
PLANET MICROCHIP
MPLAB-IDE
RTC
101 ASP
Microchip Technology Taiwan
Slide 1
Workshop 100
MICROCHIP

歡迎參加 101ASP 教育訓練

- 使用軟體工具
  - **MPLAB IDE v8.00 (或更新版本)**
  - **MPASM , MPLINK , MPLIB**
- 使用硬體工具
  - **MPLAB ICD2**
  - **Microchip APP001 Workshop Board (PIC16F877A inside)**
- 參考資料
  - **Microchip PIC16F877A Data Sheet**
  - **MPASM User's Guide with MPLINK and MPLIB**
  - **APP001 Workshop Board 電路圖**

Workshop 101ASP 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 101ASP 的內容

1. PIC16Fxxx Mid-Range 架構
2. 開發工具介紹
3. 使用 MPLAB IDE v8.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

PIC16Fxxx 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**

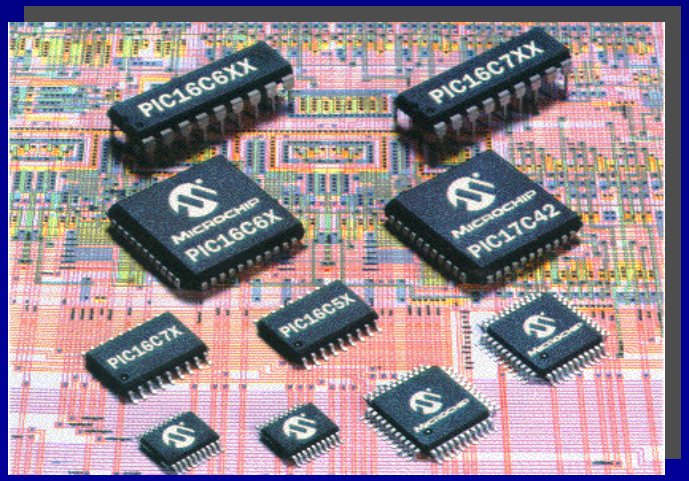
使用 MCU 的優缺點

- 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

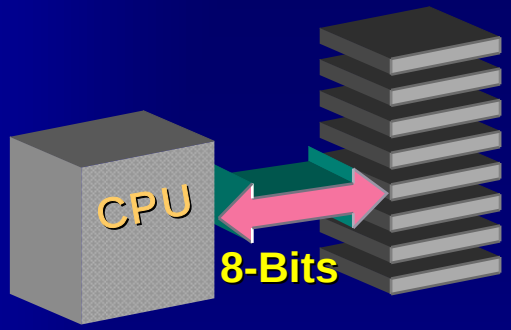
- 俱備 **RISC Microcontroller** 的特點
- 高效能的 **PICmicro** 具備 **RISC Microcontroller** 的特點，其主要的優點如下：
  - 使用 **Harvard** 管線式架構
  - 大部分指令均能在單一週期值內執行完成
  - 支援指令預提取功能
  - 所有指令爲 **“Single Word”**
  - **Long Word** 的指令編碼
  - 精簡指令集，重複指令極少
    不易造成混淆



PICmicro 的架構比較

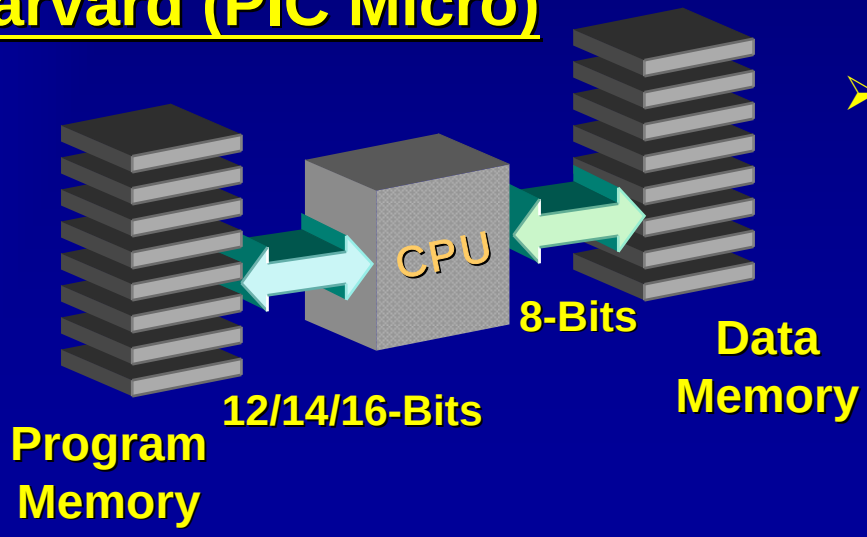
PIC 使用 Harvard Architecture

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)



