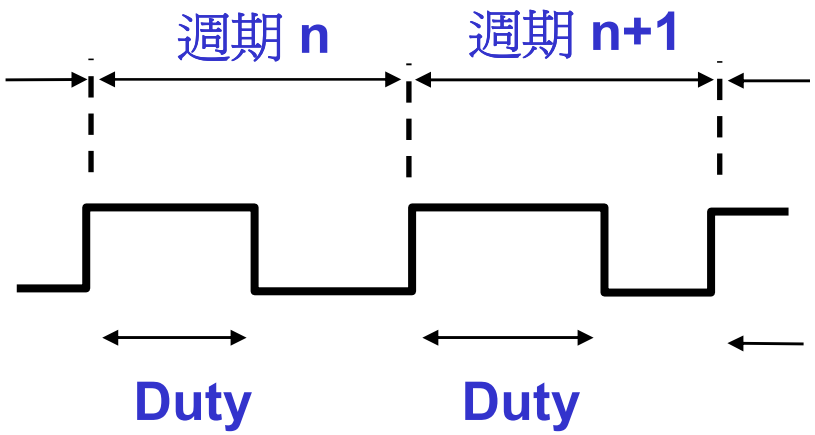
  - `int result;`
  - `result = ReadADC( );`

MICROCHIP
Regional Training Centers

練習 3-1
ADC 的基本轉換

- 開啓 “\RTC\W401\Ans3-1\Ex3-1.mcp”
- 利用 A/D 函數庫來撰寫
- 將 PORTD (LED) 設定爲輸出模式
- 設定 AN0 爲 A/D 輸入端，其餘均設爲一般數位 I/O 腳功能
- 調整 VR1 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

PWM 基本說明



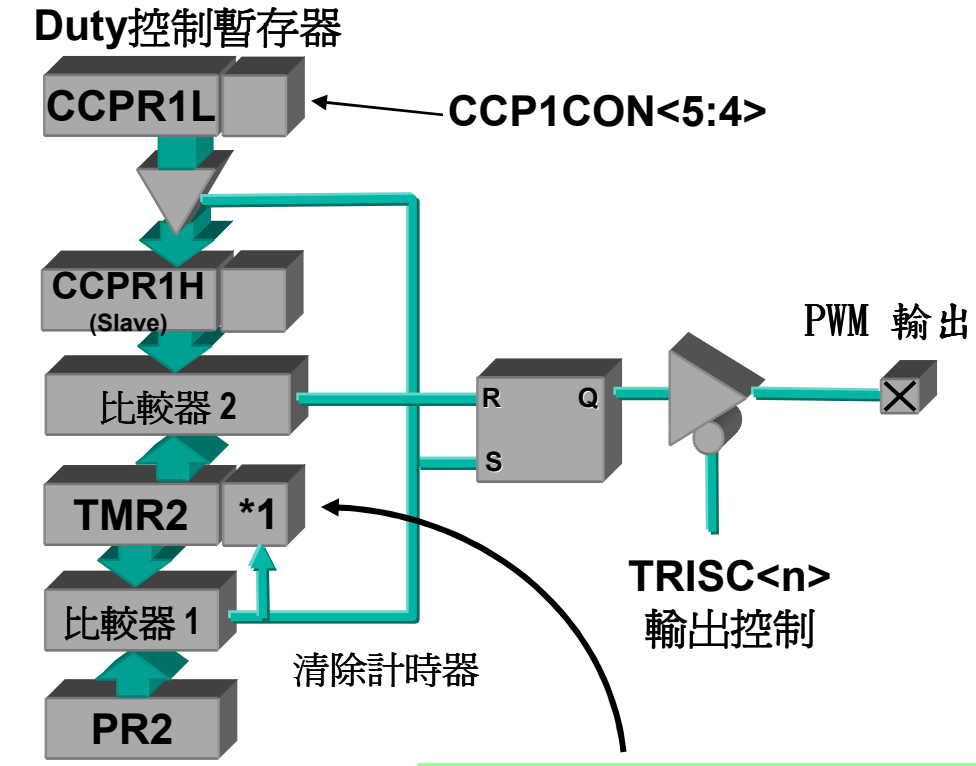
- 週期的時間均固定，亦即頻率固定不變
- **Duty** 所佔的時間是可調整的

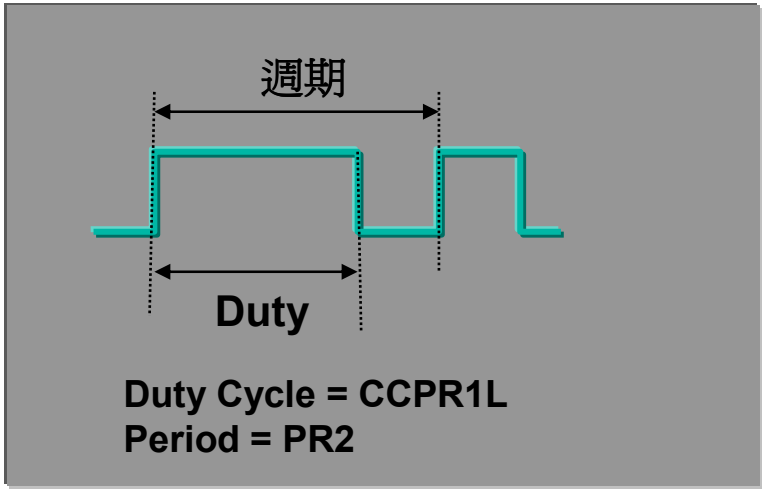
若振幅爲 5V ，Duty 爲 38%，則平均輸出電壓爲：

*5V * 0.38 = 1.9V*

18F4520 PWM 原理

無雜訊脈衝的PWM原理





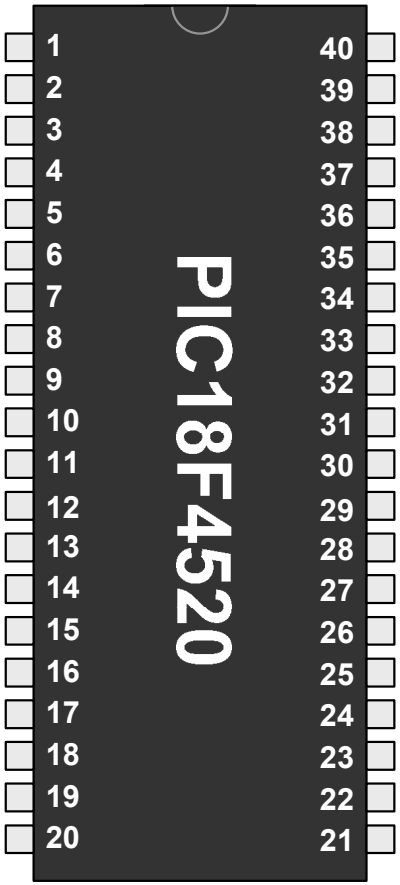
f_{osc} = 40MHz

解析度	最高PWM頻率
10-bits	39.06kHz
8-bits	156.24kHz
7-bits	312.48kHz
5-bits	1249.92kHz

* 8位元的Timer2和2位元的預除器(Q Clock)
組成10位元的Duty比較計時器

PWM 的輸出腳設定
Configure Bits 設定從 CCP2 腳輸出

- 將 **TRISC1** 設成輸出腳功能，**PWM2** 自 **RC1** 腳送出



RB3/AN9/CCP2(1)

