
PLANET MICROCHIP
MPLAB-IDE
Workshop 100

歡迎參加 W100 教育訓練

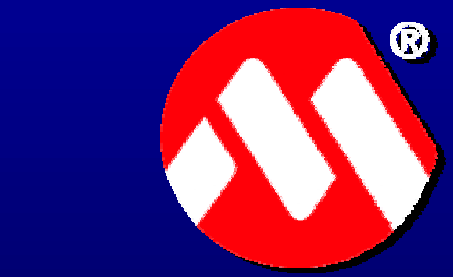
- 使用軟體工具
  - MPLAB IDE v6.10.00 (或更新版本)
  - MPASM , MPLINK , MPLIB
- 使用硬體工具
  - MPLAB ICD2
  - Microchip APP001 Workshop Board (PIC16F877A inside)
- 參考書籍
  - Microchip PIC16F87xA Data Sheet (DS39528A)
  - MPASM User's Guide with MPLINK and MPLIB (DS33014)
  - MPLAB IDE v6.10 中文使用手冊
  - APP001 Workshop Board 電路圖

Workshop 100 課程目標

- 認識 Microchip 的 PICmicro
- 使用 Microchip 提供的開發工具
- 組合語言的正確撰寫格式
- 基礎的程式寫作
  - I/O 的使用
  - 延時副程式 (Delay Subroutine)
  - 類比 / 數位 轉換器 (ADC) 的使用
- MPLAB-ICD2 及多功能實驗板的操作

Workshop 100 的內容

1. PIC Mid-Range (14-bit 指令) 架構
2. 開發工具介紹
3. 使用 MPLAB IDE v6.xx
4. PIC16F877A 基本指令介紹
5. 使用 MPLAB ICD2 除錯
6. 基本 I/O 控制
7. 了解旗號的意義與用法
8. 速度可調整的霹靂燈設計



PIC Mid-Range (14-bit 指令) 架構

嵌入式控制器

- **Embedded Controller ：**
  - 整合產品所需的各項功能於單一晶片中
- 一般的嵌入式控制器包括以下部份
  - **CPU core**
  - **Program Memory (ROM / OTP ROM / MASK ROM / FLASH)**
  - **Data Memory (RAM)**
  - **Data EEPROM**
  - **I/O**
  - **各種周邊， 如 ADC， PWM， TIMER， I^2C， USART … 等**
  - **看門狗 (Watch Dog)， 電源異常偵測 (Brown Out Detect)， 低電壓偵測 (Low Voltage Detect)， 及內部重置 (Internal RESET)**
  - **LCD Driver**

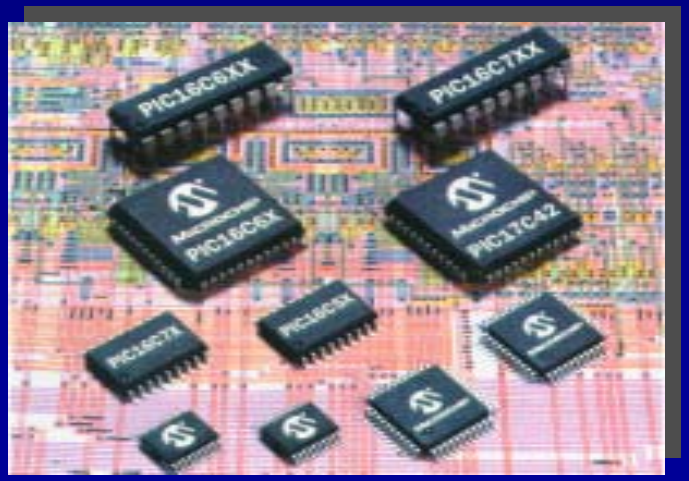
使用 MCU 的優缺點

- 優點：
  - 減低產品開發成本，提昇功能
  - 增加設計彈性，使產品有改版及升級的空間
  - 維修成本低廉
  - 產品的一致性高
  - 小型化容易
  - 易於用來實現複雜的數學及邏輯算式
  - ……. More & More

- 缺點：
  - 反應速度通常不及直接組合而成的專用硬體電路
  - 雜訊免疫能力的問題
  - 價格較 ASIC 的元件高

PICmicro® MCU 架構

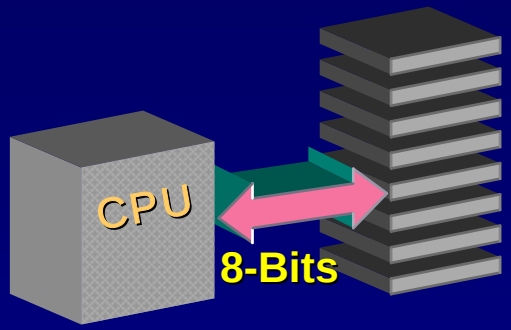
- 俱備 **RISC Microcontroller** 的特點
- 高效能的 **PICmicro** 具備 **RISC Microcontroller** 的特點 , 其主要的優點如下 :
  - 使用 **Harvard** 管線架構
  - 大部分指令均能在單一週期值內執行完成
  - 支援指令預提取功能
  - 所有指令為 **“Single Word”**
  - **Long Word** 的指令編碼
  - 精簡指令集,重複指令極少
    不易造成混淆



PICmicro 的架構比較

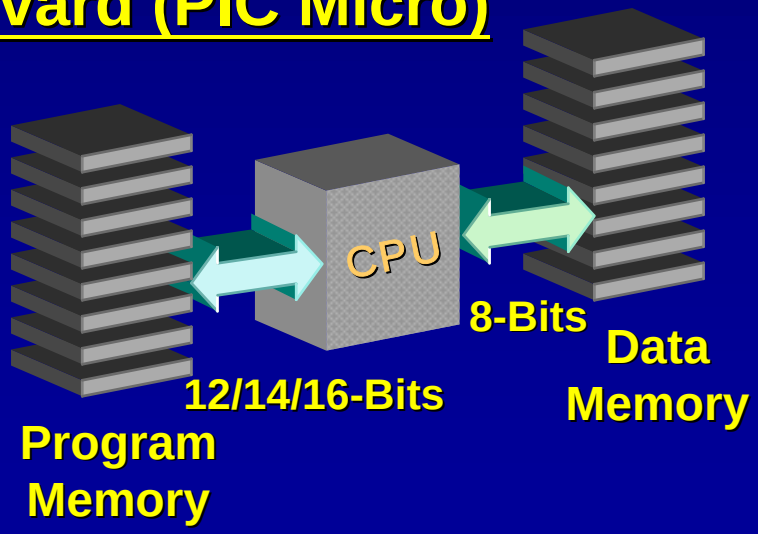
PIC 使用 Harvard Architecture

Von Neumann (一般MCU)



- 經由相同的記憶體來提取指令與存取資料
  - 指令與資料無法有效率的同時被處理
  - MCU 的操作效率受到此結構影響而變差

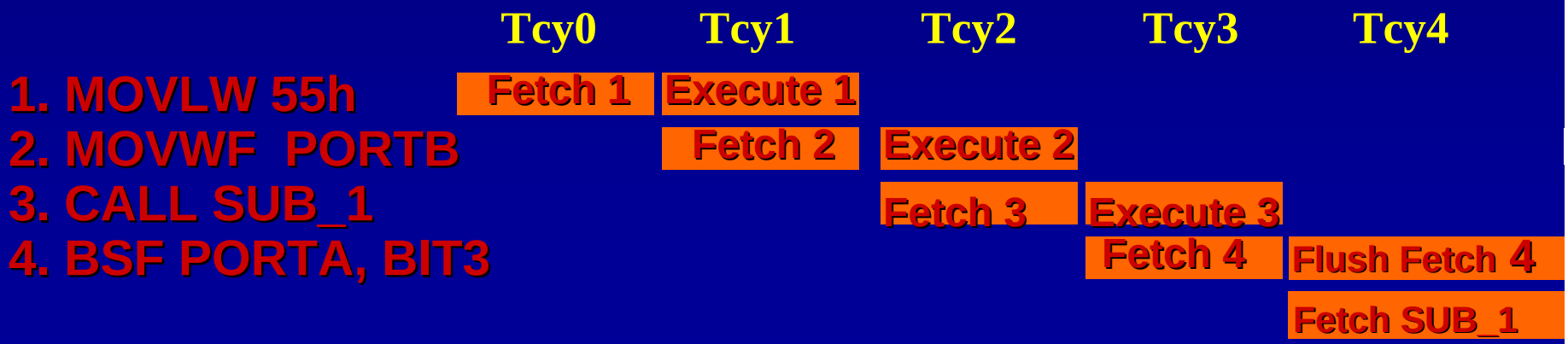
Harvard (PIC Micro)



- 使用兩個不同的空間與管線來存取程式和資料
  - 增加處理資料的效能
  - 使得MCU可以具有不同寬度的程式記憶體與資料記憶體 (8 Bit 寬的 Data Memory , 12/14/16 Bit 寬的 Program Memory)

PICmicro MCU Architecture
Pipelining

- 對大部份的MCU而言, 指令的提取與執行是連續發生但其每一個指令周期只有一個動作發生
- PICmicro 使用 Harvard 結構且使用流水管(Pipeline)的運作模式
- Pipeline 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
- 程式分支的有關指令 (例如: GOTO, CALL 或 Write to PC) 則需要兩個指令周期



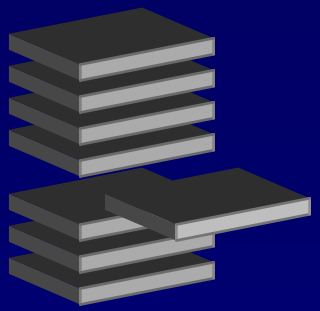
PICmicro Architecture

較長的指令寬度（Long Word Instruction）

- 將指令與資料匯流排分開的方式， 使得PICmicro可以依照需求來調整所需的指令寬度
- PICmicro 指令寬度為 12， 14 or 16-bits，稱為一個 Instruction Word，每一指令只佔一個 Word (16Fxxx)
- PICmicro 的資料匯流排(Data Bus)寬度是 8 bits
- 若於 PIC16CXX 具備 2K x 14 words 的程式記憶體可完成的工作約相當於其他有4K 記憶體的MCU
- 單一周期的指令運作使MCU的處理能力提高許多

PICmicro Architecture

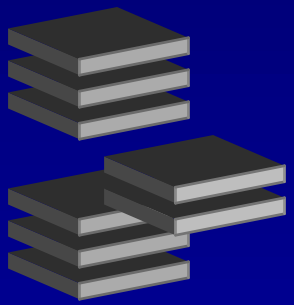
Long Word Instruction (con't)



PICmicro

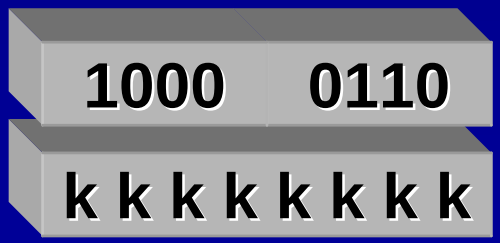
```
movlw     #imm<8>
```

只須一個Word即可完成指令編碼



MC68HC05

```
ldaa      #imm<8>
```

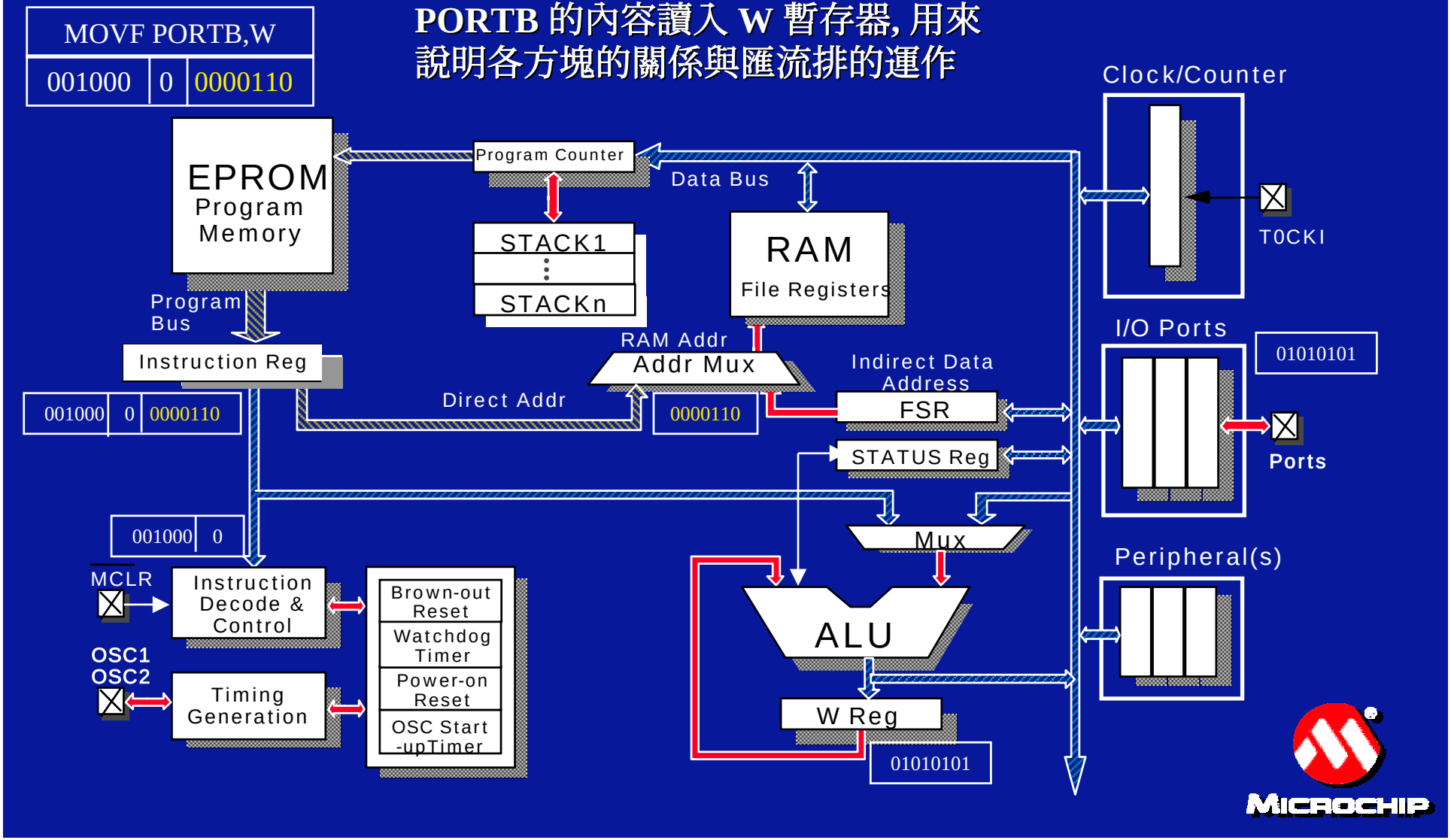


需要兩個Bytes來進行指令編碼

PICmicro Architecture

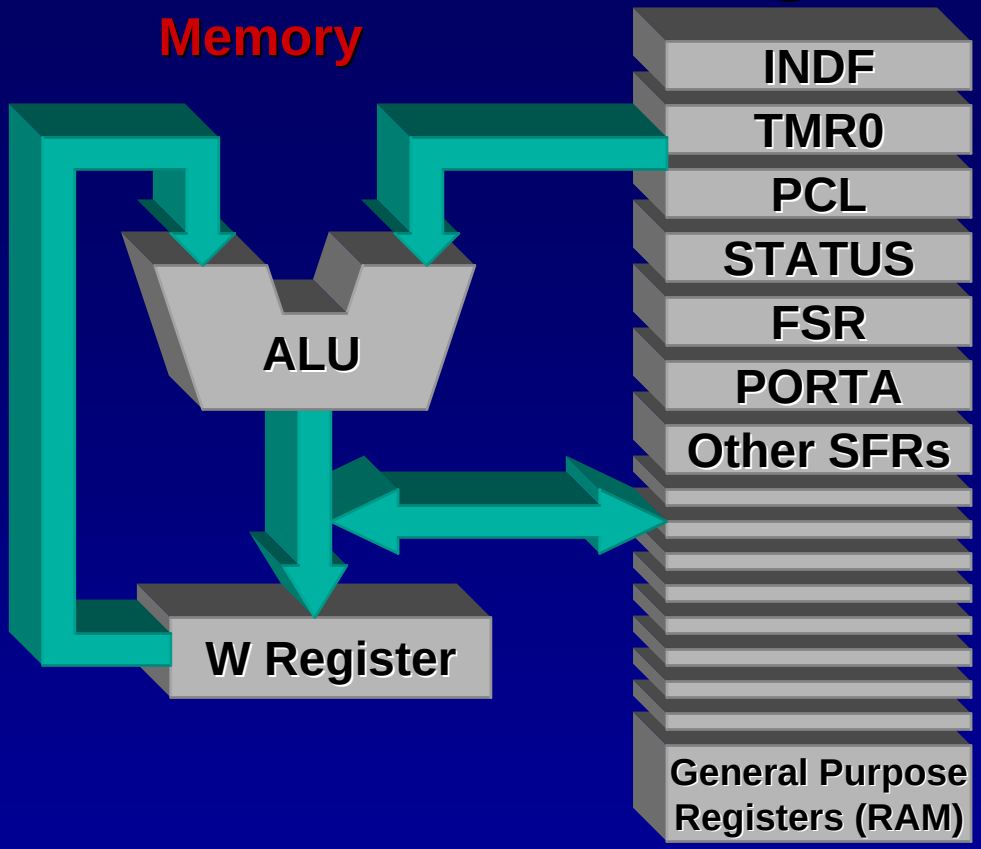
PIC16CXX 內部方塊圖

以 MOVF PORTB,W 指令, 將 PORTB 的內容讀入 W 暫存器, 用來說明各方塊的關係與匯流排的運作



PICmicro Architecture

應用 Register File 的概念



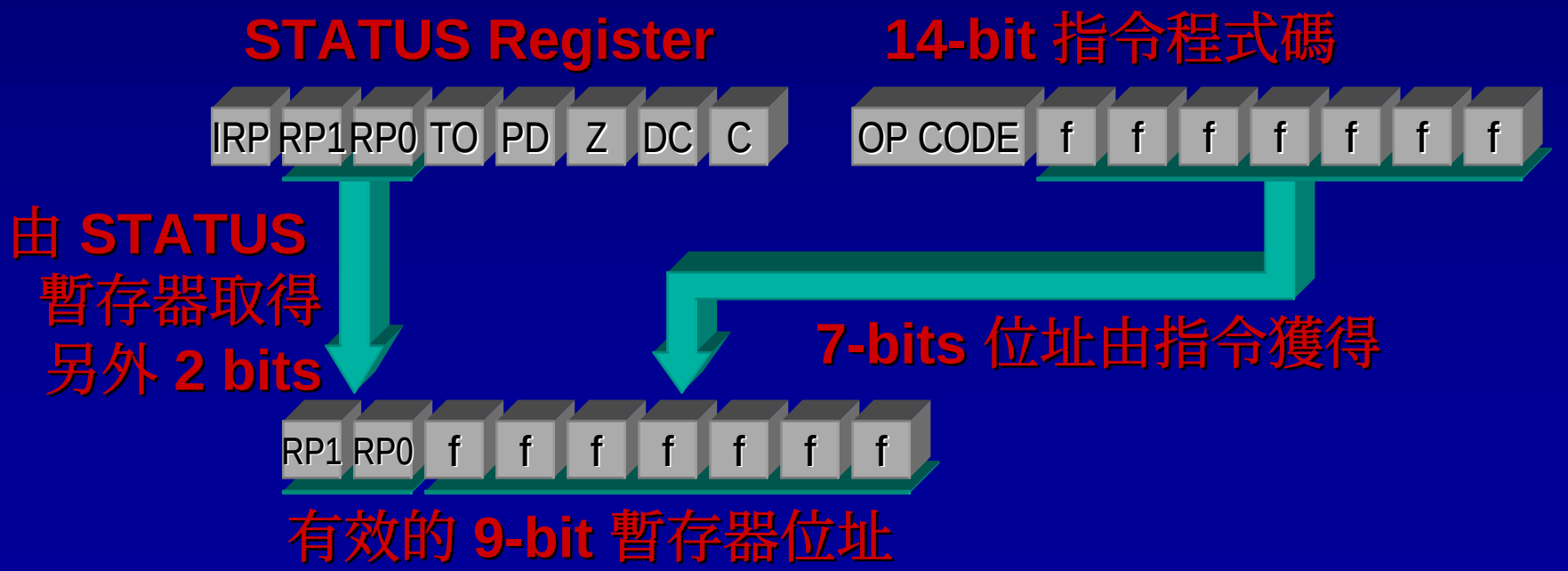
- 暫存器庫(Register Bank)由許多一般暫存器與特殊用途暫存器 (SFR) 組成
- 所有周邊(如：I/O)的操作與暫存器相同
- 任何一個暫存器都可被用於存取Register File的指令所操作
- Long word 的指令編碼方式使得指令與暫存器的位址僅以一個word即可表示