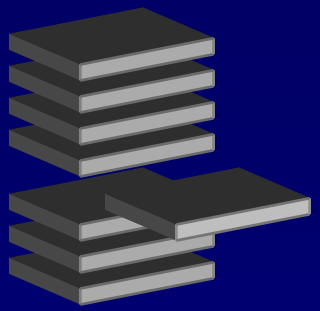
- 使用兩個不同的空間與管線來存取程式和資料
  - 增加處理資料的效能
  - 使得MCU可以具有不同寬度的程式記憶體與資料記憶體

PICmicro Architecture
長的指令編碼（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** 指令寬度爲 **12， 14 or 16-bits**，稱爲一個 **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 **PIC16Fxxx** 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

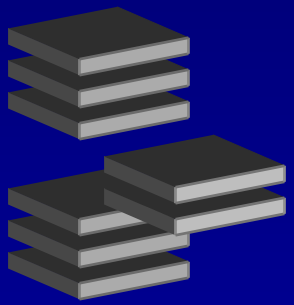
Long Word Instruction (con't)



PICmicro

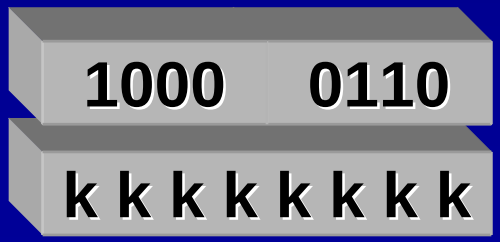
```
movlw     #imm<8>
```

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

PIC 架構
Pipelining

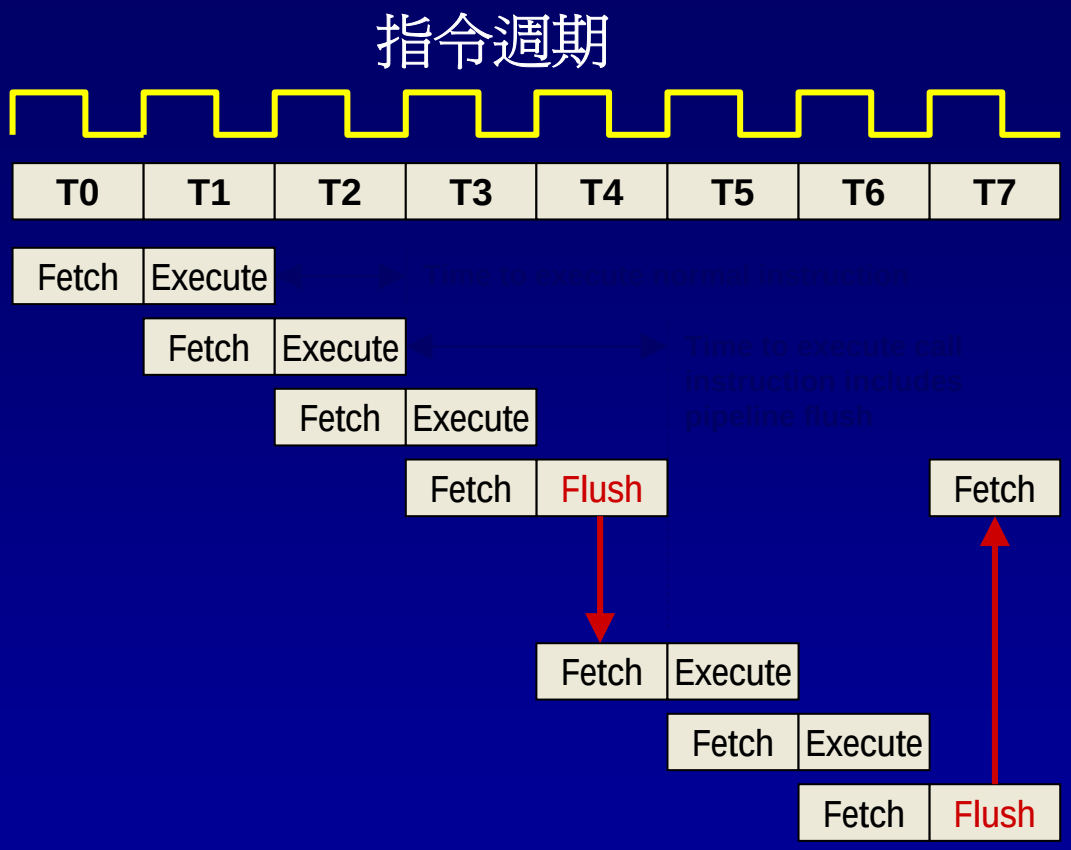
- 對大部份的MCU而言, 指令的提取與執行是連續發生但其每一個指令周期只有一個動作發生
- PICmicro 使用 Harvard 結構且使用流水管(Pipeline)的運作模式
- Pipeline 讓指令的提取與執行可同時進行.
- 指令的執行時間只需一個周期
- 程式分支的有關指令 (例如: GOTO, CALL 或 Write to PC) 則需要兩個指令周期