(1) Optional CCP2 on RB3:
If CONFIG3H<CCP2MX> = 0

TRISB<3> = 0

RC1/T1OSI/CCP2(1)

(1) Default CCP2 on RC1:
If CONFIG3H<CCP2MX> = 1

TRISC<2> = 0

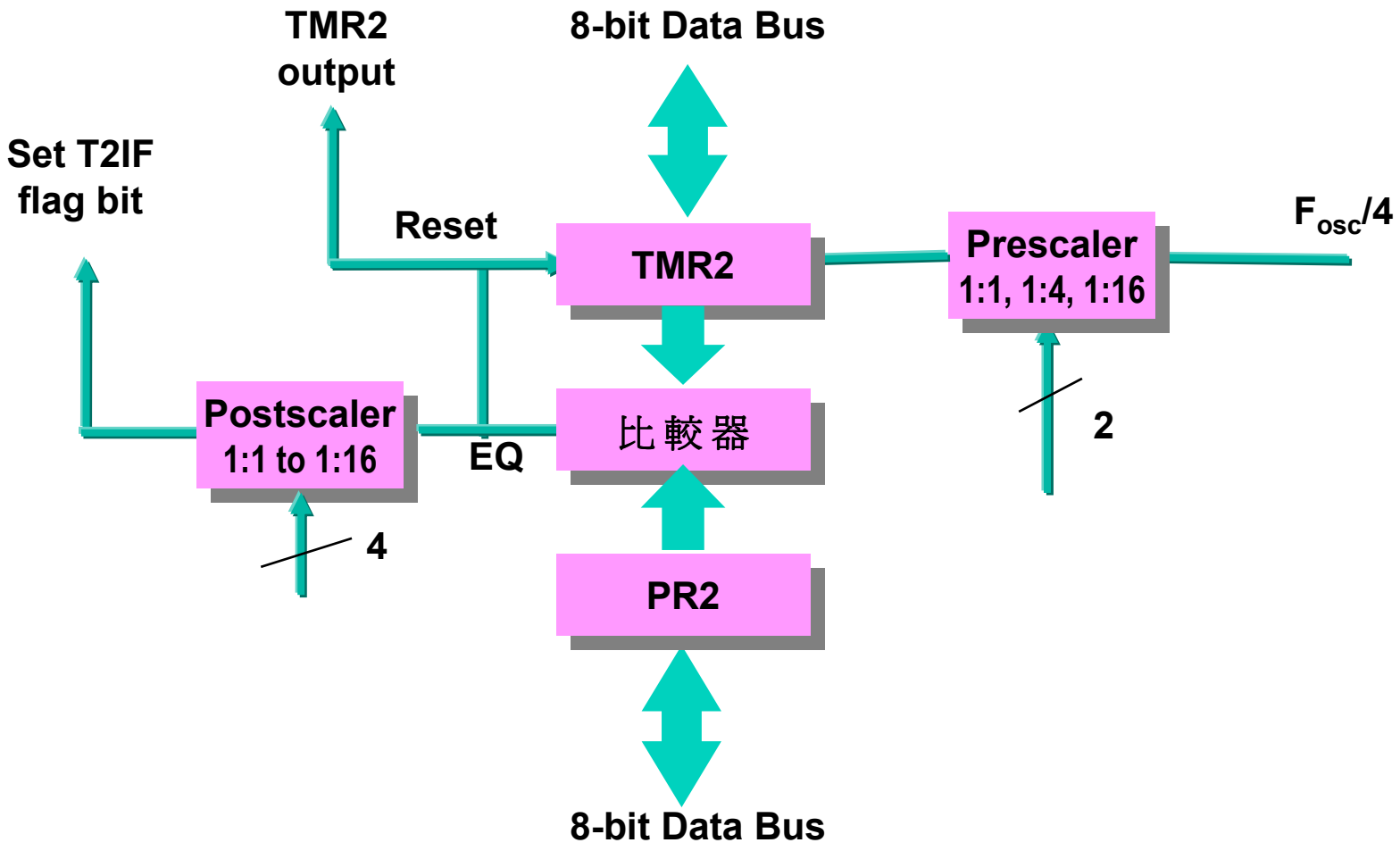
MICROCHIP
Regional Training Centers

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** 爲一可讀、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)** 清除爲零，重新計時
- **PWM** 模式下，是 **Period & Duty** 的基本頻率來源
- 也可以當成 **MSSP (SPI™)** 傳送速率的設定

MICROCHIP
Regional Training Centers

Timer 2 架構圖



利用 VR 控制 PWM 輸出 (控制 Duty Cycle)

```
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
        CCPR1L=ADRESH;               // Set A/D result to Duty Reg.
   }                                 // with 8-bit Resolutions
}
```

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 OpenPWM2(char period)

- OpenPWM1(0xFF); // 0xFF 爲 PR2 暫存器的設定值

void SetDCPWM2 (unsigned int dutycycle)

- SetDCPWM1(1023);

MICROCHIP
Regional Training Centers

練習 3-2
用ADC控制 LED 的亮度

- 開啓 “\RTC\W401\Ans3-2\Ex3-2.mcp”
- APP001 實驗版的 JPCCP2 要接給 RC1
- 利用 A/D、PWM 及Timer2 函數庫來撰寫
- 設定使用 PWM2 做爲輸出 (CCP2/RC1 接腳)
- 設定 PWM 的 PR2 爲 0xFF (Period 爲最大值)
- 調整 VR1 以改變 AN0 的輸入電壓，並將其 A/D 轉換結果做爲 PWM 的 Duty，直接調整 LED (D9) 的亮度
- 將 A/D 最高的八位元顯示在 PORTD 的 LED

MICROCHIP
Regional Training Centers

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

第四章

MPLAB C18 使用詳述

1.資料型別　2.變數形別

3.運算元　4.程式控制

4.模組化設計

5.指標與陣列

6.結構的資料形態

MICROCHIP
Regional Training Centers

C18 資料型別

MICROCHIP
Regional Training Centers

MPLAB C18 的資料型別

- **MPLAB C18 支援的整數資料型別**

Type	Size	Minimum	Maximum
char(1,2)	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	$2^{-126} \approx 1.17549435e-38$	$2^{128} * (2-2^{-15}) \approx 6.80564693e+38$

MICROCHIP
Regional Training Centers

資料型別 (signed & unsigned)

- **unsigned**
  - 通稱爲“無號數”，即正整數。
- **signed**
  - 通稱爲“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)**
  - 應用的場合爲一 16-bit 的整數 (正、負數)
- **short long**
  - 應用的場合爲一 24-bit 的整數 (正、負數)
  - 非常適合使用在“*far rom* ”長指標運算
- **long**
  - 應用的場合爲一 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 位址	0x100	0x101	0x102	0x103
內容值	0xDD	0xCC	0xBB	0xAA

記憶體模式
(near , far & rom , ram)

	程式記憶區(rom)	資料記憶區(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_MSG[ ]="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;
```

MICROCHIP
Regional Training Centers

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?”

MICROCHIP
Regional Training Centers

C18 變數類別

MICROCHIP
Regional Training Centers

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，變數可視爲 **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;
```

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--;
}
```

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，實際佔有記憶體的變數。
- 只要是視野內的函數均可存取此變數。
- 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。

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

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

/*** AddOne Function ***/
void AddOne(void)

{
  static unsigned char count=0;
  count++;
}
```

MICROCHIP
Regional Training Centers

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 **PIC18F4520** 的記憶體均可視爲一暫存器使用，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 ...