14-bit 指令格式的範例:資料存取

PICmicro Architecture

資料記憶體：直接定址

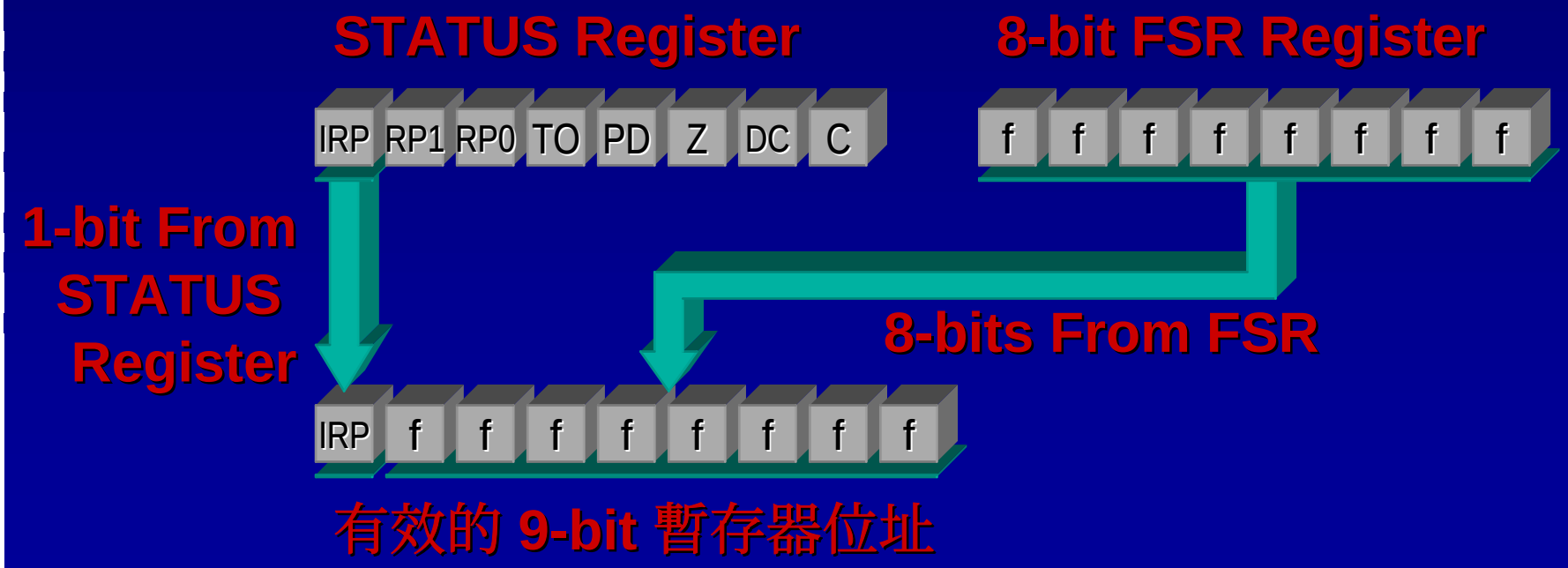
- Mid-Range (14-bit 指令PIC) 的每一個暫存器庫為 128 Bytes
- PIC16CXX 的最大DATA RAM大小為 4 個 BANK (512 Bytes)
- 資料記憶體的有效位址為 9 bits
- 7-bit 的位址直接來自於指令中
- 2-bit 由 STATUS 暫存器獲得, 以定址完整的 4 個 BANK



PICmicro Architecture

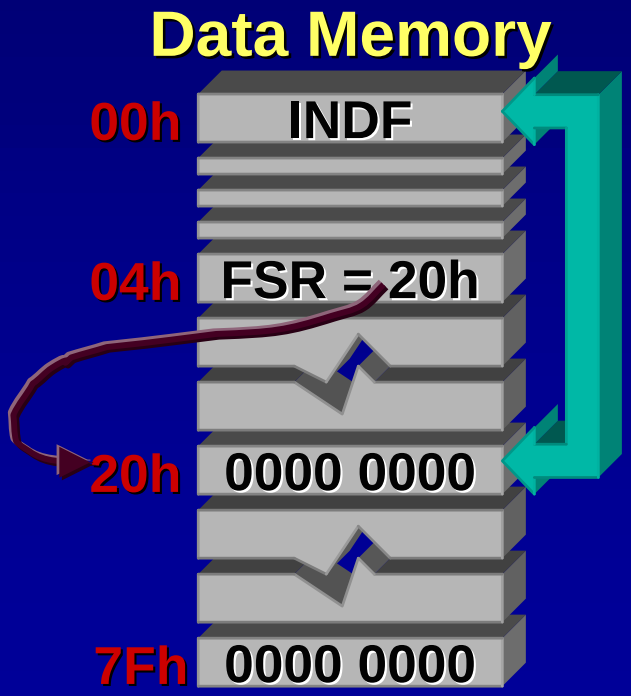
資料記憶體：間接定址

- Mid-Range (14-bit PIC) 的資料記憶體有效位址為 9 bits, 最大定址範圍是 512 Bytes (4 BANKs)
- 8-bit 的位址由 FSR (File Select Register) 獲得
- FSR 可提供 256 Bytes 的定址能力 (2 BANKs)
- 有效位址的第 9 位元由 STATUS 的 IRP bit 控制



PICmicro Architecture
資料記憶體: 間接定址的使用範例

- 清除由位址 0x20 to 0x7F 的資料暫存器
  - 所需間接位址被寫入 FSR 中
  - 當 INDF 被當成運算元(Operand)時, 被 FSR 暫存器內容所指到的位址才是真正操作對像




```
        movlw   0x20
        movwf   FSR
LOOP    clrf    INDF
        incf    FSR,F
        btfss   FSR,7
        goto    LOOP
        <next instruction>
```

PICmicro Architecture
程式記憶體：立即值定址

- 將 8-bit 的常數(Literal) 直接置於指令的 bit-0 至 bit-7
- 配合不同的OP Code 將常數做為運算元
- 可直接將常數與 W 暫存器運算
- 使用於常數操作指令, 如 movlw, addlw, retlw,, 等

14-bit Instruction for Literal Instructions

PICmicro Architecture

程式記憶體: 程式計數器 (PC) 的絕對定址

- 使用於 CALL 及 GOTO 等控制指令
- 藉由改變 PC (Program Counter) 來更改程式的執行位址
- CALL 和 GOTO 指令直接將 11-bit 位址置於指　令中, 可於相同的 2K 程式頁直接操作
- 若範圍超過 2K 的程式頁範圍, 需更新 PCLATH 的內容
  - MOVLW HIGH TARGET_ADDR
  - MOVWF PCLATH
  - CALL TARGET_ADDR

14-bit Instruction for call and goto

PICmicro Architecture

程式記憶體: 程式計數器 (PC) 的絕對定址

- 虛指令 “PAGESEL” 可以幫助您完成因超過 2K 程式範圍所需對 PCLATH 的設定工作
- 若欲 Call 或 Goto 的位址在 2K 的 Page 範圍之外或不確定是否同一 Page, 可用 PAGESEL 來幫助程式的處理
- 例如：要前往的位址為 TARGET_ADDR
  - 使用下列的敘述

```
PAGESEL     TARGET_ADDR
CALL        TARGET_ADDR
```

  - 相當於

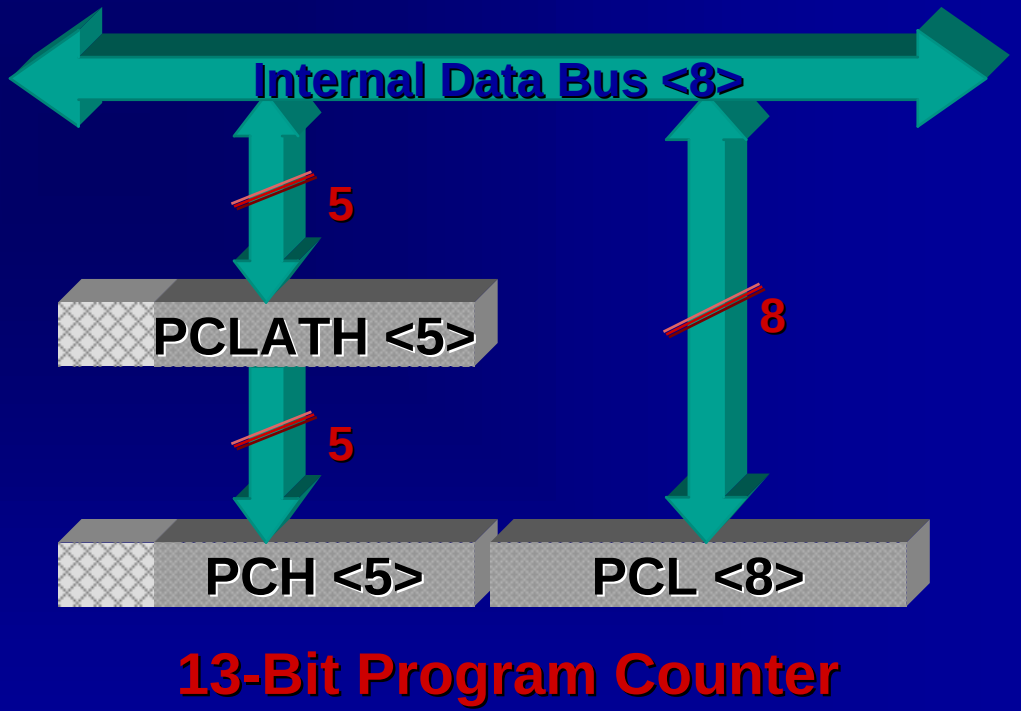
```
MOVLW       (HIGH)  TARGET_ADDR
MOVWF       PCLATH
CALL        TARGET_ADDR
```

PICmicro Architecture
程式記數器(PC) 的相對定址

- 以運算PC (程式計數器) 來達成相對定址的工作
- 應用於計算式 goto 的程式計巧
  - 將一個偏移值直接與 13-bit 的Program Counter 相加, 以獲得最終的跳躍位址
  - 可於整個 8K 的程式範圍中操作
  - 使用 PCLATH 來保持位址的 5 個高位元
  - 當 PCL 被當成運算的目的地時, 將PCLATH 以及運算結果整個一次寫入13-Bit 的 Program Counter

PICmicro Architecture
程式記數器(PC) 的相對定址

- 先將目標位址的 High Byte 寫入 PCLATH.
- 再將 low byte 寫入 PCL, 如此將會把整個13-Bit 的值寫入程式記數器 (PC)

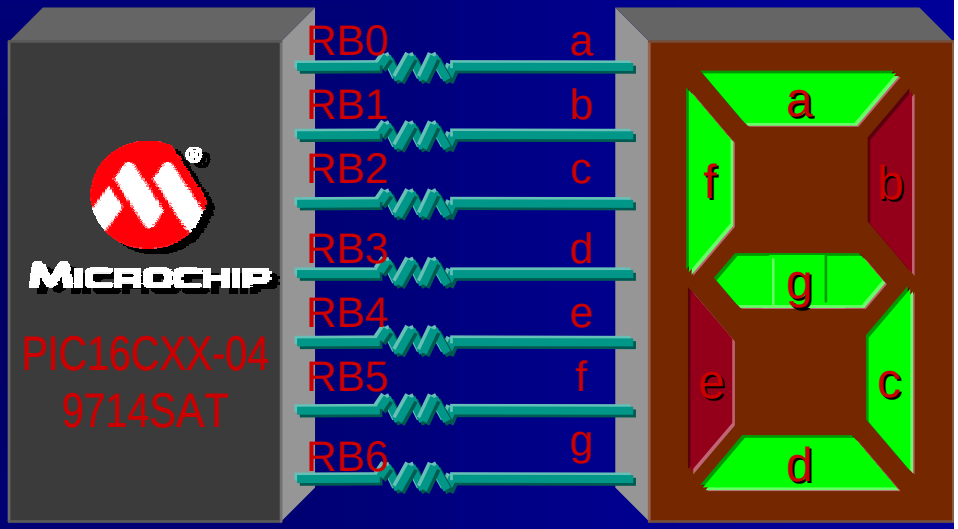


- 若要讀取PC的值
  - 可由讀取 PCL 來得到 PC 的 Low Byte
  - 讀取 PCLATH 的內容並不會得到當時的 PCH值(Bit 8 - 12 of PC)

PICmicro Architecture
相對定址 (Relative Addressing)

- 查表功能的實現範例

```
    org     0x10
    clrf    PCLATH
    movf    DisplayValue,W
    call    SevenSegmentDecode
    movwf   PORTB
    goto    Continue
SevenSegmentDecode
    addwf   PCL,F
    retlw   B'00111111' ;decode 0
    retlw   B'00000110' ;decode 1
    retlw   B'01011011' ;decode 2
    retlw   B'01001111' ;decode 3
    retlw   B'01100110' ;decode 4
    retlw   B'01101101' ;decode 5
    retlw   B'01111101' ;decode 6
    retlw   B'00000111' ;decode 7
    retlw   B'01111111' ;decode 8
    retlw   B'01101111' ;decode 9
Continue
```



01101101 W Register

01101101 I/O Port B

PICmicro MCU Architecture
Interrupt Overview

- 豐富的內部與外部中斷來源
  - 幾乎所有周邊皆有中斷 CPU 的能力
  - 有些 I/O 的變化也可構成中斷條件 (PORTB)
- 使用軟體來決定中斷被處理的優先順序
  - 可彈性的由軟體自行調整優先順序
- 有整體及各別的中斷致能控制位元
- 大部份的中斷可以用來喚醒處於SLEEP狀態的 PICmicro
- 中斷延遲固定為 3 個 cycle
  - 容易以軟體來判斷中斷的經過時間

PICmicro MCU Architecture

Interrupt Comparison

- **PIC16CXXX @ 12MHz**
  - **每一指令執行需時　1 or 2 cycle**
  - **指令周期　= clk / 4**

```
                                ; word/cycle
Int_Srv:                        ; 0 / 3
MOVWF     tempW                 ; 1 / 1
SWAPF     STATUS,W              ; 1 / 1
BCF       STATUS,RP0            ; 1 / 1
MOVWF     tempStatus            ; 1 / 1
```

- MCS-8X51 @ 12 MHz
  - 每一指令需時1 to 4 cycle
  - 指令周期 = clk/12

```
                                ; byte/cycle
Int_Srv:                        ; 0/3 to 9
PUSHPSW                         ; 2/2
PUSHACC                         ; 2/2
MOV PSW,#08h                    ; 3/2
```

PICmicro MCU Architecture

Interrupt Comparison (con't)

- 效能與所需程式記憶體總結

	PIC16CXXX	MCS-8X51
所需程式記憶體	4	11
中斷延遲 (min)	3 cycles/1.0 µs	3 cycles/3 µs
中斷延遲 (max)	3 cycles/1.0 µs	9 cycles/9 µs
中斷前置處理時間	4 cycles/1.3 µs	6 cycles/6 µs
所需執行時間	7 cycles/2.3 µs	9-15 cycles/ 9-15 µs

開發工具介紹

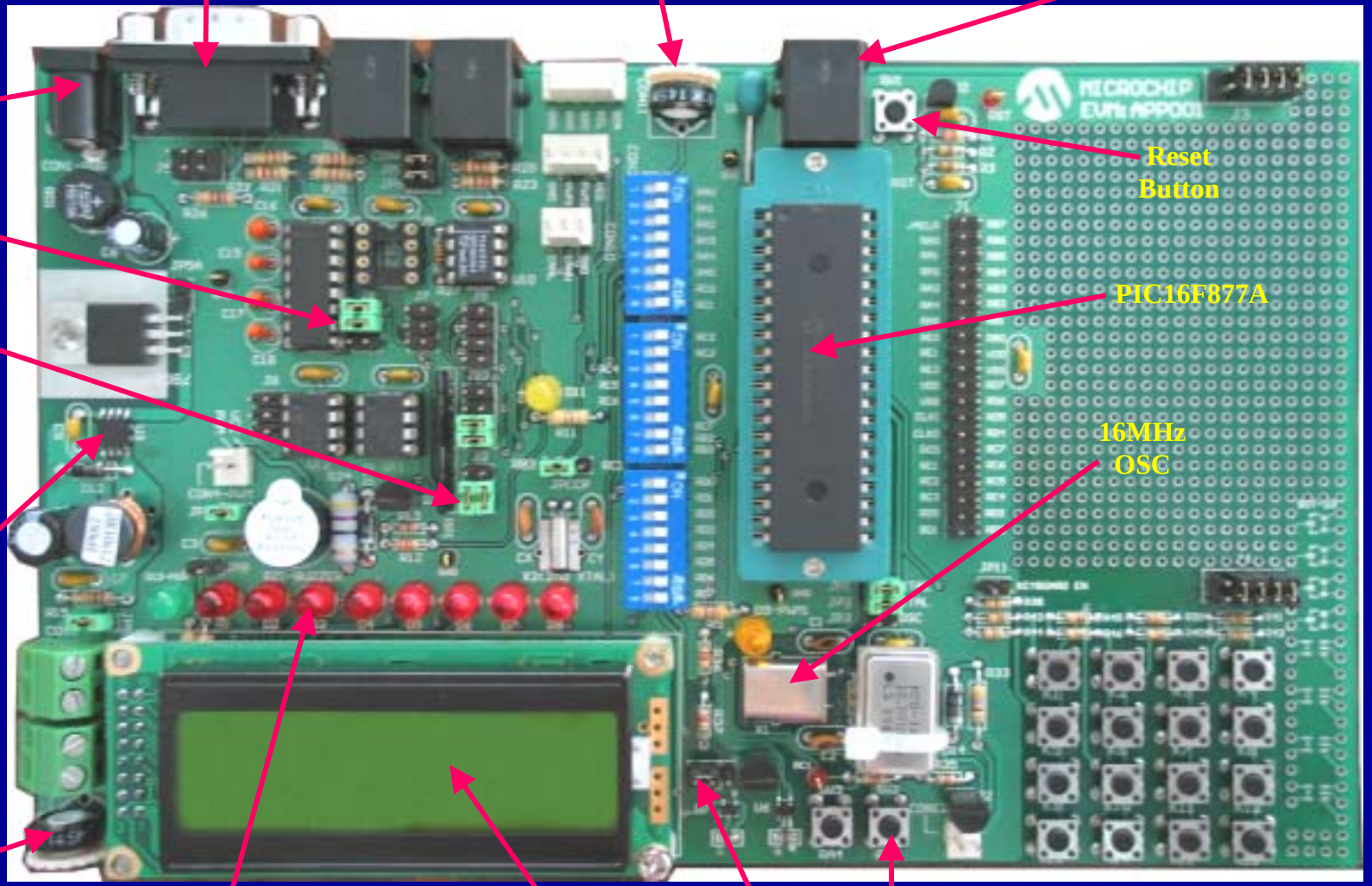
APP001 多功能實驗版

MPLAB IDE v6.xx

MPLAB ICE2000

MPLAB ICD2

APP001 實驗板


RS-232
A/D VR
ICD2
Connecter
DC 9V
Input
Reset
Button
J6
(RS-232)
PIC16F877A
J2
(32768Hz)
16MHz
OSC
+5V
S.P.S.
LCD
亮度調整
PORTD LED
2 X 16 LCD Module
J9
(LED)
Button