Instruction Pipelining

- 指令的提取與執行是同時發生的

範例程式

```
1   MAIN  movlw   0x05
2         movwf   REG1
3         call    SUB1
4         addwf   REG2
          :
51  SUB1  movf    PORTB,w
52        return
53  SUB2  movf    PORTC,w
54        return
```



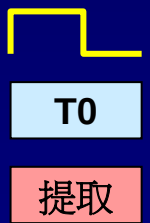
Instruction Pipelining

指令預提取

執行指令

movlw 0x05	-

指令週期



範例程式

```
 1  MAIN  movlw  0x05
 2        movwf  REG1
 3        call   SUB1
 4        addwf  REG2
          :
51  SUB1  movf   PORTB,w
52        return
53  SUB2  movf   PORTC,w
54        return
```

Instruction Pipelining

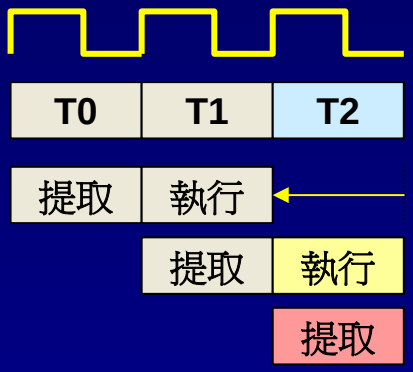
指令預提取

`call SUB1`

執行指令

`movwf REG1`

指令週期



範例程式

```
 1   MAIN  movlw   0x05
 2         movwf   REG1
 3         call    SUB1
 4         addwf   REG2
           .
           .
           .
51   SUB1  movf    PORTB,w
52         return
53   SUB2  movf    PORTC,w
54         return
```

Instruction Pipelining

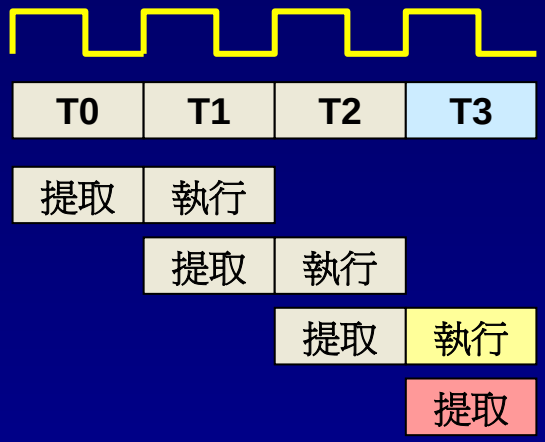
指令預提取

執行指令

`addwf REG2`

`call SUB1`

指令週期



範例程式

```
 1  MAIN  movlw  0x05
 2        movwf  REG1
 3        call   SUB1
 4        addwf  REG2
          :
51  SUB1  movf   PORTB,w
52        return
53  SUB2  movf   PORTC,w
54        return
```