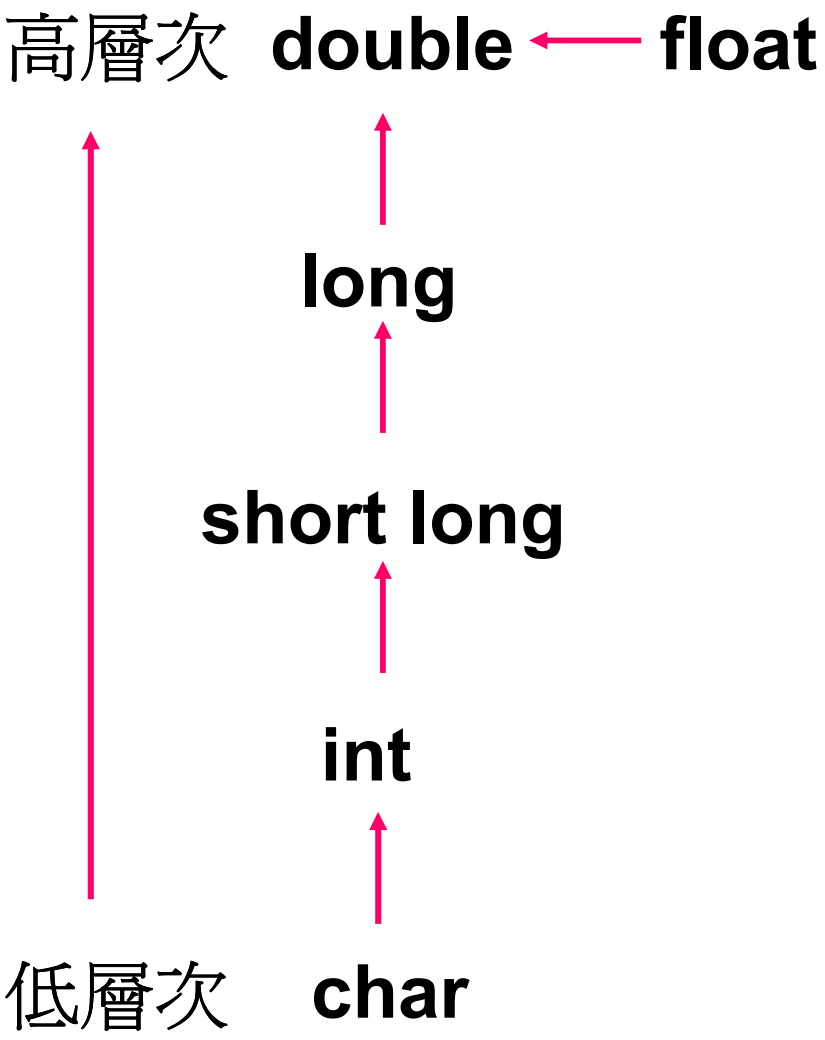
```
Unsigned char x,y,;
volatile unsigned char TMR0;

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** 用來創造自己的型別名稱。
- 主要的目的：
  - 提高程式的攜帶性
  - 讓程式更容易閱讀
  - 減少原始程式碼

```
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，兩個爲 1，結果爲真(1)
||	a || b	a 與 b 作邏輯OR運算，其中只要有1，結果爲真(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，均是以真或假爲判斷條件，
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	a=b
+=	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
|=	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

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 */

}
```

MICROCHIP
Regional Training Centers

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)

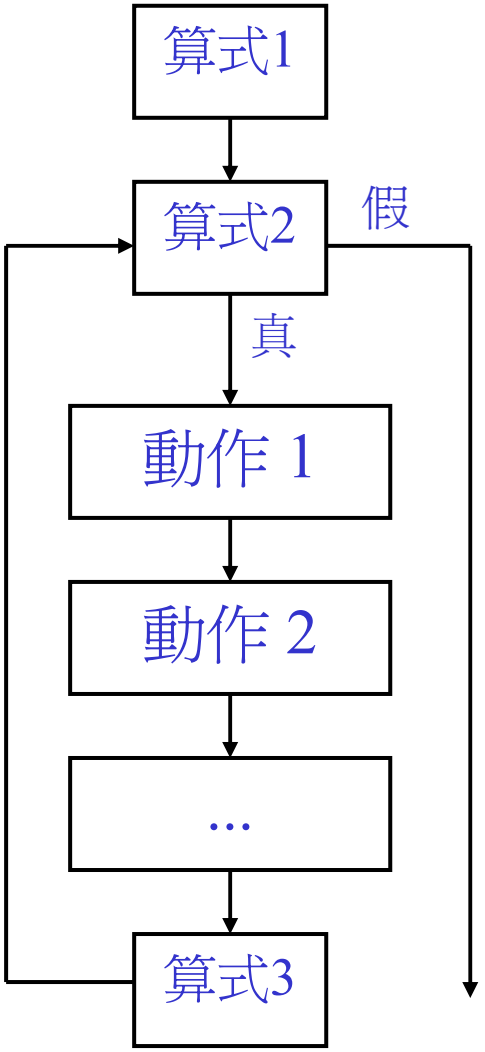
迴圈控制(Looping)

- 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;
      :
      :
      :
}
```



MICROCHIP
Regional Training Centers

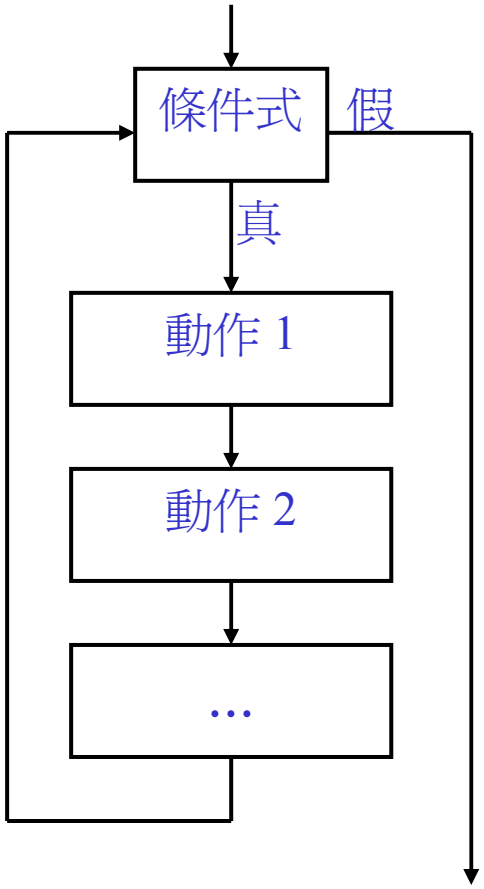
for 迴圈 - 範例

```
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;
            :
            :
            :
}
```