APP001 實驗板功能

- 練習 PIC16F877A 內部周邊的使用方法及典型的應用方式
- 40 Pin 的 PIC16Fxxx與 PIC18Fxxx 腳位相容
- 直接與 MPLAB-ICD2 連接 , 不須經由 Header Board 轉接
- 可練習的主要外接電路有:
  - LCD Module , 按鍵處理, I^2C EEPROM, I^2C溫度存取
  - SPI, RS-232 , RS-485 , CAN Controller
  - AD 轉換 , PWM 與 I/O 控制
- WORKSHOP 100 使用的主要資源
  - PORTD 用以控制 8 個獨立的 LED
  - RA0 : 使用於讀取可變電阻的電位值 (AN0)
  - RA4 : 使用於按鍵控制 (SW3)

MPLAB-IDE 功能介紹 (v6.xx)

- 高整合度的微處理器軟體 / 硬體研發及偵錯平台
- 採用專案管理模式 (Project)
- 多視窗原始程式編輯、修改
- 直接組譯 / 編譯原始程式
- 軟體模擬
- 具有輸入模擬功能
- 支援原始程式偵錯
- 支援 MPASM、 MPLINK
- 支援 C compiler

- 硬體除錯工具 :
  - MPLAB-ICE 4000
  - MPLAB-ICE 2000
  - MPLAB-ICD2
- 程式燒錄工具 :
  - PRO MATE - II
  - PICSTART Plus
  - MPLAB-ICD2

MPLAB™ v6.xx
整合式的發展環境

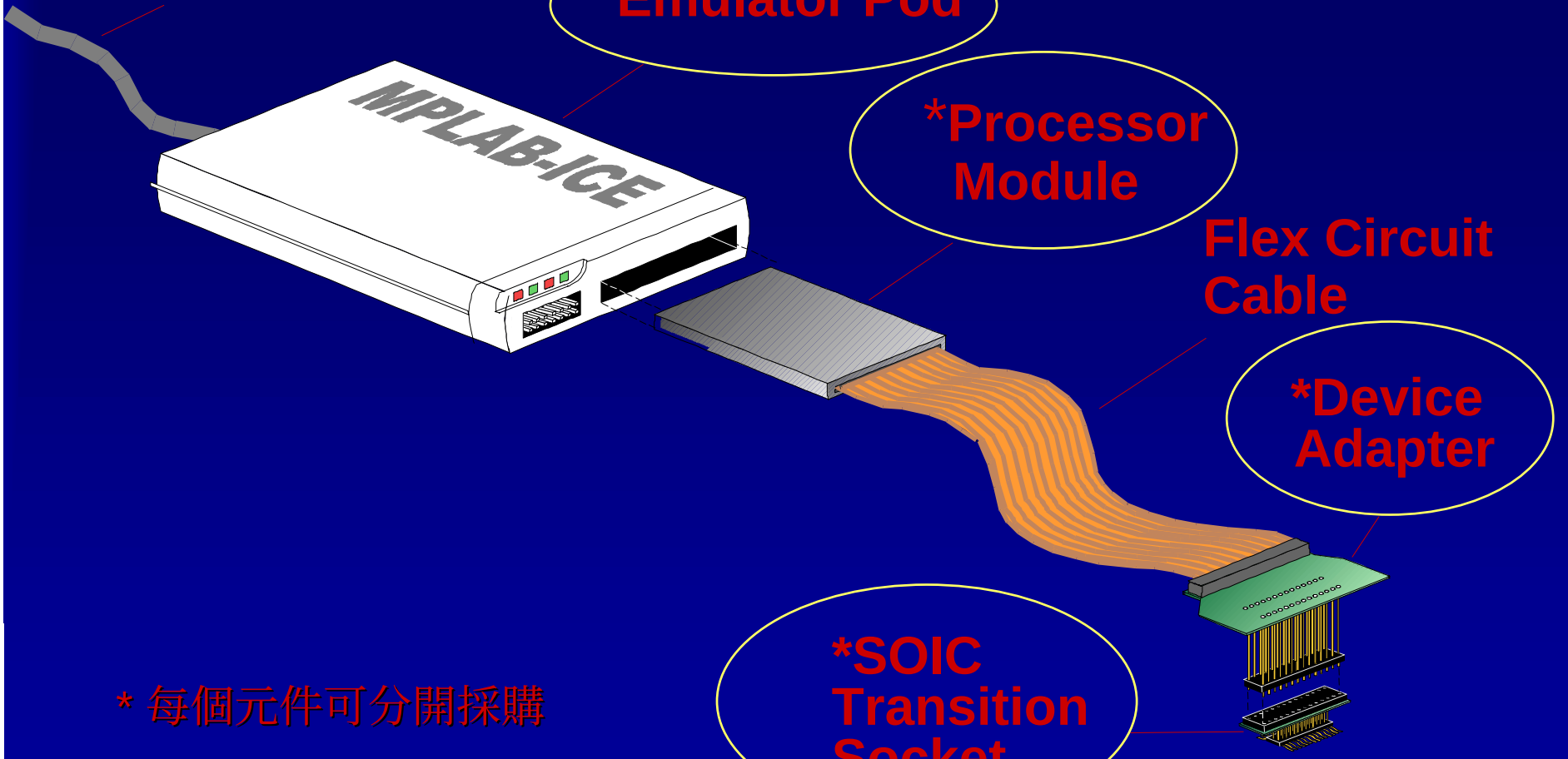
內含多功能程式編輯器

原始檔案程式偵錯功能

單一系統專案管理模式

語言工具	軟體模擬	硬體模擬器	程式燒錄器
MPASM 編譯器	MPLAB-SIM 軟體模擬器	MPLAB-ICE 2000 4000	PICSTART® Plus
MPLINK 連結器	dsPIC 軟體模擬器	PICMASTER (V5.xx)	PRO MATE®
MPLAB Cxx dsPIC C 編譯器		MPLAB ICD2	

MPLAB - ICE200



* 每個元件可分開採購

MPLAB – ICE2000

- 支援所有 PICmicro® MCU 系列的模擬
- 全速的模擬支援
  - Up to 25 MHz (PIC18CXXX)
  - 將有支援 40 Mhz 以上的版本
- 支援低操作電壓環境的模擬
  - 最低至 2.5 V
- 與 PC 使用 Parallel printer port 的界面
- Software programmable clock (在MPLAB-IDE 中可直接設定所需頻率)

MPLAB – ICE2000

- 改良甚多的 Trace 能力
  - 可將中斷點設置於 internal registers/RAM
  - 可以 Trace internal registers/RAM
  - 32K * 128 bits Trace Buffer 以及Logic analyzer
  - 可記錄執行階段的時間標記 (Time Stamp)
- Four level conditional break/trace/trigger
- 可精確量測兩事件間的時間間隔

  (Time between intervals)

MPLAB ICD 2

- 同時支援 USB 及 RS - 232 介面
- Support PIC16F及 PIC18F 系列具有 ICD 功能之 MCU 的除錯
- ICD Header 的接腳與 PIC16F87xA 使用的接腳位置相容
- 可設定中斷點, 單步執行以及進行變數的觀察
- 32K Bytes (PIC18F452) 的程式可於 3 秒內燒錄完成

MPLAB IDE 硬體支援

硬體設備	MPLAB v6.xx	MPLAB v5.xx
MPLAB-ICE4000	Yes	No
MPLAB-ICE2000	Yes	Yes
PICMASTER	No	Yes
ICEPIC	No	Yes
MPLAB-ICD	No	Yes
MPLAB-ICD2	Yes	Yes
PROMATE	Yes	Yes
PROMATE-II	Yes	Yes
PICSTART Plus	Yes	Yes

使用 MPLAB IDE v6.xx

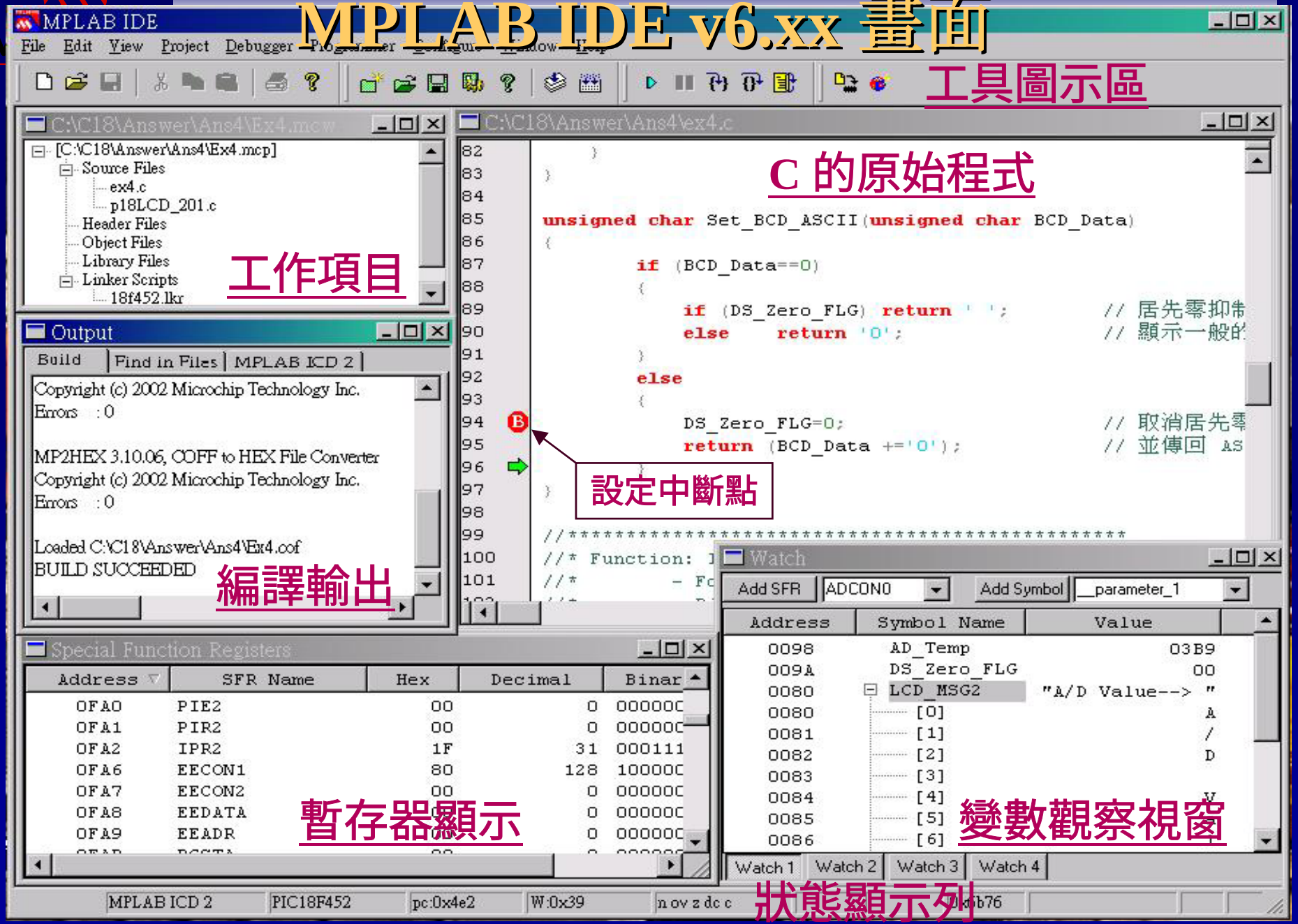
如何使用 MPLAB IDE V6.xx

- 詳細使用說明請詳細閱讀：
  - MPLAB v6.10 中文使用手冊
  - “www.microchip.com.tw”
- ICD2不知如何安裝 USB 驅動程式及使用：
  - MPLAB v6.10 中文使用手冊裡，有詳細的使用方法
  - 記住，使用 ICD2 之前一定要更新與 MPLAB IDE 相同版本的的作業系統
- 組譯工具 (MPASM) 與 C compiler 不知如何設定：
  - MPLAB v6.10 中文使用手冊
  - 範例程式可供參考

安裝 MPLAB IDE V6.xx

- 安裝 MPLAB IDE v6.xx
  - MPLAB IDE v6.xx 程式超過 20M，用 CD-ROM 安裝
  - 請參閱 MPLAB IDE v6.10 使用手冊，第 2-2 章的說明
  - 之前已有安裝舊版的 MPLAB IDE v6.xx，需先移除
  - 別忘了，還要安裝 ICD2 的 USB 驅動軟體
  - 如果有使用 ICE2000 還要再安裝驅動程式

MPLAB IDE v6.xx 畫面



了解相關檔案位置

- C:\Program Files\MPLAB IDE
  - ..\DricersXP (次目錄)
    - USB 的驅動程式 (ICD2 , ICD2000 , ICE4000)
  - ..\ICD2 (次目錄)
    - ICD2 的 Debugger 模組程式與作業系統
  - ..\MCHIP_Tools(次目錄)
    - MPLAB IDE 組譯工具， inc 檔案，lkr檔案 及 範本程式
  - ..\Programmers(次目錄)
    - PROMATE-II 的燒錄軟體版本
  - ..\READMES(次目錄)
    - MPLAB IDE 線上求助檔

f877atemp.asm 的內容

C:\Program Files\MPLAB IDE\MCHIP_Tools\TEMPLATE\Code\f877atemp.asm*

```
    list        p=16f877A       ; list directive to define processor
    #include    <p16f877A.inc>  ; processor specific variable definitions

    __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _RC_OSC & _WRT_OFF & _LVP_ON & _CPD_OFF

;***** VARIABLE DEFINITIONS
w_temp      EQU 0x71            ; variable used for context saving
status_temp EQU 0x72            ; variable used for context saving
pclath_temp EQU 0x73            ; variable used for context saving

;**********************************************************************
    ORG     0x000               ; processor reset vector
    nop                         ; nop required for icd
    goto    main                ; go to beginning of program

    ORG     0x004               ; interrupt vector location
    movwf   w_temp              ; save off current W register contents
    movf    STATUS,w            ; move status register into W register
    movwf   status_temp         ; save off contents of STATUS register
    movf    PCLATH,w            ; move pclath register into w register
    movwf   pclath_temp         ; save off contents of PCLATH register

; isr code can go here or be located as a call subroutine elsewhere

    movf    pclath_temp,w       ; retrieve copy of PCLATH register
    movwf   PCLATH              ; restore pre-isr PCLATH register contents
    movf    status_temp,w       ; retrieve copy of STATUS register
    movwf   STATUS              ; restore pre-isr STATUS register contents
    swapf   w_temp,f
    swapf   w_temp,w            ; restore pre-isr W register contents
    retfie                      ; return from interrupt

main

; remaining code goes here

    END                         ; directive 'end of program'
```

啟動 MPLAB IDE v6.xx

- 有關 **MPLAB IDE v6.xx** 的基本使用方式，請參考 **MPLAB IDE v6.10** 中文使用手冊第三章的說明。

- 使用 **MPLAB IDE** 為發展工具時，必須先建立一個專用的 **Project**，建立 **Project** 有一定的程序不可以混淆。

專案管理 (Project Management)

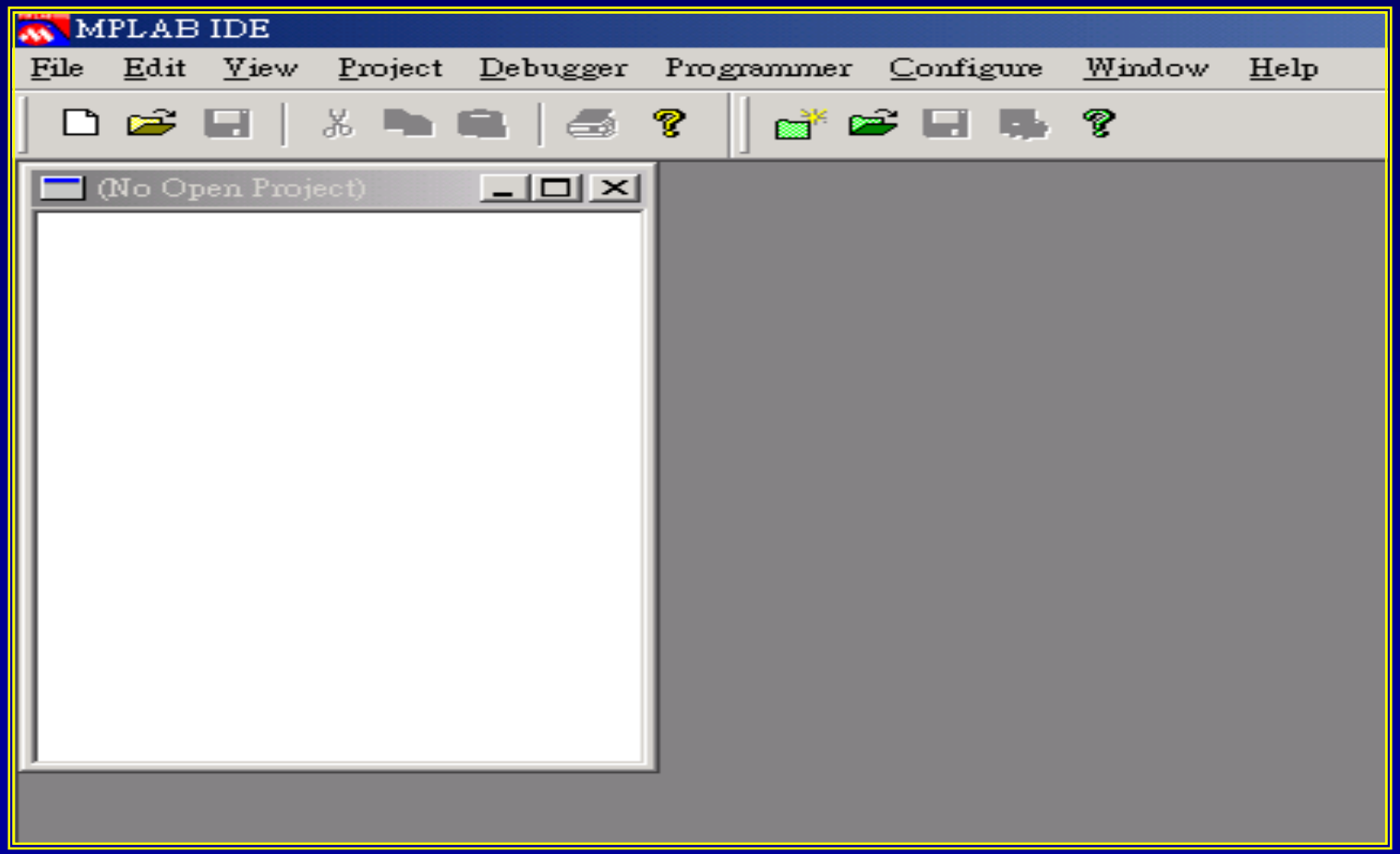
- MPLAB IDE 是採專案管理方式來完成軟體研發與設計，所以在使用 MPLAB IDE 時，就必需建立一個 Project。
- Project 的附加檔案名稱是 xx.mcp，最好與原始檔案放在同一個目錄以方便管理。
- 一個 Project 可記錄眾多資訊：
  - 視窗位置、大小、個數
  - 相關檔案的名稱、位置
  - 相關偵錯訊息的設定值
  - 組譯器、編譯器的選擇與設定
- 以 Project 觀念來保持一個完整的軟體設計，以便日後程式的維護、修改。