Instruction Pipelining

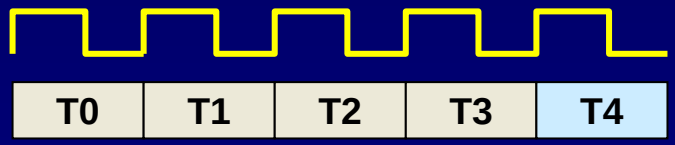
指令預提取

`movf PORTB,w`

執行指令

`call SUB1`

指令週期



執行 CALL 指令，路徑將改變所做的欲提指令動作將會無效

範例程式

```
 1  MAIN  movlw  0x05
 2        movwf  REG1
 3        call   SUB1
 4        addwf  REG2
        :
51  SUB1  movf   PORTB,w
52        return
53  SUB2  movf   PORTC,w
54        return
```

Instruction Pipelining

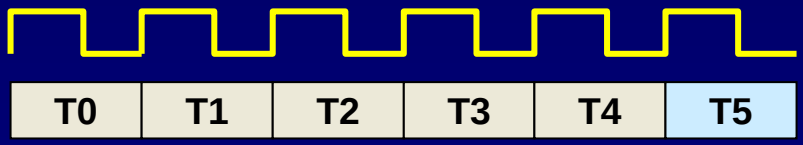
指令預提取

`return`

執行指令

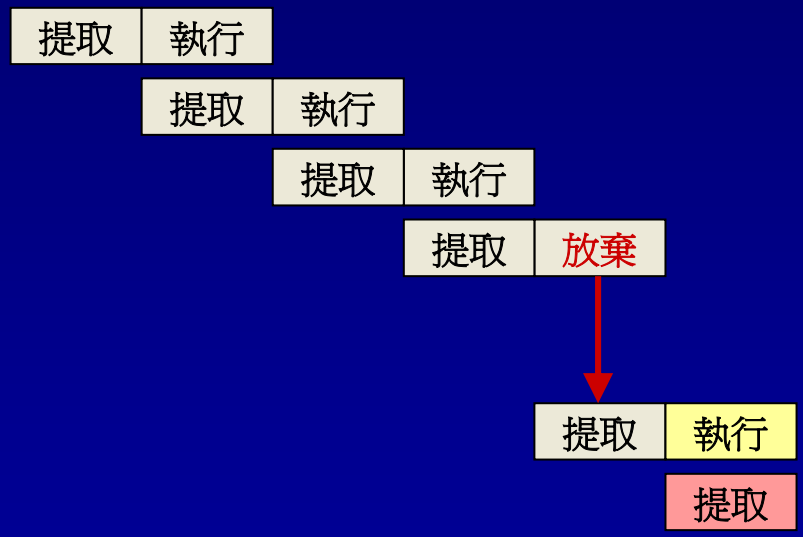
`movf PORTB,w`

指令週期



範例程式

```
 1  MAIN  movlw  0x05
 2        movwf  REG1
 3        call   SUB1
 4        addwf  REG2
          :
51  SUB1  movf   PORTB,w
52        return
53  SUB2  movf   PORTC,w
54        return
```