MICROCHIP
Regional Training Centers

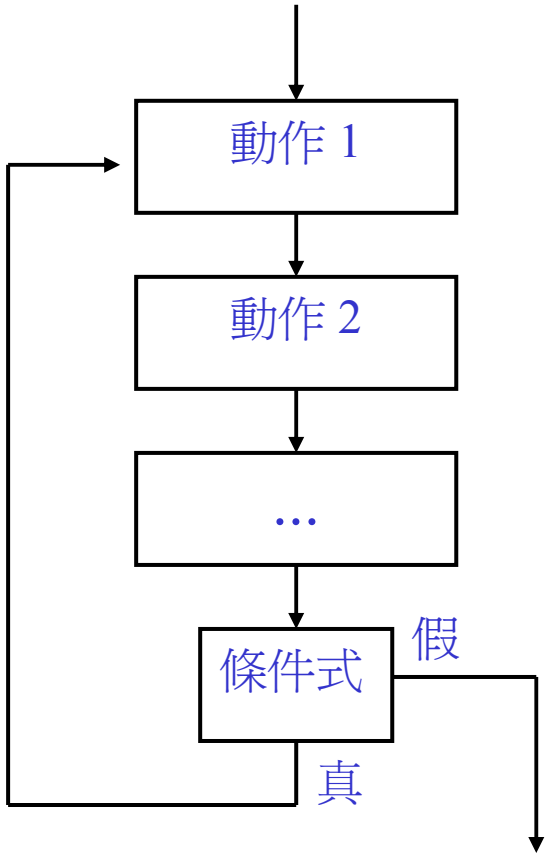
While 迴圈 - 範例

```
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) ;
```



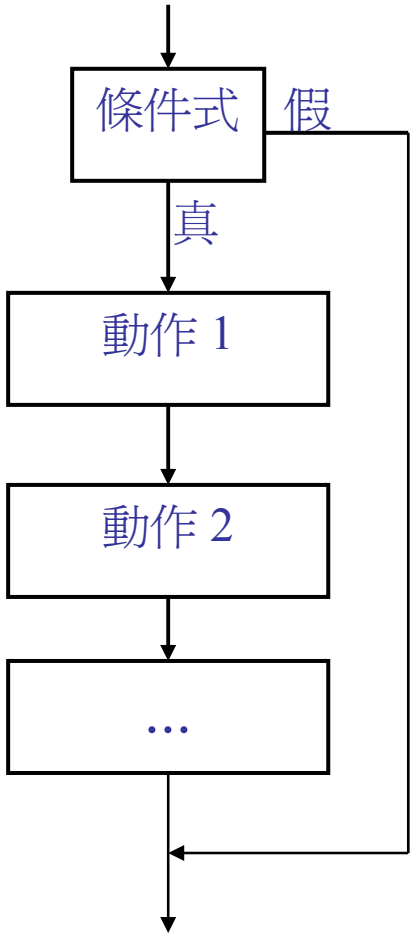
do While 迴圈 - 範例

```
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);
         }
```

if 敘述

- 格式:

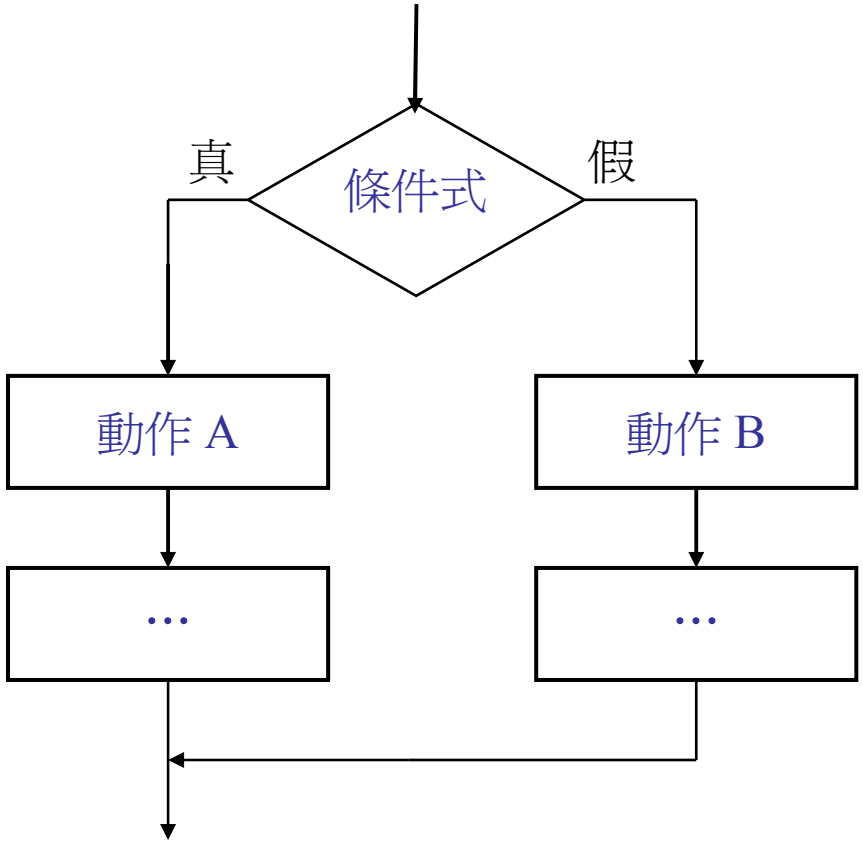
```
if (條件式)
{
        動作 1;
        動作 2;
          :
          :
          :
}
```