MPLAB IDE v6.10 (步驟 1 2 3 …)

步驟	動 作 選 項	功　　能
1	Configure→Select Device	選擇要使用的 PICmicro 然後按OK
2	Configure→Configuration Bits	設定 PIC 的工作的模式
3	Project→New	建立 Project 名稱及其路徑
4	Project →Set Language Toolsuite	選擇語言工具(C18 或 MPASM)
5	Project→Set Language Tool Locations	確定語言工具的執行路徑
6	File→New (open)　& File→Save As	建立你的原始程式檔案後存檔
7	Project→Build Options→Project	設定資料庫、含入檔及連結描述檔的路徑
8	Project Save　& Build All	儲存Project的設定與開始編譯程式

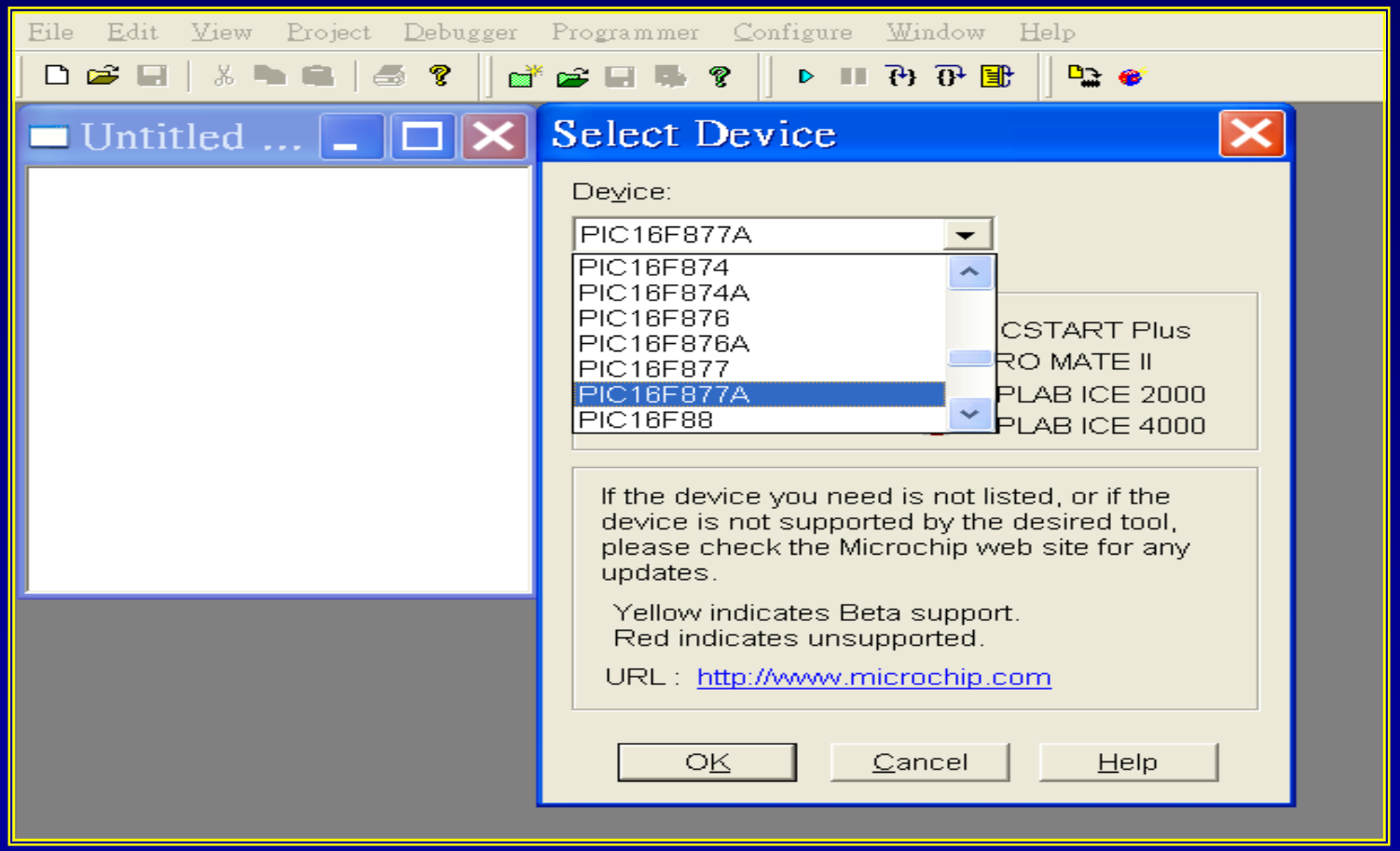
練習一：建立新的專案 (Project)

- 執行 MPLAB IDE ，將顯示一初始的空白桌面



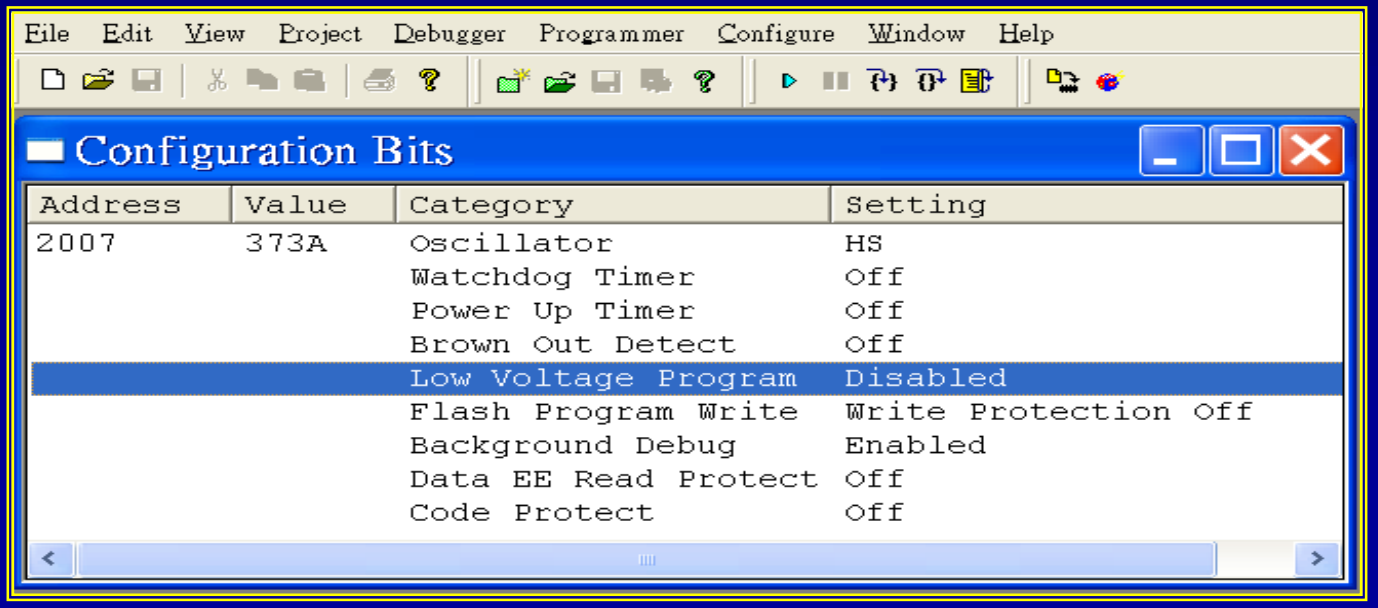
練習一：建立新的專案 (動作一)

- 使用 “Configure → Select Device” 來指定 MCU 為 PIC16F877A



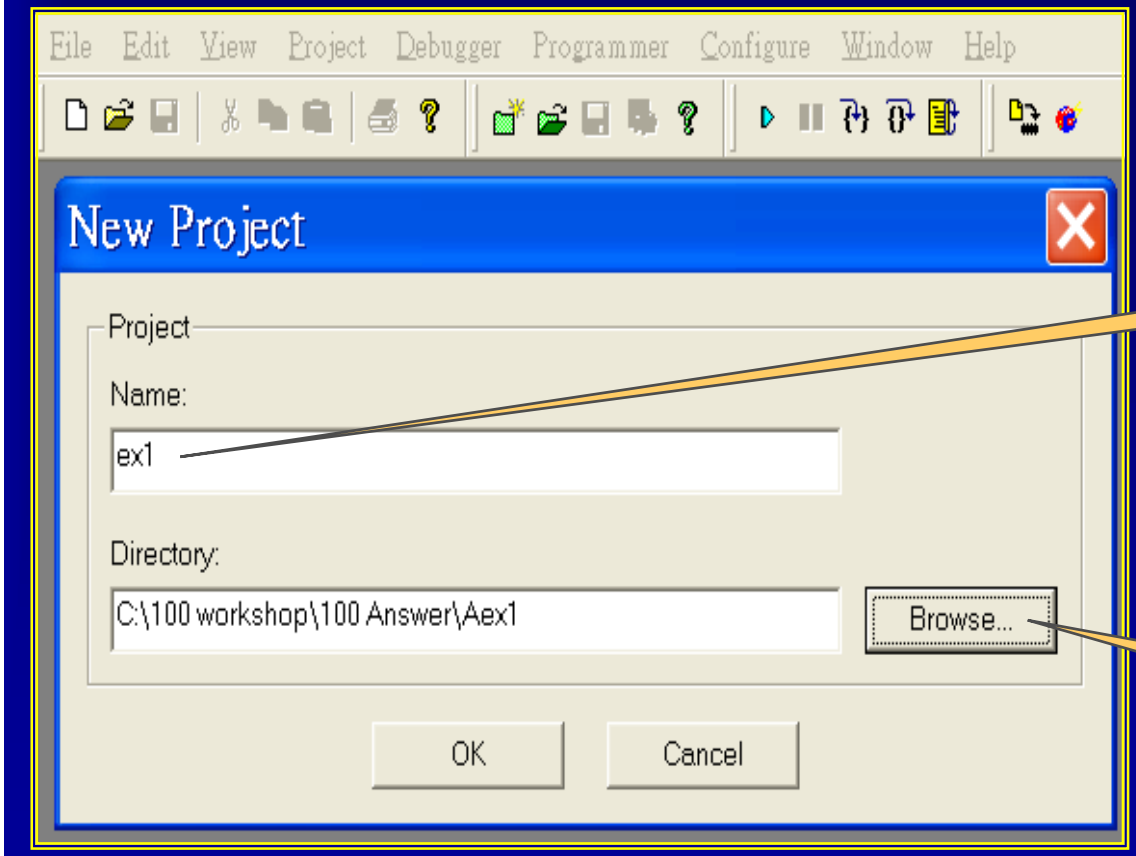
練習一：建立新的專案 (動作二)

- 使用 “Configure → Configuration Bits” 來設定 IC 運作模式
  - Oscillator 設定為 HS Mode
  - Watchdog 設定為 Disabled
  - Low Voltage Prog. 設定為 Disabled
  - Background Debug 設定為 Enabled 以便使用 MPLAB ICD 2



練習一：建立新的專案 (動作三)

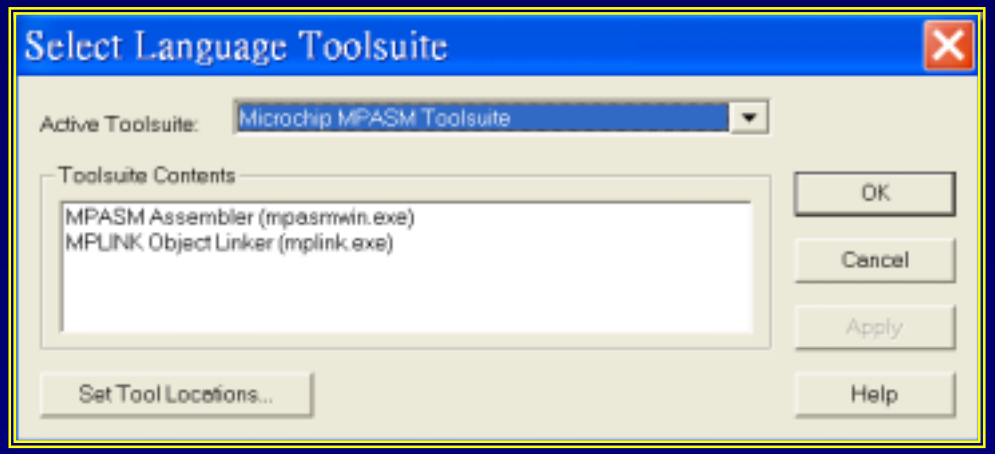
- 使用 **“Project → New Project”** 來建立一個新的專案



輸入專案名稱 **ex1.asm**

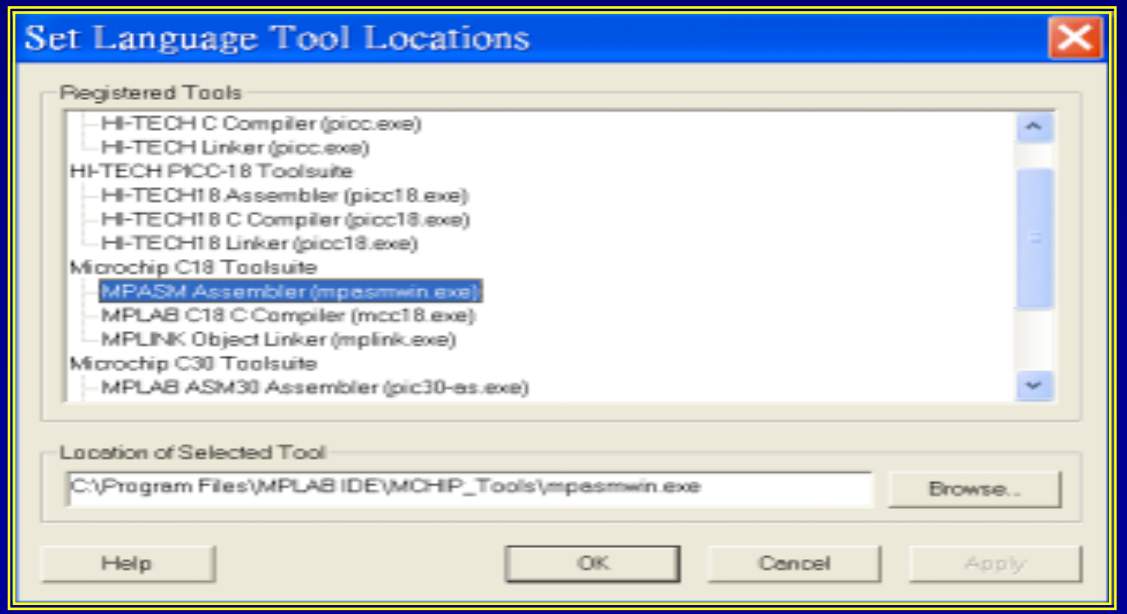
用 **Browse** 來指定目錄於 **C:\W400**

練習一：建立新的專案 (動作四&五)

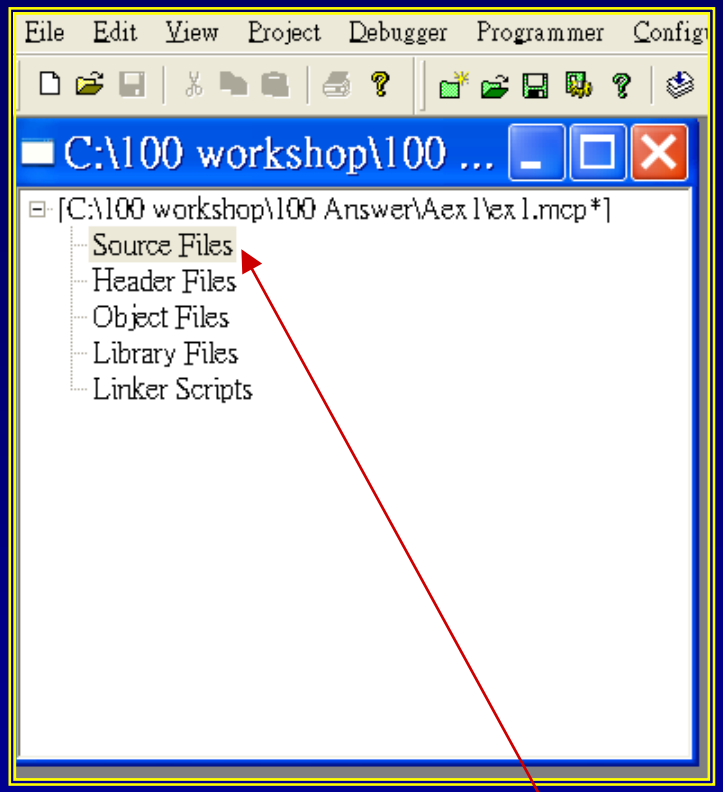


- 使用 **“Project → Set Language Toolsuite”** 來確定是使用 **MPASM** 來組譯組合語言

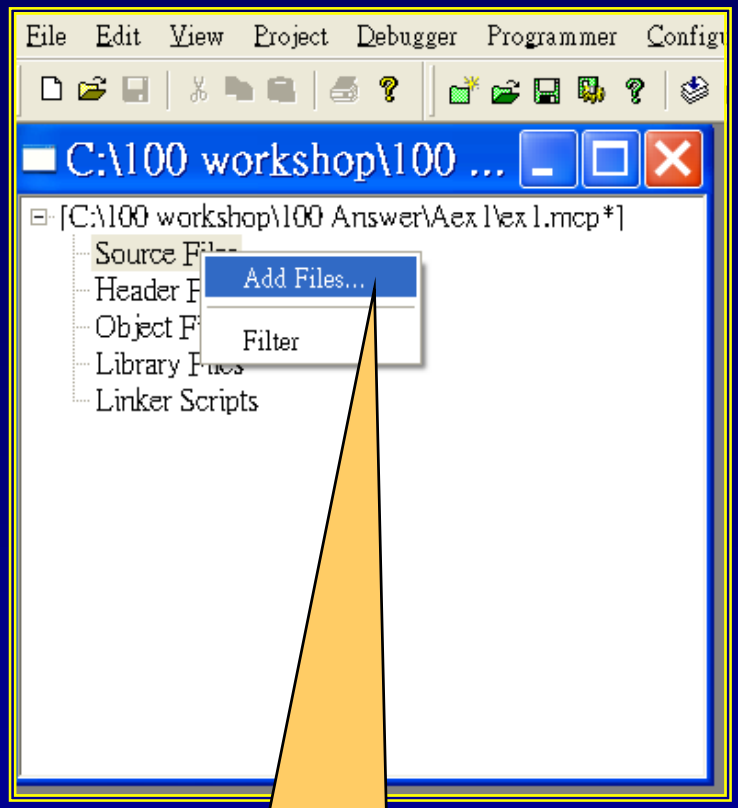
- 使用 **“Project → Set Language Tool locations”** 來確定 **MPASM** 程式所執行的位置