Instruction Pipelining

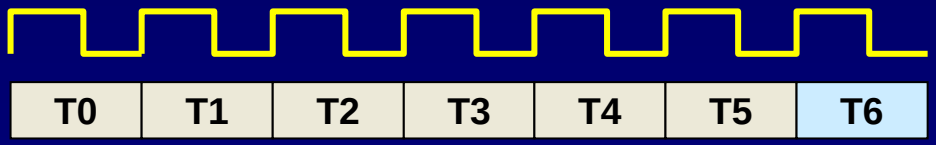
指令預提取

`movf PORTC,w`

執行指令

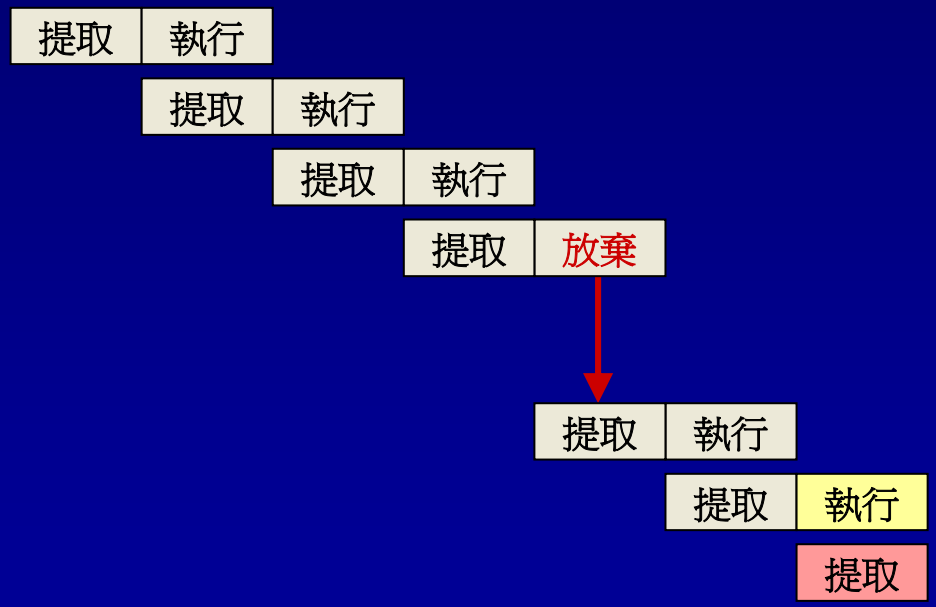
`return`

指令週期



範例程式

```
 1  MAIN  movlw  0x05
 2        movwf  REG1
 3        call   SUB1
 4        addwf  REG2
          :
51  SUB1  movf   PORTB,w
52        return
53  SUB2  movf   PORTC,w
54        return
```