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

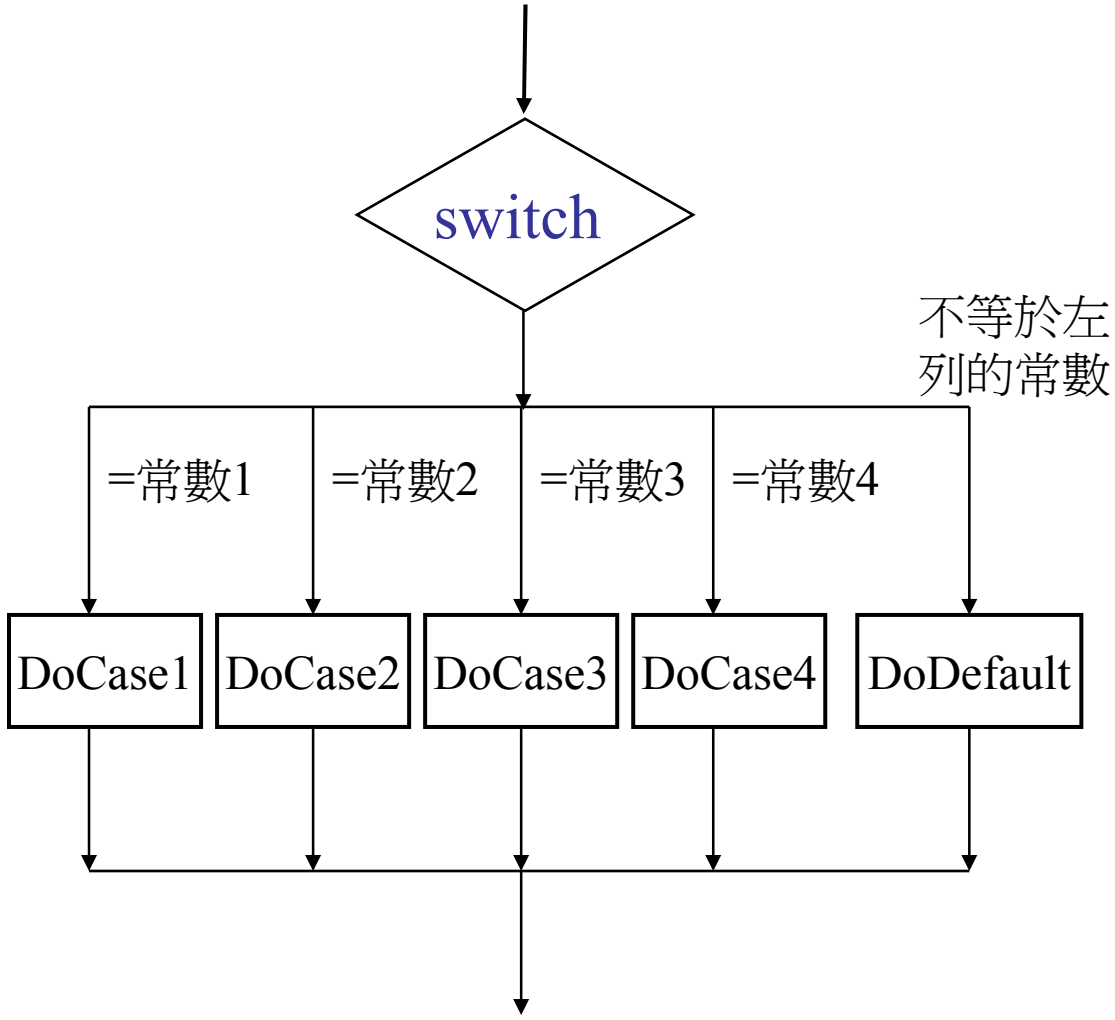
- 格式:

```
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，也不可以重覆

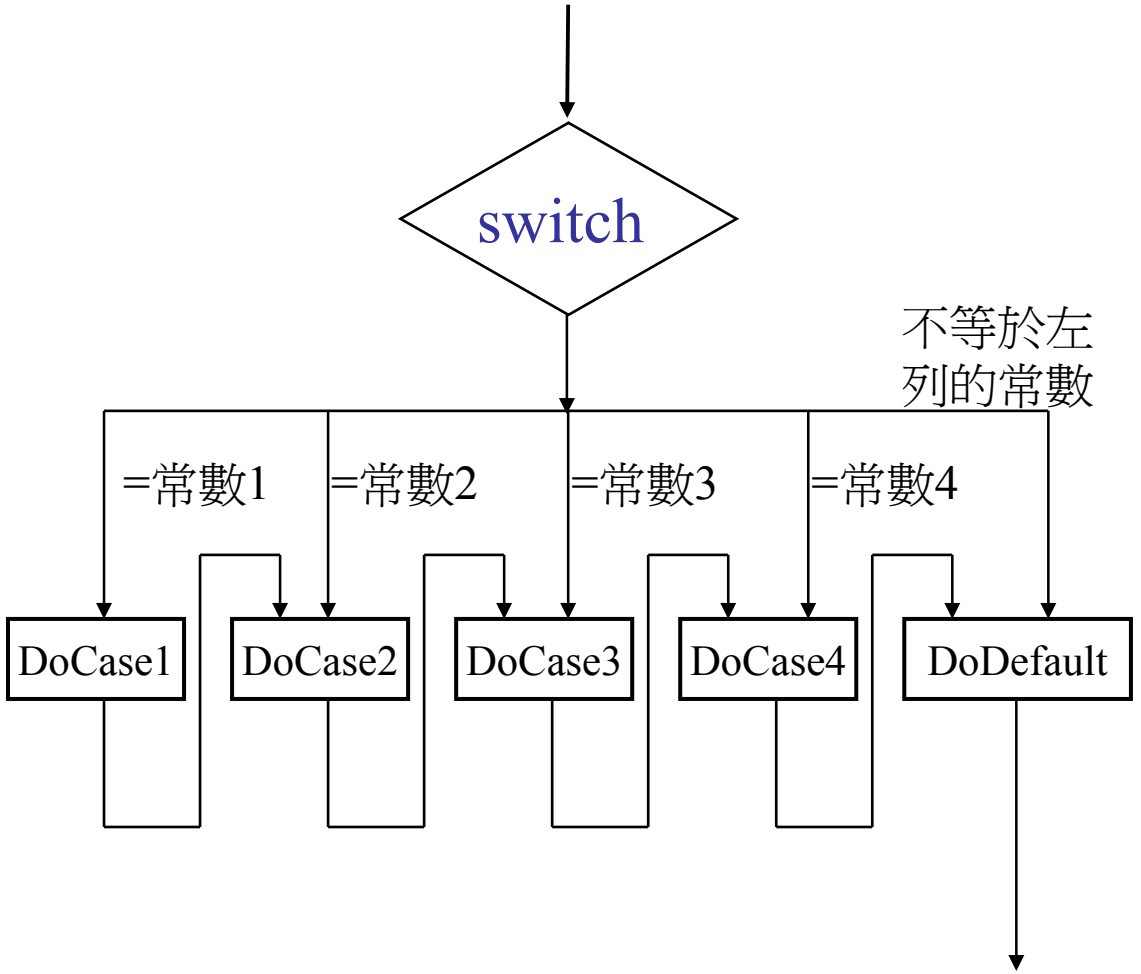
Switch + break 敘述 - 範例

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



Switch 敘述 - 範例

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

    case 2:
        DoCase2();

    case 3:
        DoCase3();

    case 4:
        DoCase3();

    default:
        DoDefault();

    }
```