練習一：建立新的專案 (動作六)

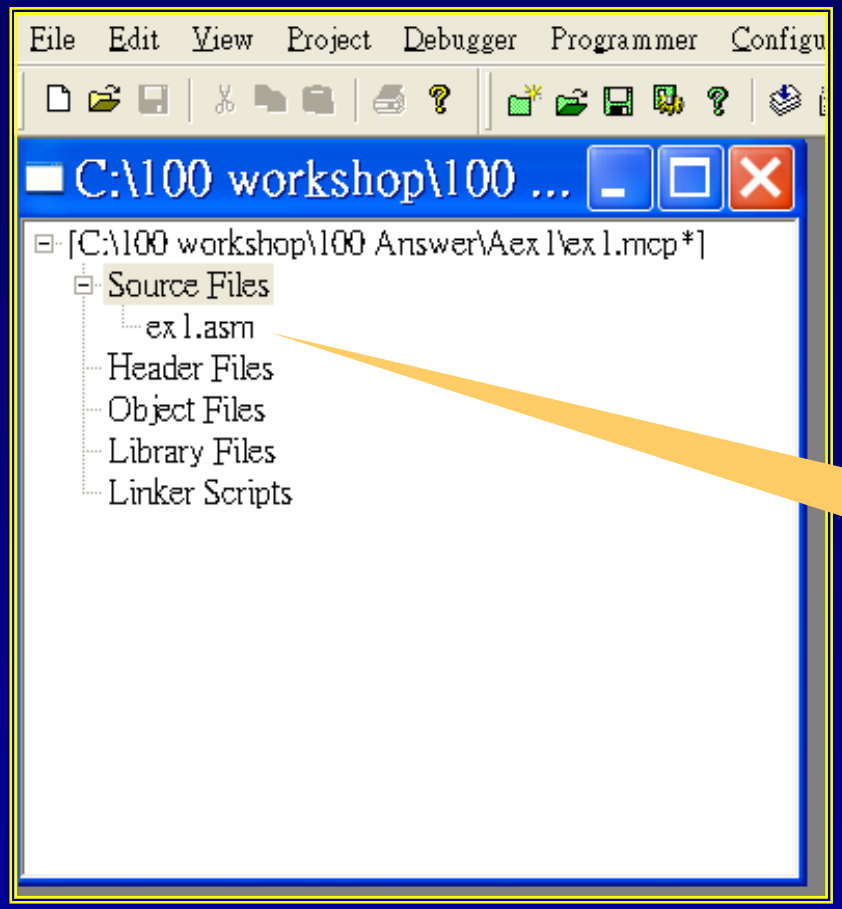


將滑鼠指到
Source Files
並按下滑鼠的右鍵



MPLAB IDE
將要求您加入檔案

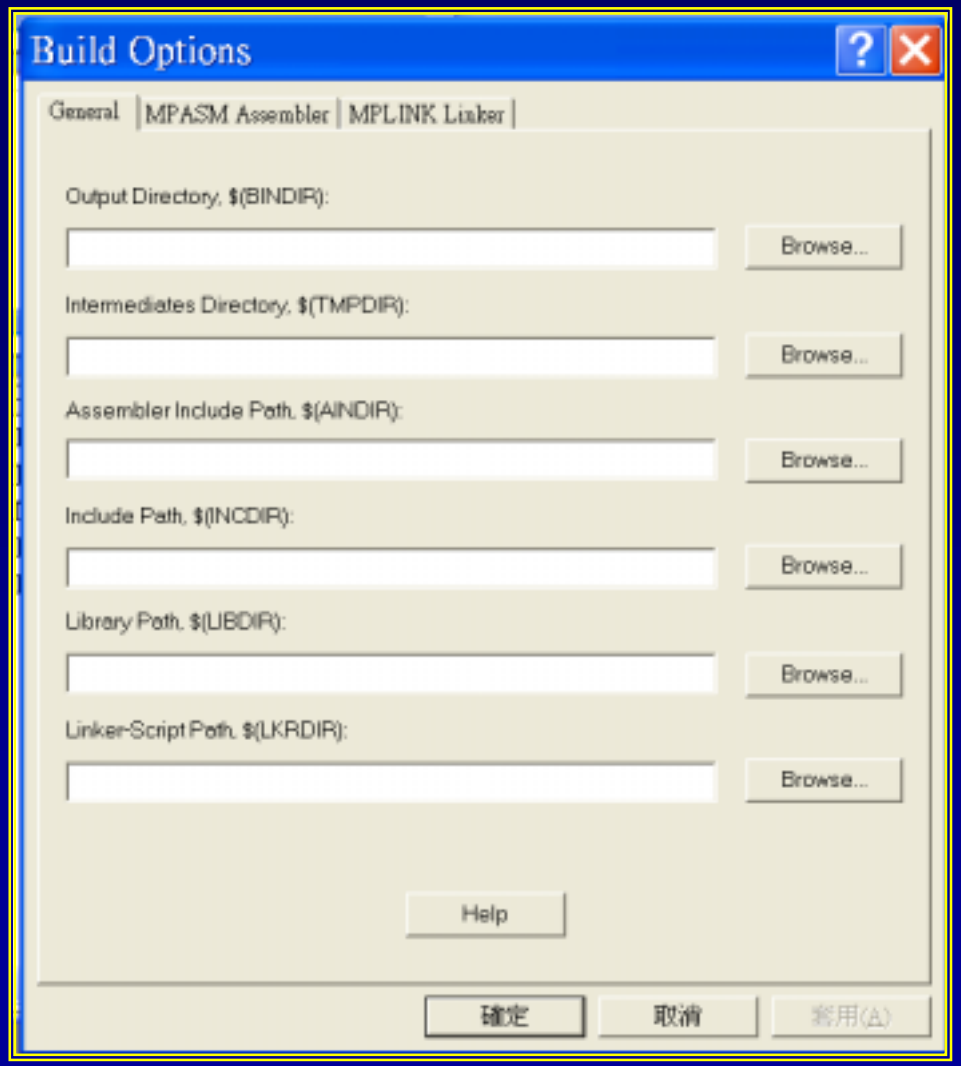
練習一：建立新的專案 (動作六)



- **Project 名稱為 ex1.mcp**
- **組合語言程式為 ex1.asm**

用滑鼠左鍵雙擊
ex1.asm 即可編輯
此檔案

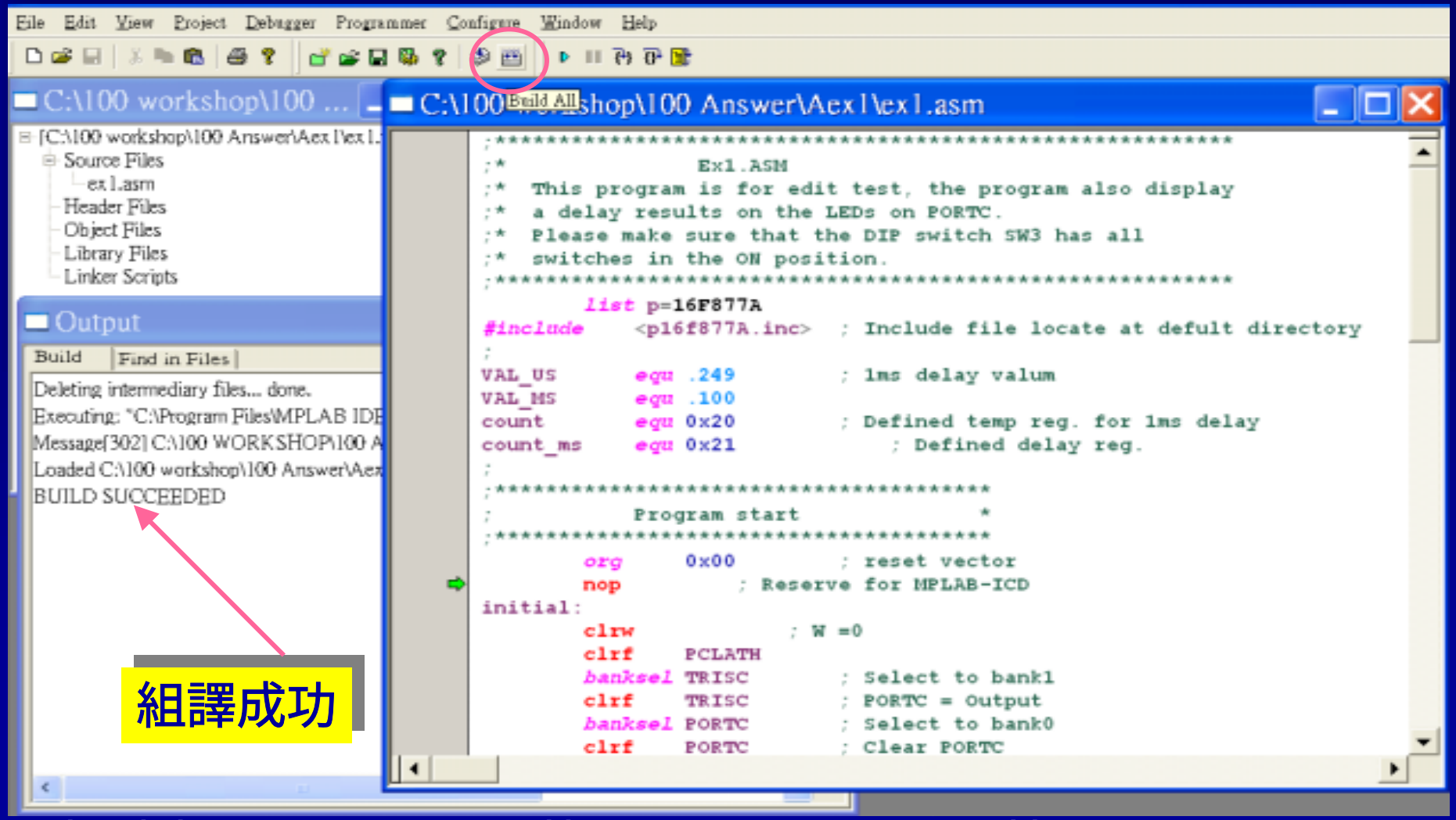
練習一：建立新的專案 (動作七)



- 使用“**Project → Build Options → Project** “啟動輸入視窗
- 該視窗是輸入資料庫、含入檔及連結描述檔的路徑位置，一般是使用 **C** 語言及多檔案的組合語言才需要輸入
- 使用單一組合語言者無需輸入路徑資料

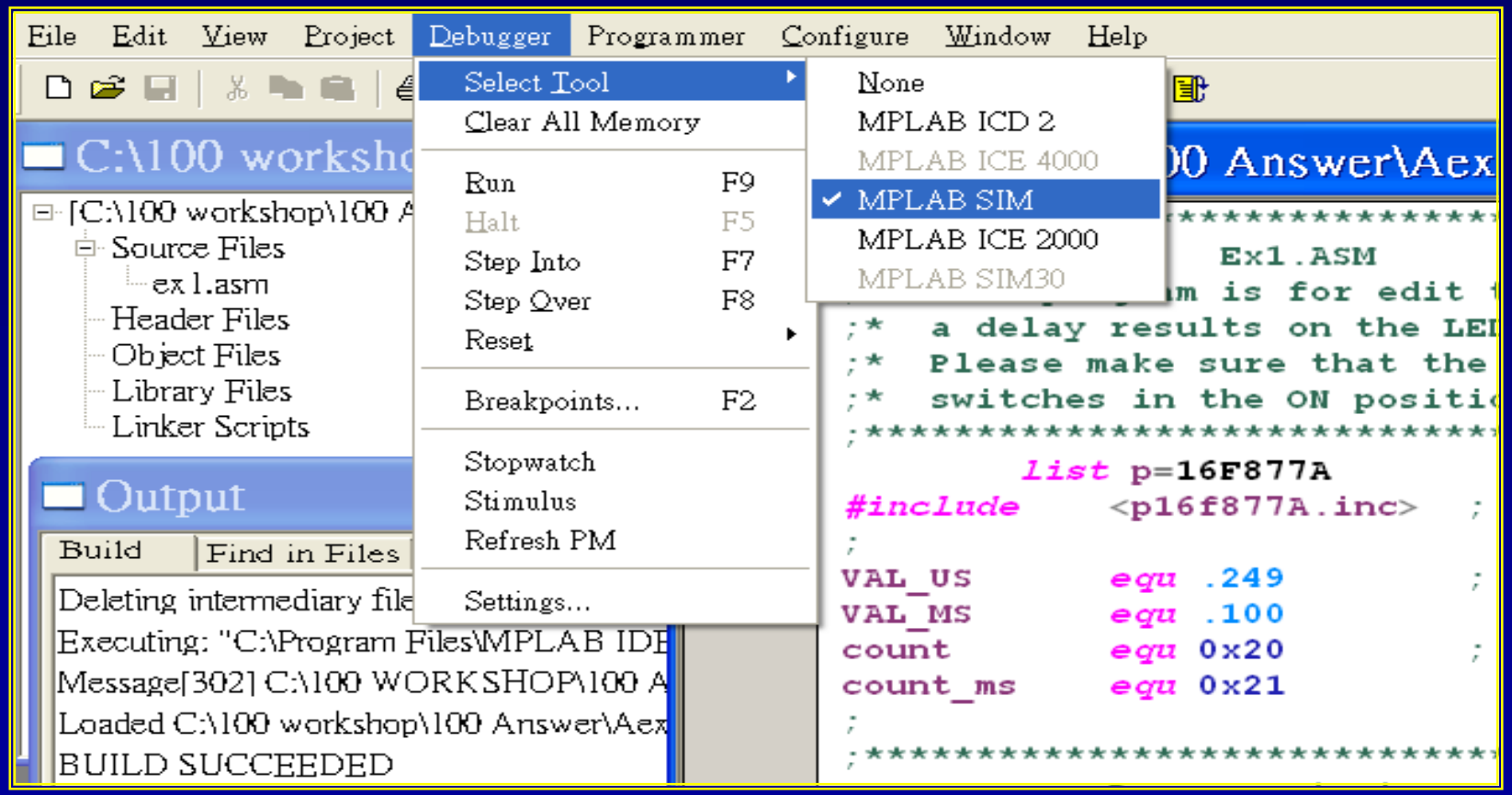
練習一：建立新的專案 (動作八)

- 使用 Project → Build All 或 Ctrl+F10 來組譯程式

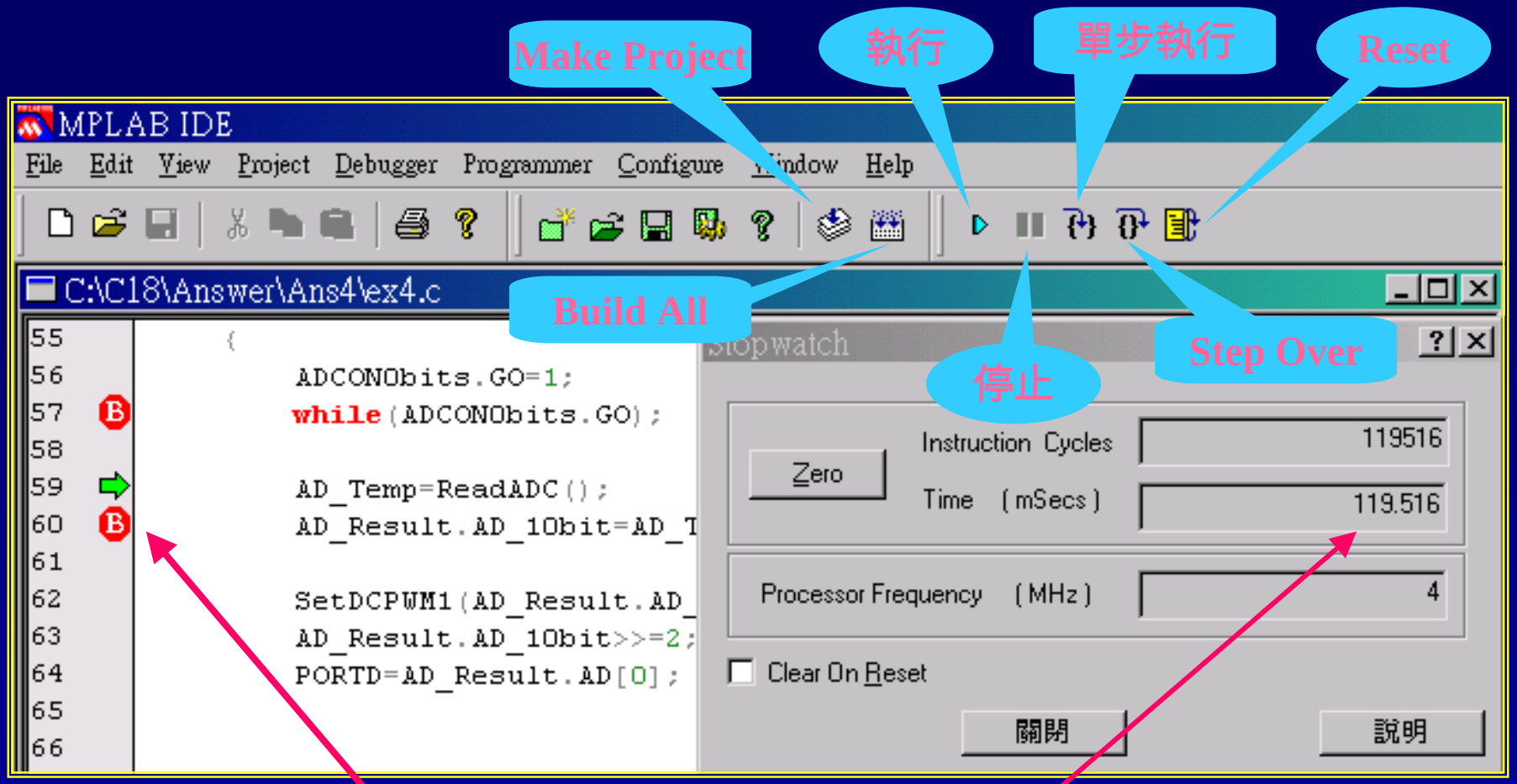


啟動基本 MPLAB-SIM

- 點選 “Debugger → Select Tool → MPLAB SIM” 以啟動內建的軟體模擬程式



MPLAB SIM 的基本除錯功能

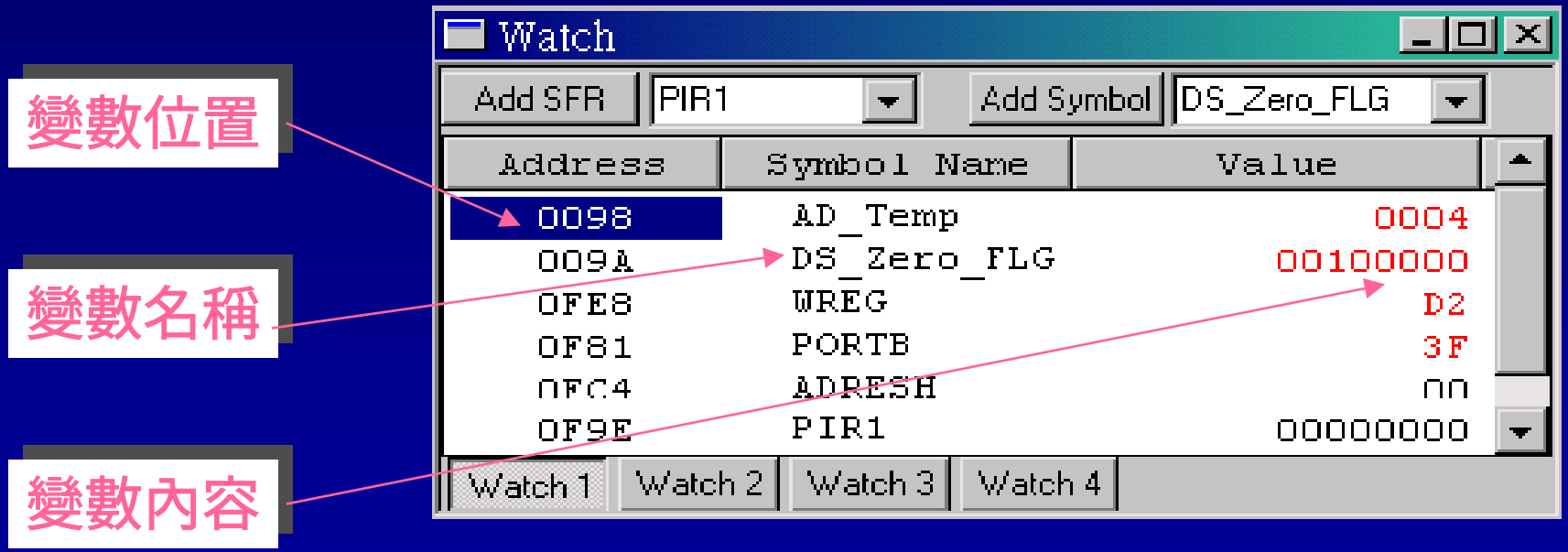


利用mouse的右鍵來設定中斷點

利用Stopwatch視窗來量測程式執行所需的時間

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



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 組合 語言下就不是那麼重要了**?**
  - 其它的視窗 **?**