Instruction Pipelining

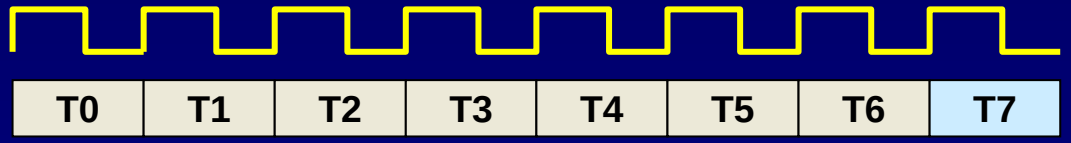
指令預提取

執行指令

`addwf REG2`

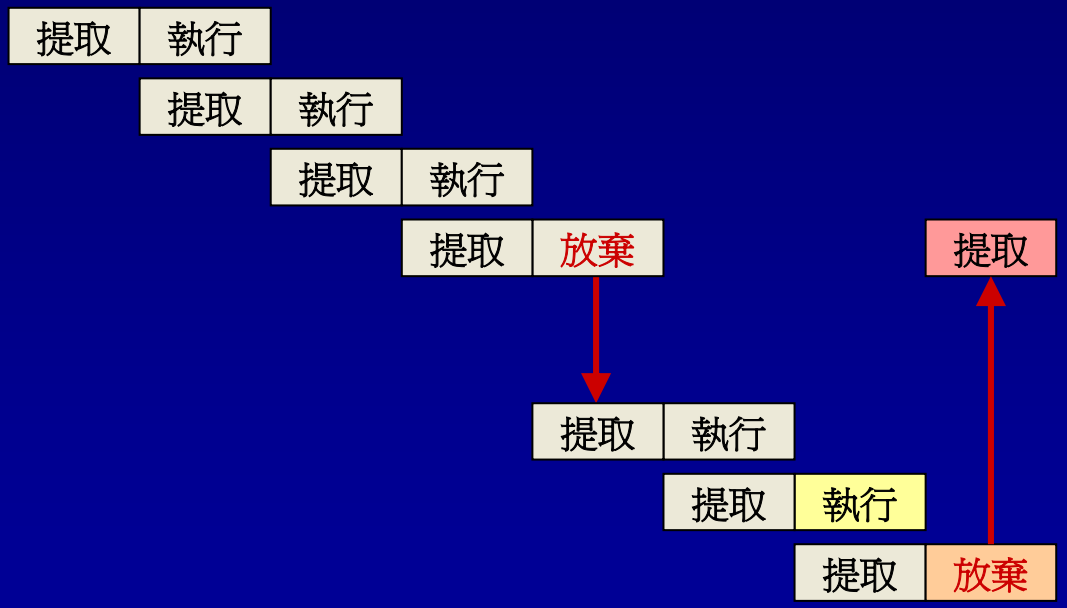
`return`

指令週期



範例程式

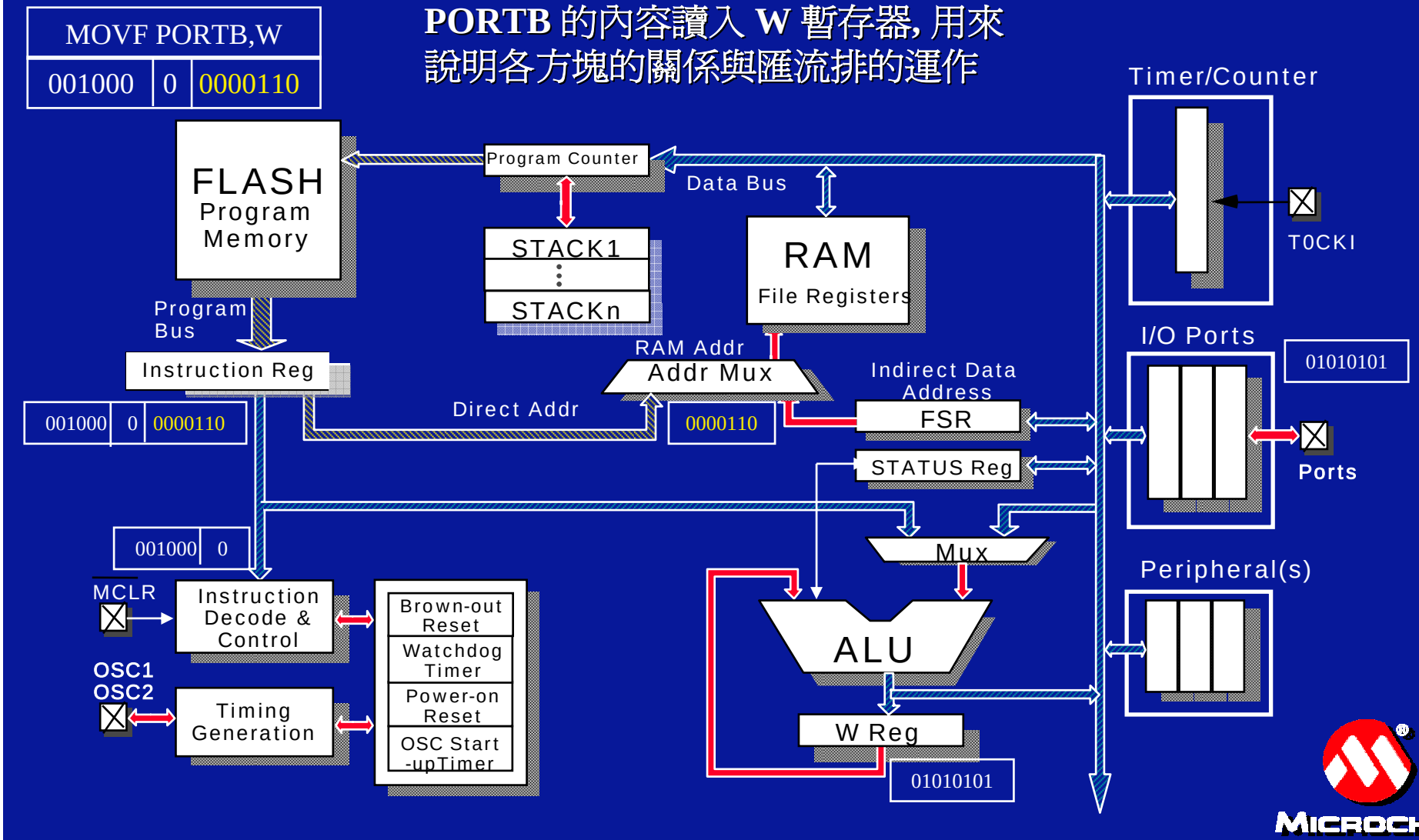
```
 1  MAIN  movlw  0x05
 2        movwf  REG1
 3        call   SUB1
 4        addwf  REG2
          :
51  SUB1  movf   PORTB,w
52        return
53  SUB2  movf   PORTC,w
54        return
```