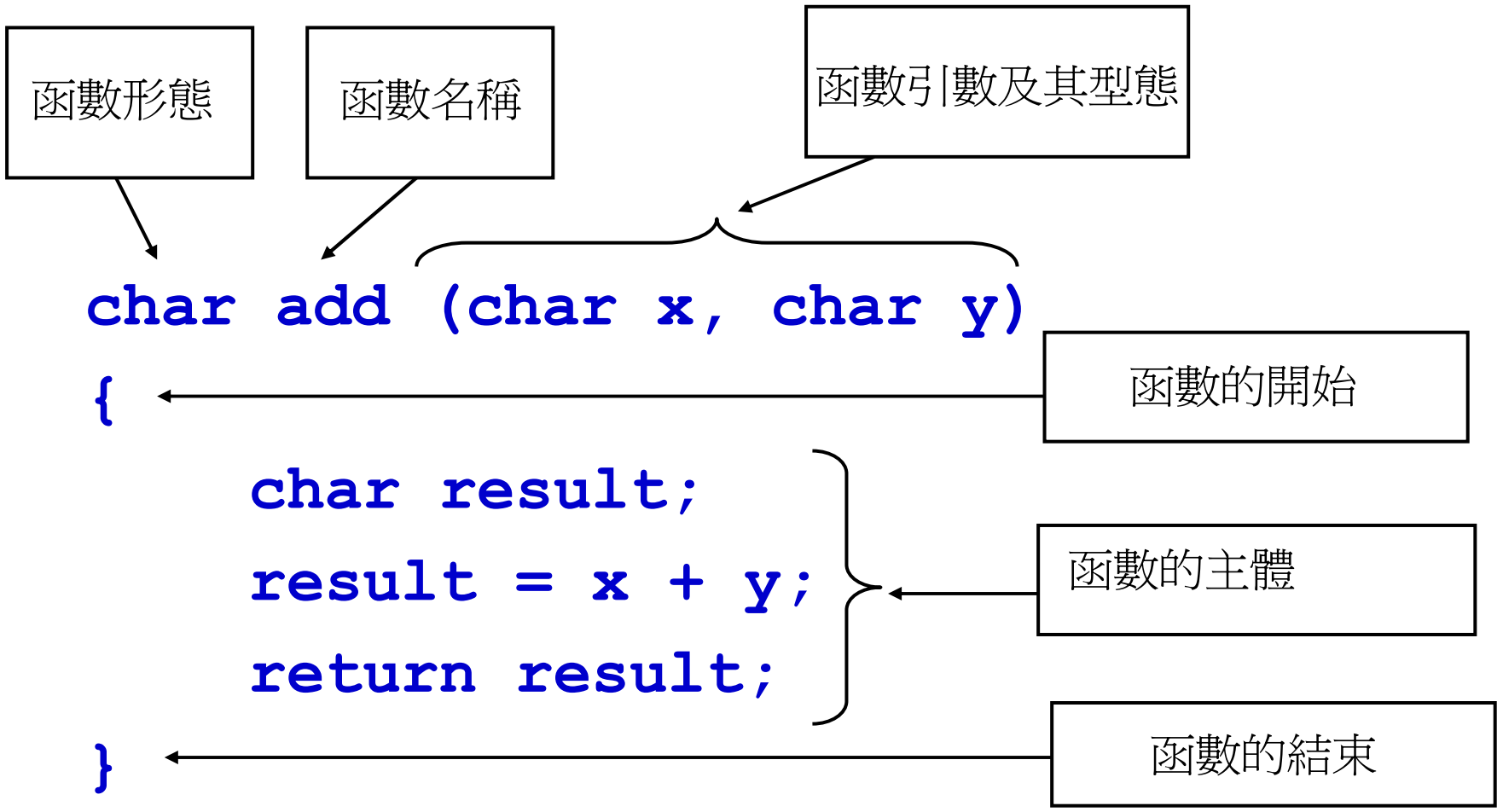
MICROCHIP
Regional Training Centers

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


函數形態
函數名稱
函數引數及其型態
char add (char x, char y)
{
char result;
result = x + y;
return result;
}
函數的開始
函數的主體
函數的結束

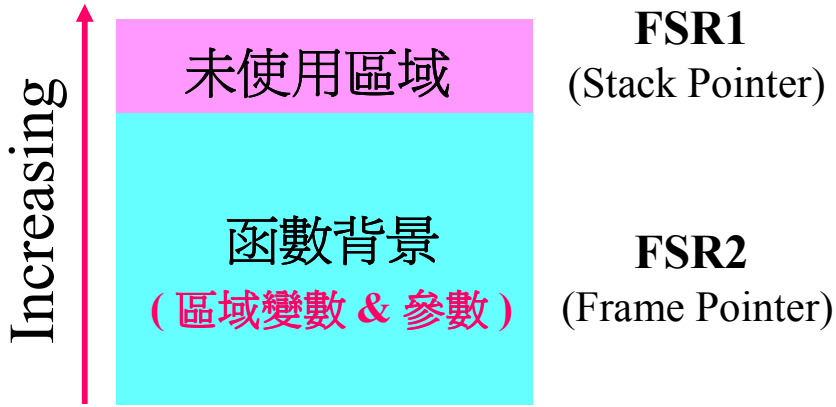
MICROCHIP
Regional Training Centers

記憶體管理
指標(FSR)

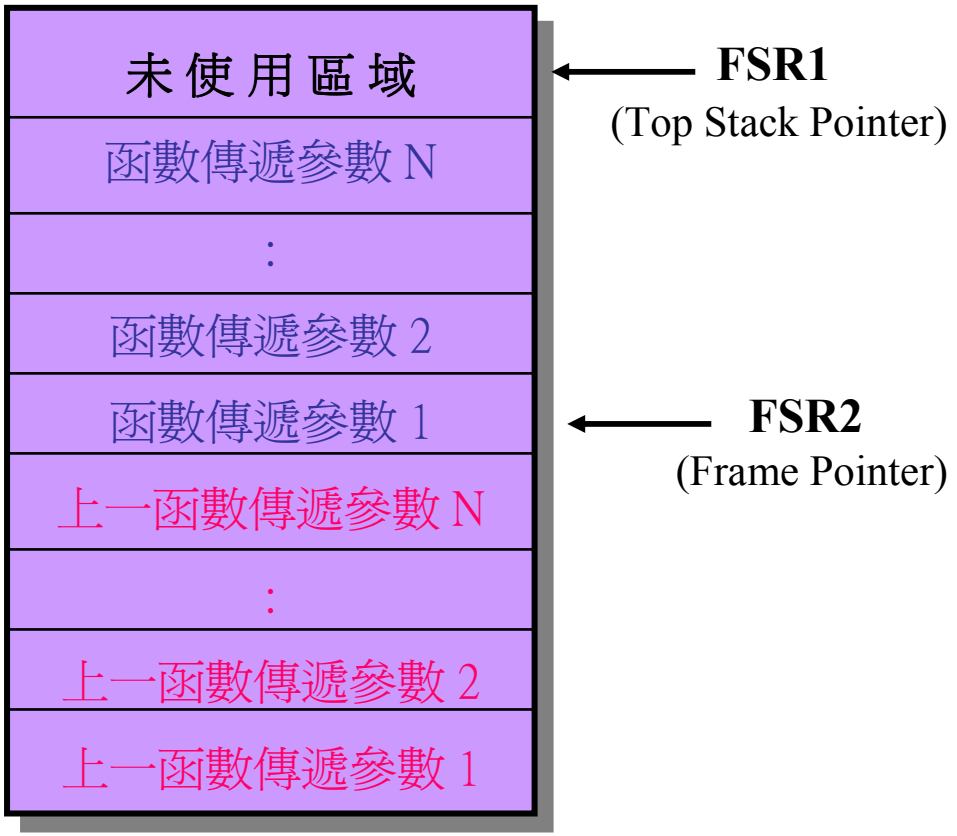
- 資料指標(Pointers to RAM)
  - FSR0 (只有較低的12 bits 有效)
  - 使用在一般通用型指標(例:陣列、字串…)
- 堆疊指標(Stack pointer)
  - FSR1
  - 指標所指的位置是下一個有效的堆疊位置
  - 最好不要動它
- 框架指標(Frame pointer)
  - FSR2
  - 指標所指的位置是上一層參數傳遞最後位置
  - 使用前其值必須存到軟體堆疊