PIC16F877A
基本指令介紹

14-Bits Core 指令集（35個）

位元組操作指令

NOP	-	No Operation
MOVWF	f	Move W to f
CLRW	-	Clear W
CLRF	f	Clear f
SUBWF	f,d	Subtract W from f
DECF	f,d	Decrement f
IORWF	f,d	Inclusive OR W and f
ANDWF	f,d	AND W and f
XORWF	f,d	Exclusive OR W and f
ADDWF	f,d	Add W and f
MOVF	f,d	Move f
COMF	f,d	Complement f
INCF	f,d	Increment f
DECFSZ	f,d	Decrement f, skip if zero
RRF	f,d	Rotate right f through carry
RLF	f,d	Rotate left f through carry
SWAPF	f,d	Swap nibbles of f
INCFSZ	f,d	Increment f, skip if zero

位元操作指令

BCF	f,b	Bit clear f
BSF	f,b	Bit set f
BTFSC	f,b	Bit test f, skip if clear
BTFSS	f,b	Bit test f, skip if set

常數操作及控制指令

SLEEP	-	Go into standby mode
CLRWDT	-	Clear watchdog timer
RETLW	k	Return, place literal in W
RETFIE	-	Return from interrupt
RETURN	-	Return from subroutine
CALL	k	Call subroutine
GOTO	k	Go to address (k is 11-bit)
MOVLW	k	Move literal to W
IORLW	k	Inclusive OR literal with W
ADDLW	k	Add literal with W
SUBLW	k	Subtract W from literal
ANDLW	k	AND literal with W
XORLW	k	Exclusive OR literal with W

f = 暫存器或記憶位址， k = 常數值 (8-bit)， b = 第幾位元 <0,7>， d = 運算後資料目的地 (1=f, 0=W)

資料移動指令
(MOVF , MOVWF)

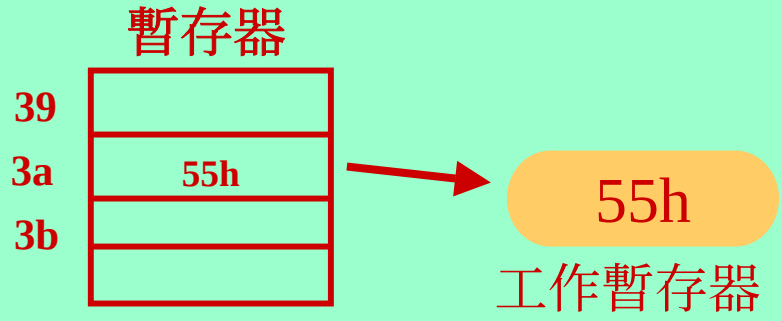
MOVF

- 語法： MOVF f , d
- 操作：將所指到的暫存器 (f) 的內容傳送到工作暫存器 (w)

例:

movf h'3A',W

將位址 $3A_{(16)}$ 的暫存器內容傳送到工作暫存器 (w) 中．。



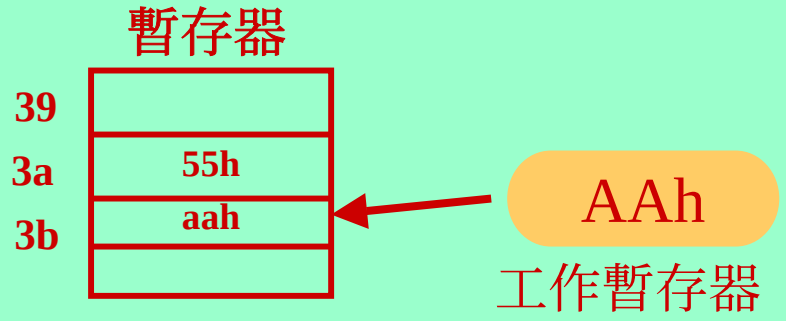
MOVWF

- 語法： MOVWF f
- 操作：將工作暫存器 (w) 的內容傳送所指到的暫存器 (f)

例:

movwf h'3B'

將工作暫存器 (w) 中的內容傳送到位址為 $3B_{(16)}$ 的暫存器中．。



資料遞減指令
(DECF , DECFSZ)

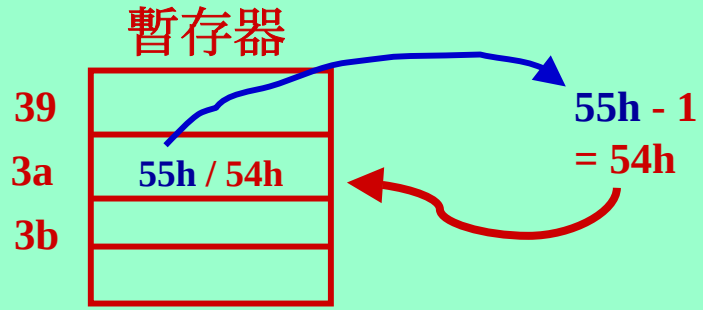
DECF

- 語法： DECF f , d
- 操作 : 將所指到的暫存器 (f) 的內容減一後回存到 (d)

例:

```
decf    h'3A', F
```

將位址 $3A_{(16)}$ 的暫存器內容減一後回存到位址為$3A_{(16)}$暫存器中。



DECFSZ

- 語法： DECFSZ f , d
- 操作 :將所指到的暫存器 (f) 的內容減一後回存到 (d)，並測試減一後的結果是不是等於零以決定是否跳過下一個指令。
  - 結果不等於零：執行下一個指令
  - 結果等於零：忽略下一個指令而直接執行下下一個指令

例:

```
decfsz   h'3A' , F
goto     結果不等於零
call     結果等於零
```

位元清除、設定指令
(BCF , BSF)

BCF

- 語法： BCF f , b
- 操作：將所指到的暫存器 (f) 中的第 (b) 個位元設定為 0。

例:

```
PORTA   equ    h'05'
        bcf    PORTA , 2
```

將位址 $05_{(16)}$ 的暫存器 (PORTA) 的 bit 2 清除為 “0”。

```
        bcf    h'3a' , 7
```

將位址 $3a_{(16)}$ 的暫存器的 bit 7 清除為“0”。

BSF

- 語法： BSF f , b
- 操作：將所指到的暫存器 (f) 中的第 (b) 個位元設定為 1。

例:

```
PORTA   equ    h'05'
        bsf    PORTA , 2
```

將位址 $05_{(16)}$ 的暫存器 (PORTA) 的 bit 2 設定為 “1”。

```
        bsf    h'3a' , 7
```

將位址 $3a_{(16)}$ 的暫存器的 bit 7 設定為 “1”。

位元測試指令
(BTFSC , BTFSS)

BTFSC

- 語法： BTFSC f , b
- 操作 :測試所指到的暫存器 (f) 的第 (b) 個的位元內容是不是等於 “0” 以決定是否跳過下一個指令。
  - Bit (b) 不等於 “0”：執行下一個指令 (條件不成立 ）。
  - bit (b) 等於 “0”：忽略下一個指令而直接執行下下一個指令，也就是說其測試條件成立。

例:

```
btfsc   h’3A’ , 5
goto    結果不等於 “0”
goto    結果等於 “0”
```

BTFSS

- 語法： BTFSS f , b
- 操作 :測試所指到的暫存器 (f) 的第 (b) 個的位元內容是不是等於 “1” 以決定是否跳過下一個指令。
  - Bit (b) 不等於 “1”：執行下一個指令 (條件不成立 ）。
  - bit (b) 等於 “1”：忽略下一個指令而直接執行下下一個指令，也就是說其測試條件成立。

例:

```
btfss   h’3A’ , 5
goto    結果不等於 “1”
goto    結果等於 “1”
```

常用的基本指令
(ADDWF , SUBWF , GOTO , CALL , RETURN)

ADDWF

- 語法： ADDWF f , d
- 操作 :將所指到的暫存器 (f) 的內容與工作暫存器 (W)的內容相加後，放置在目地暫存器 (d)

GOTO

- 語法： GOTO addr
- 操作 : 無條件的直接跳到所指定位址 (addr) 執行程式。

SUBWF

- 語法： SUBWF f , d
- 操作 :將所指到的暫存器 (f) 的內容減去工作暫存器 (W)的內容，其差放置在目地暫存器 (d)

CALL

- 語法： CALL sub
- 操作 : 無條件的直接跳到所指定的副程式位址 (sub) 執行該副程式。

常數載入、運算指令
(MOVLW , ADDLW , SUBLW, IORLW , ANDLW)

MOVLW

- 語法： MOVLW　　H’3A’
- 操作 :將所指定的常數 (3A) 載入到工作暫存器 (W)中。

IORLW

- 語法： IORLW　B’11110000’
- 操作 :將所指定的常數 (F0)與工作暫存器 (W)的內容做 OR Gate 的運算後，放回工作暫存器 (W) 。

SUBLW

- 語法： SUBLW　　H’3A’
- 操作 :將所指定的常數 (3A)減去工作暫存器 (W)的內容，其差放回工作暫存器 (W)

ANDLW

- 語法： ANDLW　B’00001111’
- 操作 :將所指定的常數 (0F)與工作暫存器 (W)的內容做 AND Gate 的運算後，放回工作暫存器 (W) 。

正確的語法表達 (一)
數值、數字(字元)表示法

- 十進制表示(Decimal): D'<十進制數目>' & . <十進制數目 >
  - MOVLW D'100' ; 載入常數 $100_{(10)}$ to W Reg.
  - MOVLW .100 ; 載入常數 $100_{(10)}$ to W Reg.
  - Const1 EQU D'200' ; Const1 = 200 (十進制)

- 十六進制表示(Hexadecimal): H'<十六進制數目>' & 0x<十六進制數目> & <十六進制數目> h
  - MOVLW H'3F' ; 載入常數 $3F_{(16)}$ to W Reg.
  - MOVLW 0x3F ; 載入常數 $3F_{(16)}$ to W Reg.
  - MOVLW 0FEh ; 載入常數 $FE_{(16)}$ to W Reg.
  - Const1 EQU H'5A' ; Const1 = 5A (十六進制)

正確的語法表達 (二)
數值、數字(字元)表示法

- 二進制表示(Binary): B’<二進制數目>‘

```
        MOVLW   B’11110000’   ; 載入常數 0xF0 to W Reg.
Const1  EQU     B’01010101’   ; Const1 = 01010101 (二進制)
```

- 字元(ASCII): A’<字元>‘ & ’<字元>‘

```
        MOVLW   A‘R’          ; 載入字元 “R” to W Reg.
        MOVLW   ‘c’           ; 載入字元 “ c” to W Reg.
Const1  EQU     ‘a’           ; Const1 = 小寫 的字元 A
```

正確的語法表達 (三)
組合語言的語法

- 指令 (虛擬指令) & 運算元

```
MOVF    f , d     ; Move RAM/Register to Destination
```

f　暫存器 或 RAM 的位址

d　選擇資料移送的目的暫存器

d = 0 時，放置在 w 暫存器，亦可寫成 “W”

d = 1 時，放回在 f 暫存器，亦可寫成 “F” 或忽略不寫

範例:

```
var_count   equ     0x3f
;
            addwf   var_count , 0
            addwf   var_count , 1
            addwf   var_count , W
            addwf   var_count , F
```

建議語法

必需要使用的虛擬指令

- **LIST - 目錄控制 (Listing Control)**
  - `list p=PIC16F877, f=INHX8M`
- **#INCLUDE - 加入一原始檔、定義檔或敘述檔**
  - `#include <C:\MPLAB\16F877.INC>`
- **EQU - 宣告常數、變數 (不可重新定位)**
  - `memory equ 0x3f`
  - `count equ .100`
  - `io_set equ B'11000011'`
- **ORG - 設定程式組譯的起始位址**
  - `org 0x00 ; 組譯位址從"00h"開始`
  - `org 0x30 ; 組譯位址從"30h"開始`
- **END - 程式結束**

虛擬指令的使用範例

```
            list        p=16f877a           ; 定義使用的 MCU 為 PIC16F877A
            #include    <p16f877a.inc>      ; 使用 16F877 的標準定義檔
;***** 定義變數‘ 常數、參數區 ******
T_DELAY     EQU         D’100’
dly_count   EQU         H’70’
;*****************************************************************************
            ORG         0x000               ; 設定程式執行位址從“0”開始 (Reset Vector)
            clrf        PCLATH
            goto        main
            ORG         0x004               ; 中斷向量進入位址
;******* 中斷處理副程式區 ******
            retfie                          ; 中斷返回
;
main        movlw       T_DELAY             ; 主程式開始
            movwf       dly_count
                  :
; remaining code goes here
            END                             ; 程式結束
```

暫存器頁(BANK)的自動設定

BANKSEL ADCON1

- 依據 BANKSEL 指令所指到的暫存器，自動加入該暫存器頁(BANK)的設定指令，此指令會依不同的 PIC 自行判斷頁數。

以PIC16F77為例 ： 共有四個暫存器頁(BANK) 就會使用到兩個換頁指令

```
000B   1683 1303      00049                 banksel ADCON1
Message[302]: Register in operand not in bank 0.  Ensure that bank
000D   019F           00050                 clrf    ADCON1
Message[302]: Register in operand not in bank 0.  Ensure that bank
000E   170C           00051                 bsf     PIE1,ADIE
000F   1283 1303      00052                 banksel ADCON0
0011   30C1           00053                 movlw   h'c1'
0012   009F           00054                 movwf   ADCON0
```

本文檔的編輯 (EDIT)

- 檔案管理 (File)
  - 開啟新檔 (New File)
  - 開啟舊檔 (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

練習二：輸入一個組合語言程式

Ex1.ASM (第一頁)

```
;*********************************************************
;*  This program is for edit test, the program also display
;*  a delay results on the LEDs on PORTD.
;*  Please make sure that the DIP switch SW3 has all
;*  switches in the ON position.
;*********************************************************
          list p=16F877A
#include  <P16F877A.inc>                ; Include file locate at default directory
;
VAL_US              equ       .160      ; 1ms delay valum
VAL_MS              equ       .100
count     equ       0x20                ; Defined temp reg. for 1ms delay
count_ms  equ       0x21                ; Defined delay reg.
;
;*************************************
;*          Program start             *
;*************************************
          org       0x00                ; reset vector
          nop                           ; Reserve for MPLAB-ICD
initial:
          clrw                          ; W =0
          clrf      PCLATH
          banksel   TRISD               ; Select to bank1
          clrf      TRISD               ; PORTC = Output
          banksel   PORTD               ; Select to bank0
          clrf      PORTD               ; Clear PORTC
```

練習二：輸入一個組合語言程式

Ex1.ASM (第二頁)

```
;
;************ Main *********************
;
start:
          incf      PORTD,f             ; PORTD = PORTD + 1
          call      delay_100ms         ; Call delay routine
          goto      start
;
;-------- 100 ms delay routine ----------
delay_100ms:
          movlw     VAL_MS
          movwf     count_ms
loop_ms   call      delay_1ms
          decfsz    count_ms,f
          goto      loop_ms
          return
;
;-------- 1 ms delay routine -----------
delay_1ms:
          movlw     VAL_US
          movwf     count
dec_loop  call      D-short
          decfsz    count,f
          goto      dec_loop
          return
;
```

```
;--------     Delay 5uS
D_short   call      D_ret


          call      D_ret
          call      D_ret
          call      D_ret
          nop
          nop
D_ret     return
;
          end
```

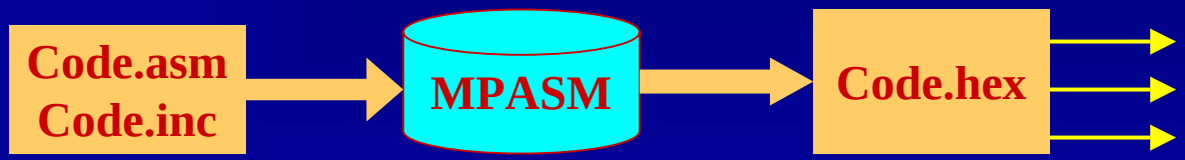
練習三：認識檔案

- 利用練習二所輸入的原始程式 **EX1.ASM** 來建立一個新的以 **EX1.mcp** 為名的 **Project**。
- 將**Ex1.mcp** 經 **Build-All** 後，確定沒有組譯錯誤產生
- 利用 **“File → Open”** 將所有的檔案顯示出來
- 試著將這些檔案打開，並了解它們是什麼樣的檔案

MPASM
組合語言組譯器

- MPASM 是 Microchip PICmicro Assembler 的縮寫
- MPASM 的功能是將組合語言翻譯成 Intel Hex 格式的機械碼或是翻譯成可連結的 OBJ 碼
- 目前 W100 的範例程式是採用單一組合語言架構

單一原始組合語言檔的組譯方式



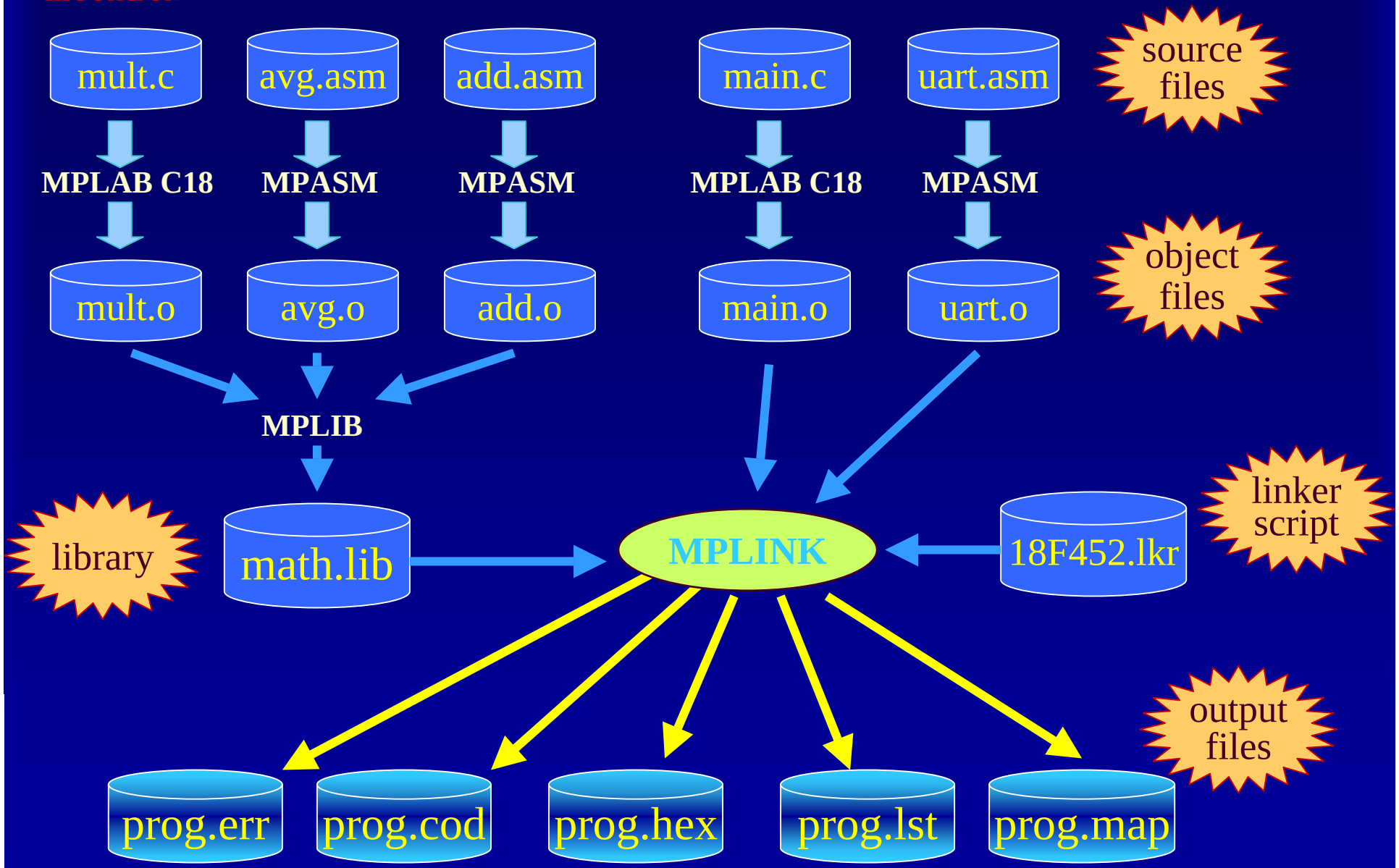
燒錄器 (PRO MATE - II)
軟體模擬 (MPLAB-SIM)
硬體除錯器 (ICE - 2000)
(MPLAB ICD2)

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

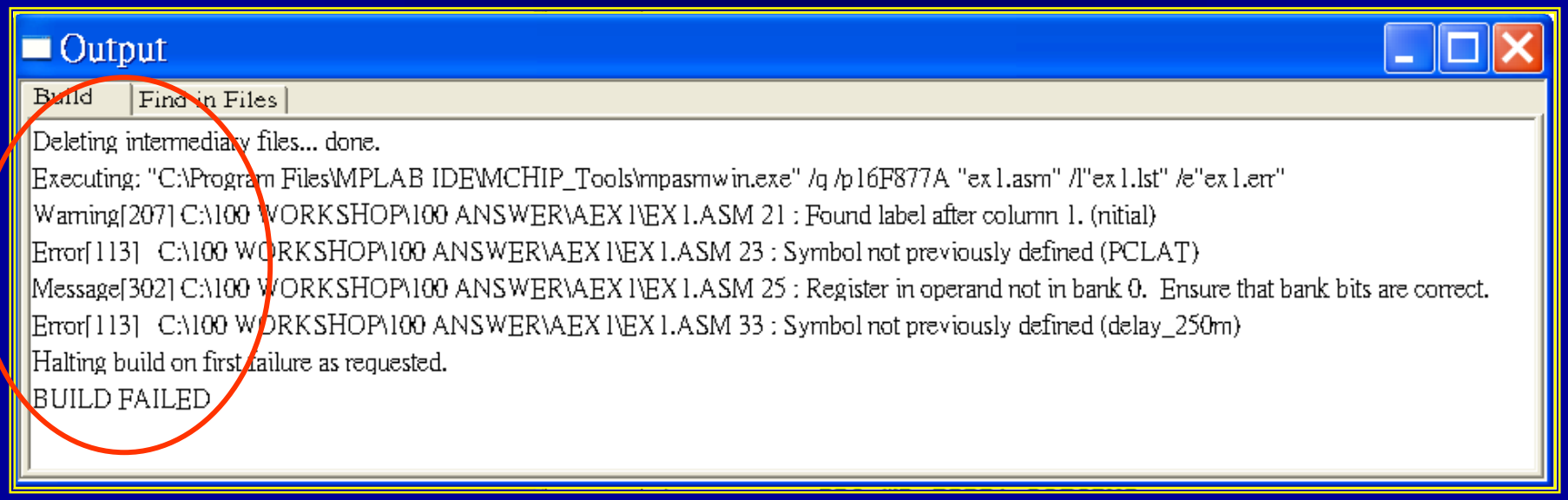
- MPLINK 能作做什麼 ?
  - 安排實際位址給程式(CODE)及資料(RAM)
  - 產生可執行擋 (HEX)
  - 安排堆疊位址及深度給 MPLAB C18
  - 提供 COD 檔以利程式偵錯
  - 讓程式更容易模組化
  - 連結資料庫 (Library)

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
18F452.lkr
linker script
output files
prog.err
prog.cod
prog.hex
prog.lst
prog.map

練習四：組譯原始程式

- 在 **Build Output** 視窗：
  - 什麼是 Make Project ?
  - 什麼是 Build All ?
  - 組譯的結果是 “Build Failed” 或 “ Build completed successfully” ?
  - 組譯的結果有 “Message”、 “Warning”、 “Error” 怎麼辦?
  - 組譯成功後，會有那些檔案產生 ?



指令的執行時間

```
; This is delay subroutine for 1ms
; Oscillator Frequency : 4MHz
; ** 若使用16MHZ的操作頻率時,速度則快4倍

VAL_US          EQU     .249
                call    delay_1ms       ; 2us
                :

delay_1ms:
                movlw   VAL_US          ; 1us

                movwf   count           ; 1us

dec_loop        nop                     ; 1us

                decfsz  count,f         ; count=0, 2us
                                        ; count>0, 1us
                goto    dec_loop ; 2us

                return                  ; 2us
```

該指令執行所需的時間

2us
+
1us
+
1us
+
(1us+1us+2us)*248
+ (1us+2us) = 995us
+
2us

1001us

練習五：寫個 Delay 的副程式

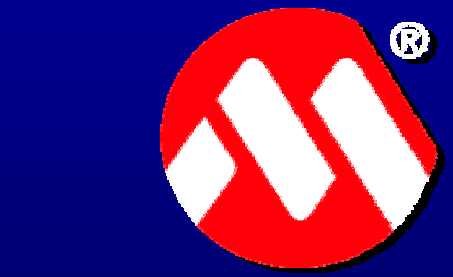
Ex5.ASM

❶ 利用前面提及的 “虛擬指令的使用範例” 來撰寫 Delay 的副程式

- Delay 0.5mS 的副程式
- Delay 10mS 的副程式 (利用 CALL Delay 0.5mS 的方式)
- Delay 200mS 的副程式 (利用 CALL Delay 10mS 的方式)
- 建立 Reset 向量起始位置及架構整個可執行的主程式

❷ 如何偵錯、測量 Delay 副程式 (使用軟體模擬)

- Reset , Run , Halt , Step , Step Over
- 設定軟體中斷點 , Watch Window 變數觀察
- Stopwatch Window , Files Registers Window (RAM)

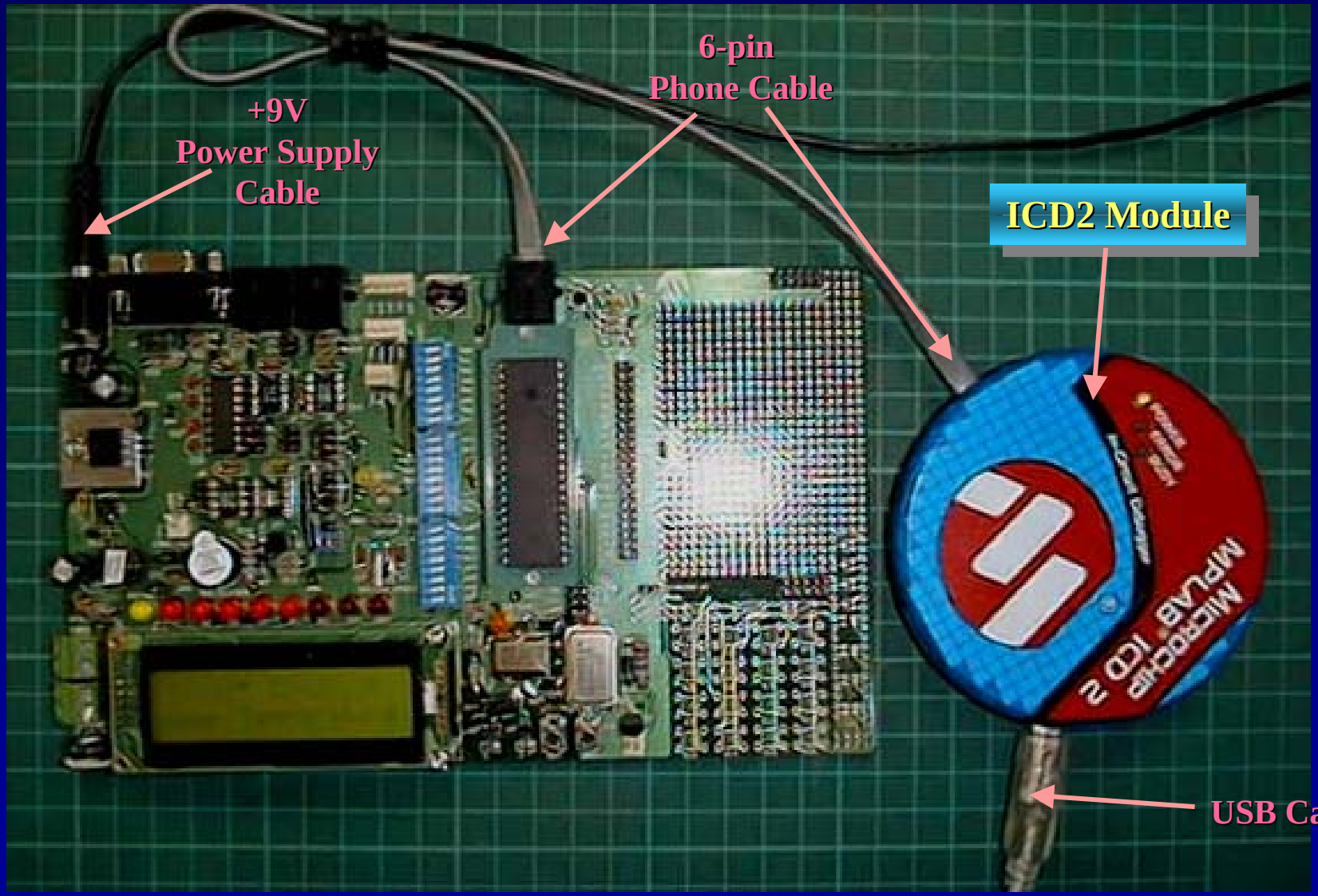


使用 MPLAB ICD2 除錯

MPLAB-ICD2 功能

- 全速執行
- 單步執行
- 單點硬體中斷
- 變數觀察，原始程式除錯等級
- 快速載入程式到模擬元件
- 可當模擬元件的燒錄工具
- 工作電壓 ： **2.5V to 5.5V**
- 頻率範圍 ； **32KHz to 20MHz**
- **RS-232** 或 **USB** 介面
- 價格便宜
- 直接使用在 **MPLAB-IDE v6.xx**

MPLAB-ICD2 配線圖



MPLAB-ICD2 注意事項

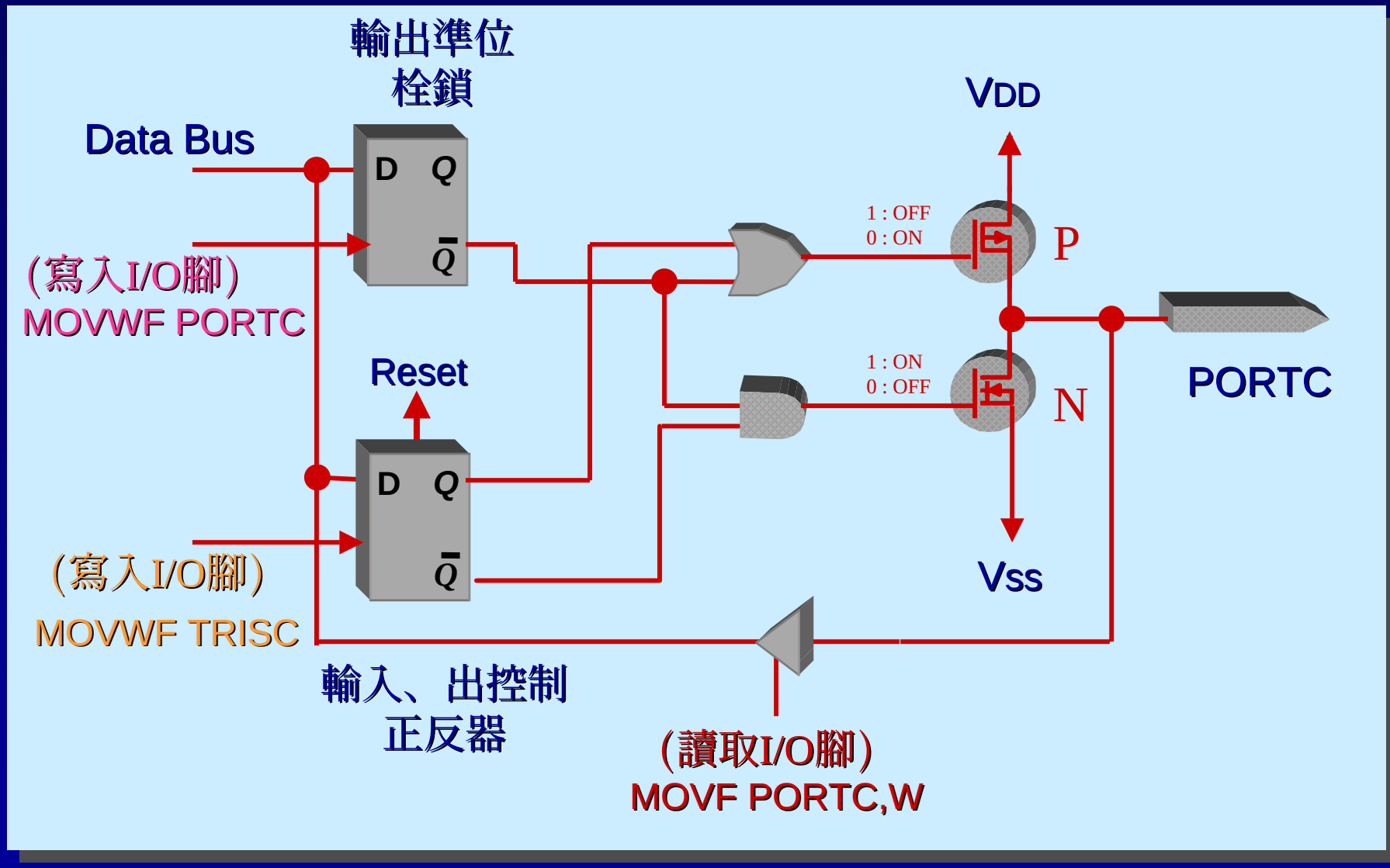
- ICD2 佔用一層堆疊, 使用者只可使用七層堆疊
- 程式位址 (0x1F00-0x1FFF) 保留給監督程式使用
- 程式位址 0x0 必須填入“NOP” 指令
- RB6 & RB7 保留給 ICD 做除錯用
- Data Memory 0x70 , 0x1EB – 0x1EF 等六個 RAM 將被佔用 !!
- MCLR pin 會出現 13V 的電壓 (thru a 1kohm resistor)

使用 MPLAB-ICD2

- 有關使用 ICD2 的詳細步驟，請參閱：
  - **MPLAB IDE v6.10 中文使用手冊，第五章**
- 使用 ICD2 時，務必重新載入目前所使用MPLAB IDE 版本的 ICD2 Firmware 以確保 ICD2 能與 MPLAB IDE 相容
  - 先點選 **“Debugger → Select Tool → MPLAB ICD2”**
  - 再點選 **“Debugger → Download ICD2 Operating System”**

基本 I/O 控制

基本 I/O 操作


輸出準位
栓鎖
Data Bus
D Q
Q
(寫入I/O腳)
MOVWF PORTC
Reset
D Q
Q
(寫入I/O腳)
MOVWF TRISC
輸入、出控制
正反器
(讀取I/O腳)
MOVF PORTC,W
VDD
1 : OFF
0 : ON
P
1 : ON
0 : OFF
N
Vss
PORTC

設定 I/O Pin

- 試著將 I/O (PORT) 視為一個暫存器 (RAM)來操作
- TRISx 為 I/O 輸出或輸入的控制暫存器
  - 檢查一下，PORTx 與 TRISx 所在的位址有何不同？
  - 要設定 PORTx 為輸出時，則將相對應的 TRISx 設定為 “0”
  - 要設定 PORTx 為輸入時，則將相對應的 TRISx 設定為 “1”

例 ： 將 PORTD 的 RD0-RD3為輸出腳， RD4-RD7為輸入腳
並使 RD0-RD3 送出 “1,0,1,0” 的準位。

```
clrf        PORTD
banksel     TRISD
movlw       B’11110000’
movwf       TRISD
banksel     PORTD
movlw       B’00000101’
movwf       PORTD
```

PORTD 直接驅動 LED

- PIC16F877A I/O Port 基本電器規格
  - IC V_{DD} pin 最大電流：250mA
  - IC V_{SS} pin 最大電流：300mA
  - 每個 I/O pin 最大趨動 / 吸入電流：25mA