記憶體管理
函數呼叫的軟體堆疊

基本軟體堆疊



軟體堆疊函數呼叫時



MICROCHIP
Regional Training Centers

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

Call-by-Value

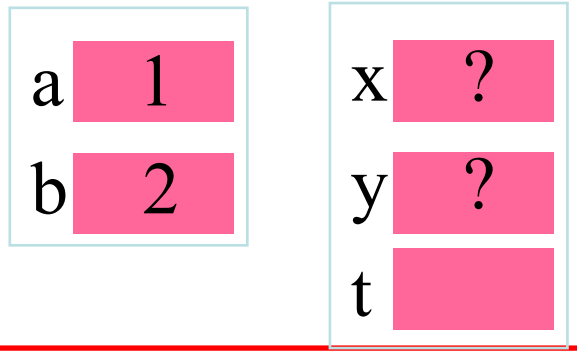
呼叫者

```
void fun(void)
{
   int a,b;
   a=1;
   b=2;
   swap(a,b);
}
```

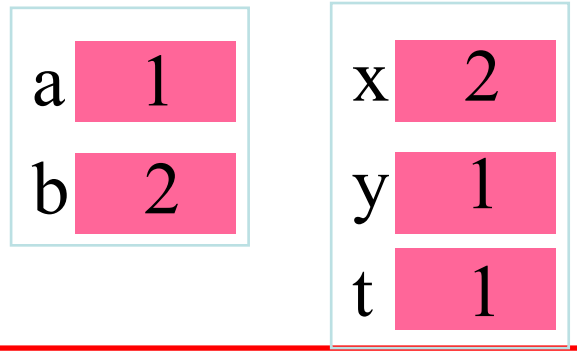
被呼叫者

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

Fun 呼叫swap前



Fun 呼叫swap後



MICROCHIP
Regional Training Centers

何謂 Call-by-Reference

- 以變數的位址作爲引數的傳遞
- 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爲一指標)
- 被叫用的函數要將引數宣告爲 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

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]);
}
```

被呼叫者

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

執行前

i=1
a 陣列內容 = 1, 2, 3

執行後

i=2
a 陣列內容 = 1, 1, 3

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
        }
}
```

MICROCHIP
Regional Training Centers

函數回傳值 (Returning Values from Functions)

- 函數可以利用 **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數外宣告
  - 函數必須冠上傳回值的資料型態

MICROCHIP
Regional Training Centers

函數回傳值 (範例)

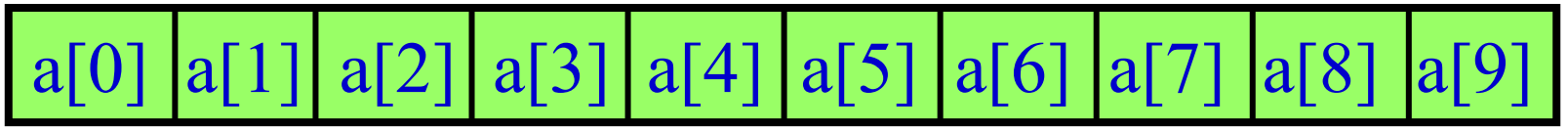
```
Void main (void)
{
   unsigned char c=0;

   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)

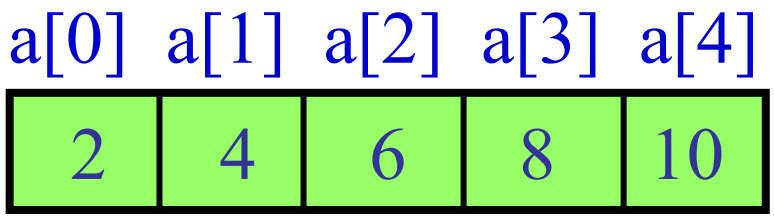
陣列 (Array)

- 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爲該陣列的元素個數，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]

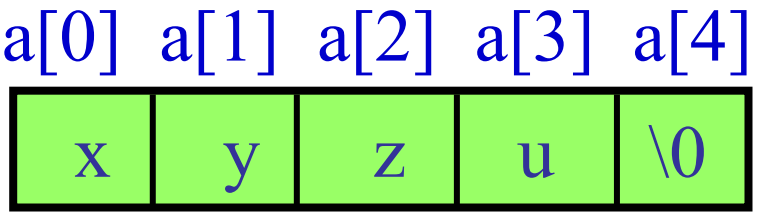


陣列初值的設定

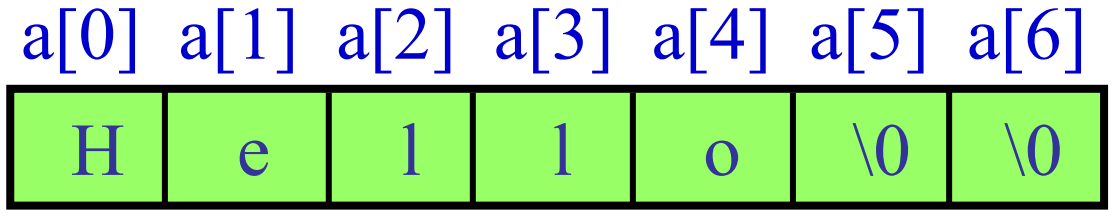
- **Static char a[5] = {2,4,6,8,10}; // 常數設定**



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



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



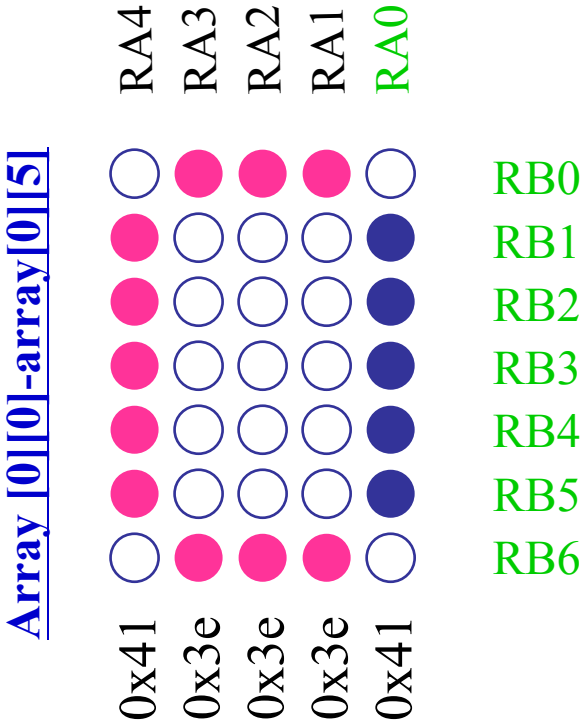
陣列 和 字串

- 字串是多個字元的組合，最後是以**0x00 (通稱null character \0)**爲結束字元
- 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);
        }
   }
}
```