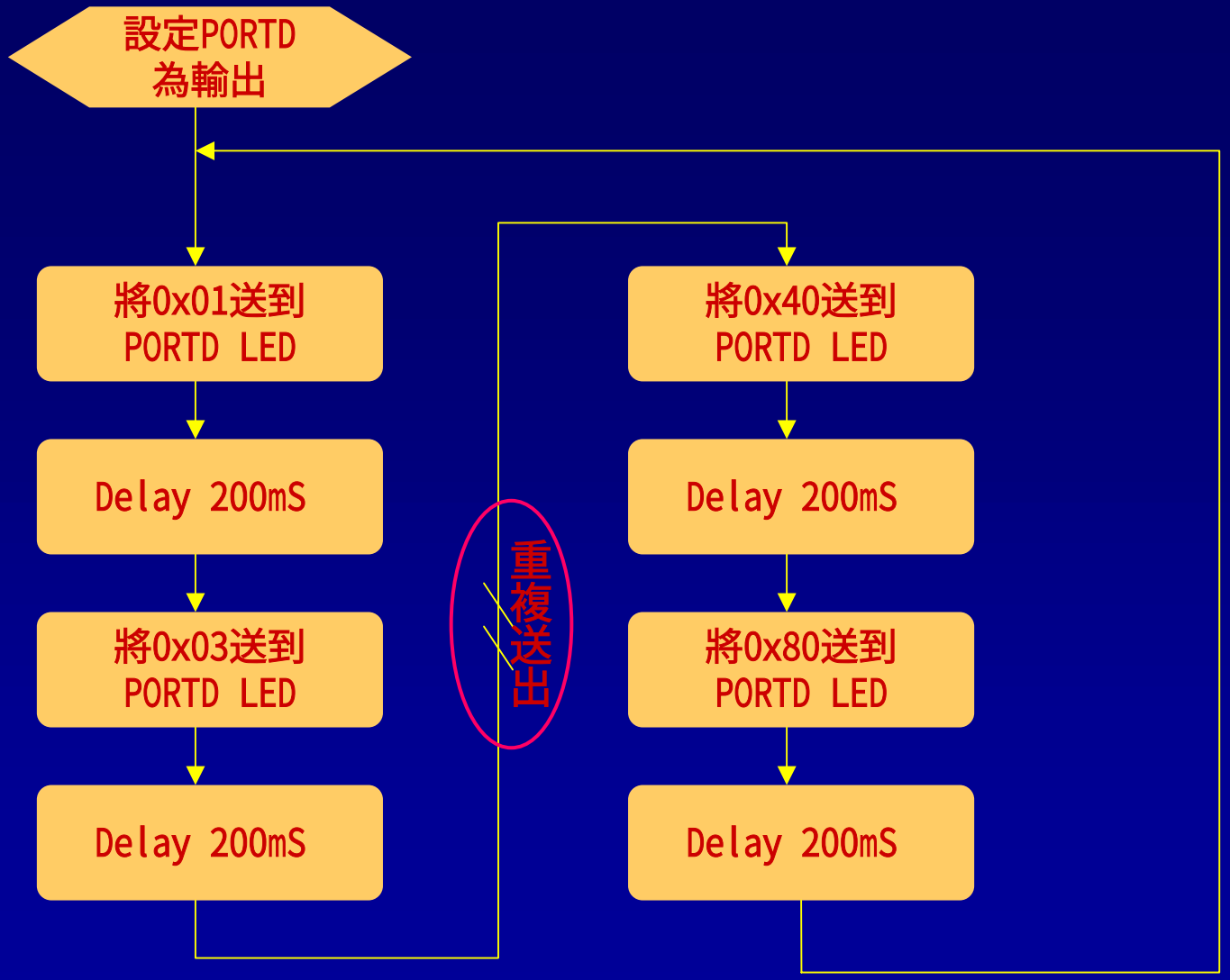
- 常見 I/O Port 的名詞解釋
  - 什麼是 V_{OL}
  - 什麼是 V_{OH}
  - 什麼是 V_{IL}
  - 什麼是 V_{IH}
  - 什麼是 I_{IL}
  - Min , Typical , Max 所代表的意義

練習六：基本 I/O Port 控制

Ex6.ASM

- 將練習五完成的 200mS 延遲副程式，作為延遲時間的基準
- 設計一個程式能將 PORTD 的 LED 從“零”開始每隔 200mS 自動加一
- 利用 ICD2 來進行除錯
  - 設中斷點、變數觀察視窗

LED 亮燈控制


設定PORTD
為輸出
將0x01送到
PORTD LED
Delay 200mS
將0x03送到
PORTD LED
Delay 200mS
重複送出
將0x40送到
PORTD LED
Delay 200mS
將0x80送到
PORTD LED
Delay 200mS

練習七：基本 跑馬燈控制

Ex7.ASM

- 以練習六為程式架構，仍以 200mS 延遲副程式作為控制跑馬燈的移位的時間基準
- 設計一個程式能將 PORTD 的 LED 從最左邊的(b0) 每隔 200mS 自動向右移一位
- 當LED亮到最後一位時(b7)，重新從b0開始
- 利用 ICD2 來進行除錯
  - 設中斷點、變數觀察視窗

了解旗號的意義與用法

旗號的功能

- 旗號的主要功用：
  - 在程式中用來判斷數值、變數的比較結果
  - 程式中的迴圈控制
  - 狀態的判斷以決定程式執行的路徑
  - 數學運算或數值轉換時有關進位、借位、半進位、有號數及溢位的處理
- PIC 的旗號：
  - C - 進位旗號
  - Z - 零旗號
  - DC - 半進位旗號

C 旗號的改變

- C - 進位旗號

➳ 執行加法時 (結果有進位時 C=1)

例一 : 假設 W reg. = 9A 時，執行 “ADDLW 0x80“ 後， C = ??

假設 W reg. = 3F 時，執行 “ADDLW 0x80“ 後， C = ??

➳ 執行減法時 (結果無借位時 C=1)

例二 : 假設 W reg. = 9A 時，執行 “SUBLW 0x80“ 後， C = ??

假設 W reg. = 3F 時，執行 “SUBLW 0x80“ 後， C = ??

(執行減法時，注意誰是被減數、誰是減數)

➳ 執行旋轉指令

例三 : 假設 Reg.30的內容= AA & C = 1 時，執行 “ RLF 0x30,W “ 後，
C = ?? , W Reg.的 bit0 = ??

假設 Reg.30的內容= AA & C = 1 時，執行 “ RRF 0x30,W “ 後，
C = ?? , W Reg.的 bit0 = ??

Z 旗號的改變

- 會影響 Z 旗號的指令共有十六個，一般而言只要會改變運算結果的指令均可能會影響 Z 旗號

  例如: 算數指令、邏輯指令、旋轉指令、清除指令
  MOVF, COMF, DECF, INCF

- 一般而言只要運算結果為零則 Z=1

  例 : 假設 W reg. = 55 時，執行 “ANDLW 0xAA“ 後， Z = ??
  假設 W reg. = 3F 時，執行 “SUBLW 0x3F“ 後， Z = ??

- 不會影響 Z 旗號的特殊指令

  例如: MOVWF, SWAPF , MOVLW

DC 旗號的改變

- 一般稱之為“半進位旗號”，發生的原因則是當做算數運算結果有發生較低的位元組 **(b3:b0)** 有進位產生時，則 **DC = 1**。
- 會影響 **DC** 旗號的指令
  - **ADDWF， SUBWF， ADDLW， SUBLW**
- 在普通情形下，很少會使用到半進位旗號，只有在做十進制的運算才會使用，利用**DC**旗號來決定其結果是否還需做加六的調整以符合十進制的結果。

  例 ： BCD 的加、減法

如何運用旗號

- 利用減法指令來做數值大、小的比較：
  - 若 F reg. 減 W reg. (SUBWF F,W)

C 旗號	Z 旗號	比較結果
1	0	F > W
0	0	F < W
1	1	F = W

  - 若需判斷 F > = W , 可直接測試 C = 1 即可 (Z 可忽略)
  - 若要判斷 F < = W , 就必須要先可測試 C = 0 , Z=1 兩條件同時成立

- 利用旋轉指令來改變 C 旗號， 可執行位元的檢測
- 超過 8 bit 的加、減法別忘記 C 旗號的意義

練習八：基本旗號控制

Ex8.ASM

- 將練習七的程式改寫：
  - 將 **PORTD LED** 顯示方式改用旋轉指令，並檢查進位旗號是否已被設定，來決定程式是否重新再執行一次
  - **Delay 200mS** 不變
  - 相同的功能，程式是否變小了?

練習九：設計一霹靂燈

Ex9.ASM

- 設定 PORTA 的 RA4 為輸入腳 (按鍵)
  - 注意 “PORTA & PORTE” 初始設定為 A/D 輸入

- 將顯示值 “b’00000111” 輸出到 PORTD，若 RB0 = 1 (按鍵放開) 則每隔 0.1SEC 向左移一位 (一次移 8 個位元)，RB0 = 0 (按鍵按下) 則每隔 0.1SEC 向右移一位 。

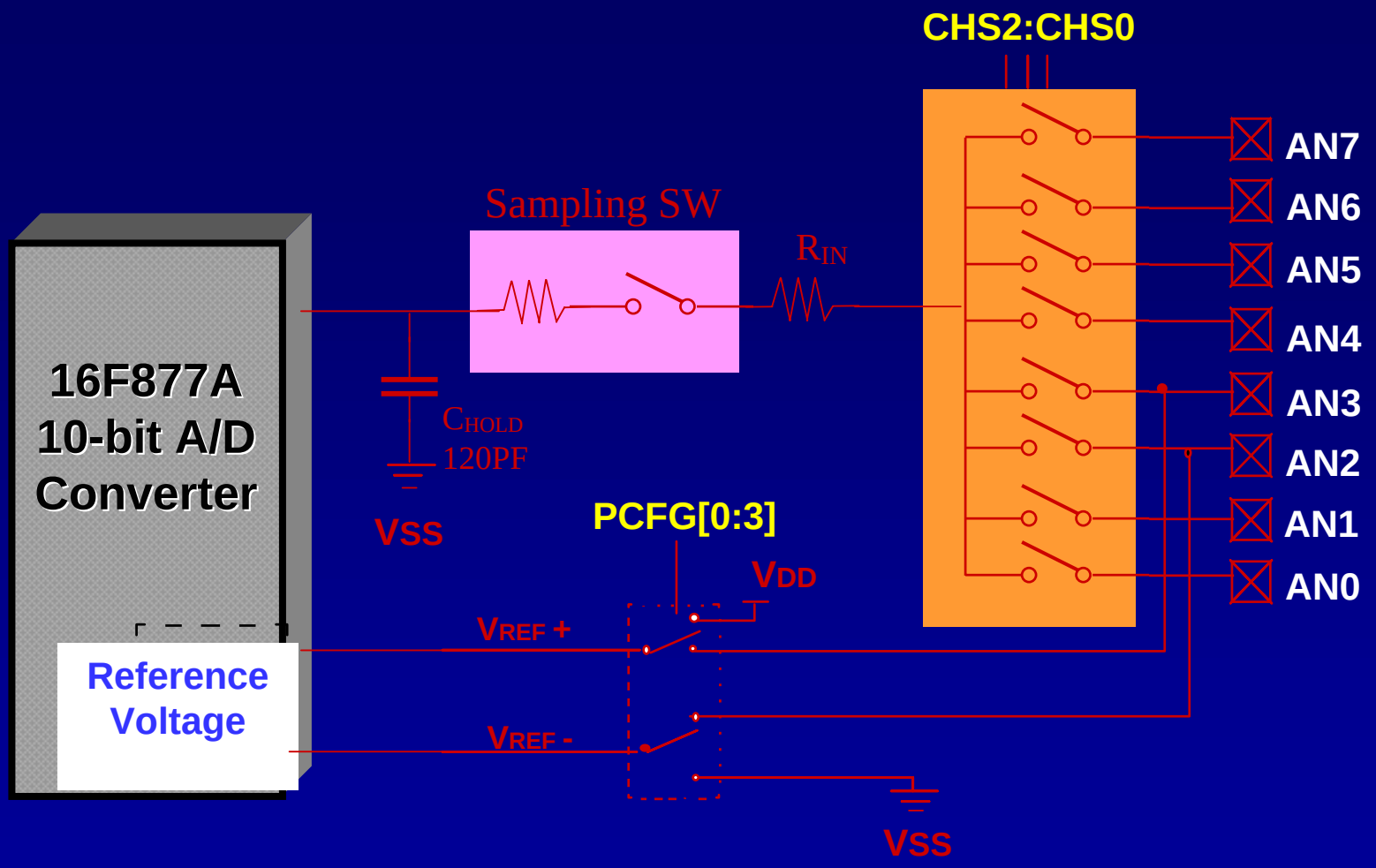
  注意 : Carry 旗號 是否受其它程式影響而消失 ?

速度可調整的霹靂燈設計

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 µS (輸入阻抗<10K)
- 類比輸入轉換時間：19.2 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 39.2 µs
  - 如輸入腳位固定，其轉換時間只需：29.2 µs

10-bit A/D 方塊圖


CHS2:CHS0
AN7
AN6
AN5
AN4
AN3
AN2
AN1
AN0
Sampling SW
RIN
16F877A
10-bit A/D
Converter
CHOLD
120PF
VSS
PCFG[0:3]
VDD
VREF +
VREF -
Reference
Voltage
VSS

See Data Book

A/D 控制暫存器

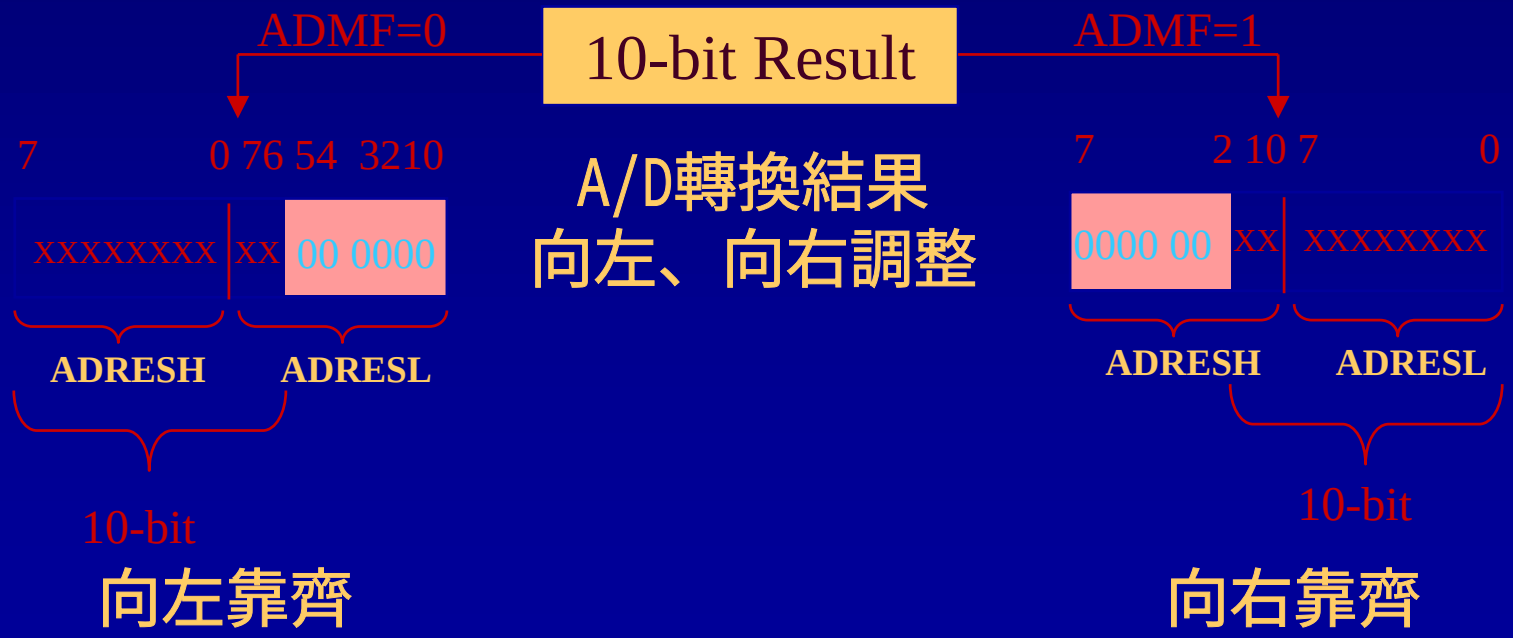
ADCON0 Register

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

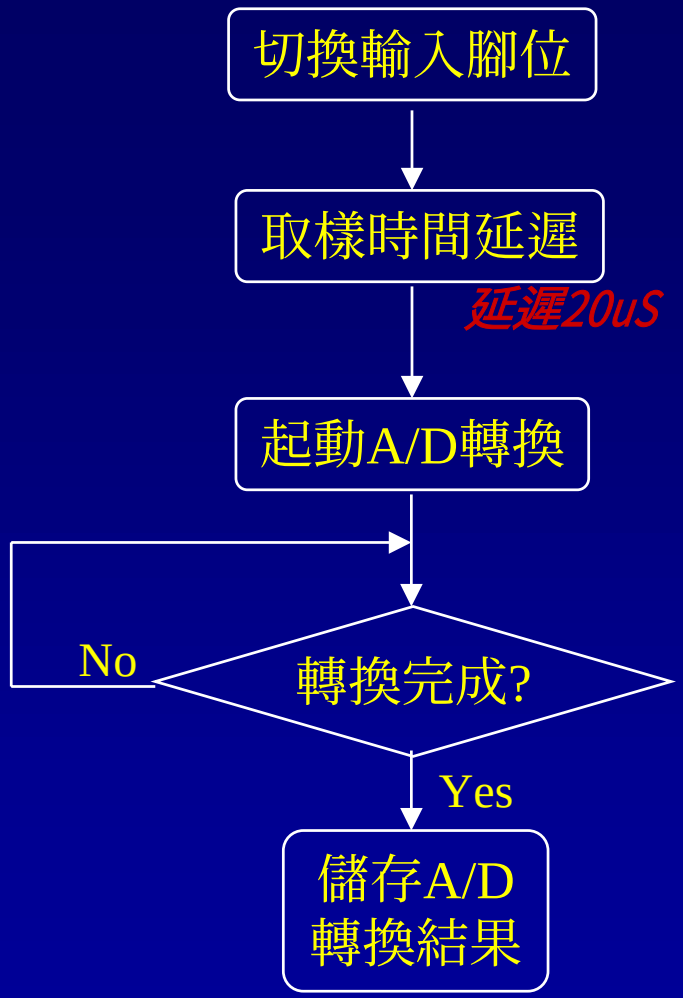
bit7 bit0

ADCON1 Register

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



A/D 轉換基本流程



A/D 轉換程式

```
main    banksel  TRISD
        clrf     TRISD           ; Set PORTD for LED Output Port
        movlw    b'00000111'     ; Disable the Analog Comparator
        movwf    CMCON
        movlw    b'00001110'     ; Select AN0 for the A/D input
        movwf    ADCON1
        banksel  ADCON0
        movlw    b'10000001'     ; Enable A/D converter module
        movwf    ADCON0
Loop    call     Convert
        movwf    PORTD           ; Put the A/D result on LED
        goto     Loop
;
Convert:
        call     Delay_20uS      ; Delay for sample hold
        bsf      ADCON0,GO       ; Start convert A/D
AD_Loop btfsc    ADCON0,GO       ; Completed?
        goto     AD_Loop         ; No, loop test.
        movf     ADRESH,W        ; Yes, save the A/D result to W reg.
        return
```

練習十：以 VR 來控制霹靂燈旋轉的速度 (Ex10.ASM)

進階題：是否可在加入 VR (可變電阻)來控制跑馬燈移動的速度？

- 按鍵仍控制 LED 左、右旋轉
- VR 接在 A/D的 CH0 (RA0)
- 注意 BANK 的切換

- 注意：先將 PORTA 的比較器關閉
  - CMCOM [CM2,CM1,CM0]=111