MICROCHIP
Regional Training Centers

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”**內的資料最多爲 **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\18f452.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。

MICROCHIP
Regional Training Centers

記憶體管理
資料陣列的擴展 - 範例

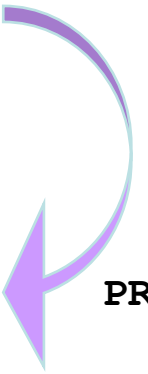
Large arrays - C source

```
#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++;
   }
}
```

記憶體管理
資料陣列的擴展 - 範例

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 #pragme udata 指定

MICROCHIP
Regional Training Centers

指標 (Pointers)

- 如果有一個變數內存放的是“變數A”的位址
  - 則這個變數就稱之爲“變數A”的指標
  - 這個變數稱爲指標變數
  - 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，就是宣告爲指標型態
  - 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

- 假設指標爲: `char *Ptr1;`
  - `Ptr1=0x20;` 將指標位址設爲 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(目前位址爲 0x20)
  - `Ptr1 += 5;` 將指標位址加 5,成爲 0x25
  - `*Ptr1 += 5;` 將常數5與指標所指的位址內容相加

- 指標運算
  - 指標本身可以被增加或遞減
  - 指標可以用計算的方式加、減一個值(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 爲指標變數
char Array1[] = "Hello";          // 宣告陣列 Array1[]
char *APtr;                       // 宣告 APtr 爲指標變數

VPtr = &Var;                      // 將 Var 變數的位址傳給 VPtr
APtr = Array1;                    // 將陣列 Arrar1 的起始位址傳給 APtr
Aptr = &Array1[0];                // 作用與上一行相同
                                  // 注意 : Aptr爲16-bit的指標變數
```

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
}
```

綜合問題 ？

請說明程式中

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

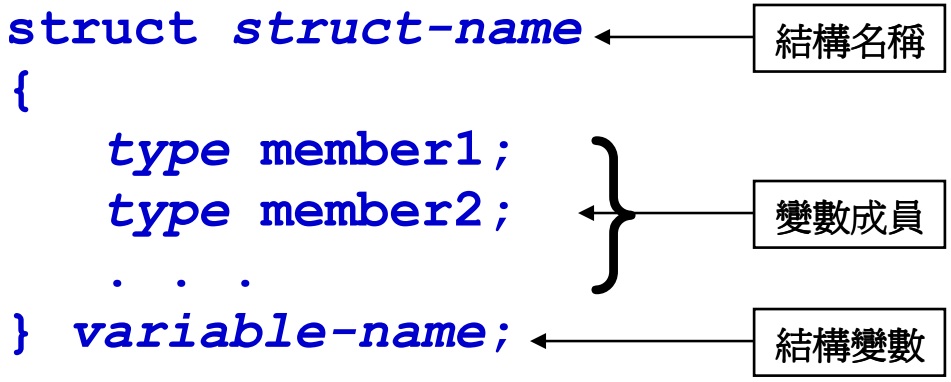
三種函數的執行結果有何不同？

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



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，最大值爲一個位元組(byte)
- 成員爲一個個單獨的位元(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爲一位元結構的共用
    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 {
        unsigned INT0:1;             // 定義PORTB的另外功能
        unsigned INT1:1;
        unsigned INT2:1;
        unsigned CCP2:1;
    } ;
} PORTBbits ;
```

摘錄自“P18F4520.h”檔

使用位元結構 (範例)

```
PORTB=0x34;              /* 記著！ PORTB & PORTBbits
                            的位址定義是在 p18C4520.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;
```

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)內的變數稱爲共用元素
  - 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;
```

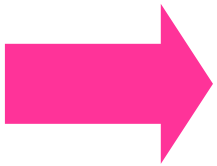
共用型態名稱

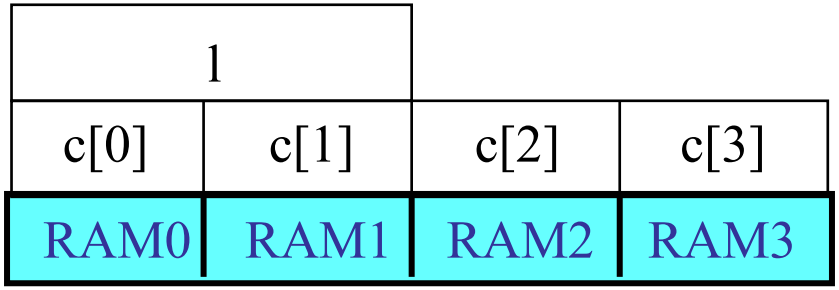
共用元素成員

共用型態變數

共用型態的資料架構

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





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

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 ;
```

MICROCHIP
Regional Training Centers

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)
}
```

MICROCHIP
Regional Training Centers

練習四的準備

如何使用 LCD 函數

- 使用**LCD**函數庫時，須同時使用定義檔 **“WAP_LCD.h”**和**“delays.h”**
- **WAP_LCD.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

MICROCHIP
Regional Training Centers

使用 LCD 函數

- **void OpenLCD : 開啓LCD**
  - OpenLCD ( ); (範例)
- **putsLCD & putrsLCD : 寫一字串到LCD**
  - void putsLCD (char *ptr) ---- from RAM
  - void putrsLCD (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_Crusor(1 , 0) ; 將Cursor設定在第二行第一個位置

練習 4

- 開啓 “\RTC\W401\Ans4\Ex4.mcp”
- 設定常數字串“ C18 Workshop Ex4“在ROM區
- 設定變數字串“ A/D Value- -> “在RAM區
- 利用LCD函數來撰寫，開啓 LCD 並設定爲 4-bit 模式、5x7 雙行模式、CURSOR OFF
- 在 LCD 的第一行顯示“ C18 Workshop Ex4 “
- 在 LCD 的第二行顯示“ A/D Value- ->“
- 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 Value→1023

MICROCHIP

Regional Training Centers

第一天課程結束

謝謝!