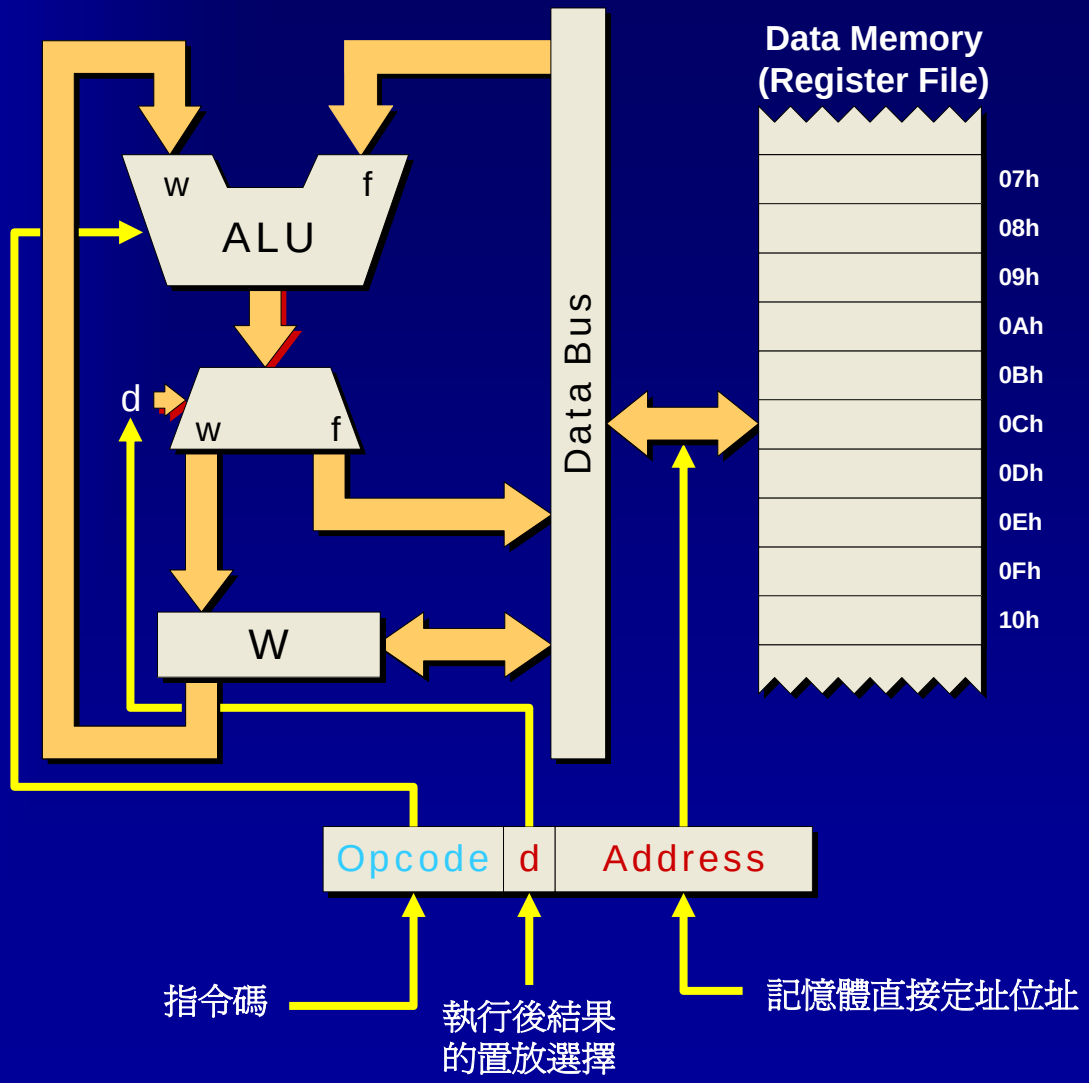
PICmicro Architecture

PIC16Fxxx 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



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) 所組成
- 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

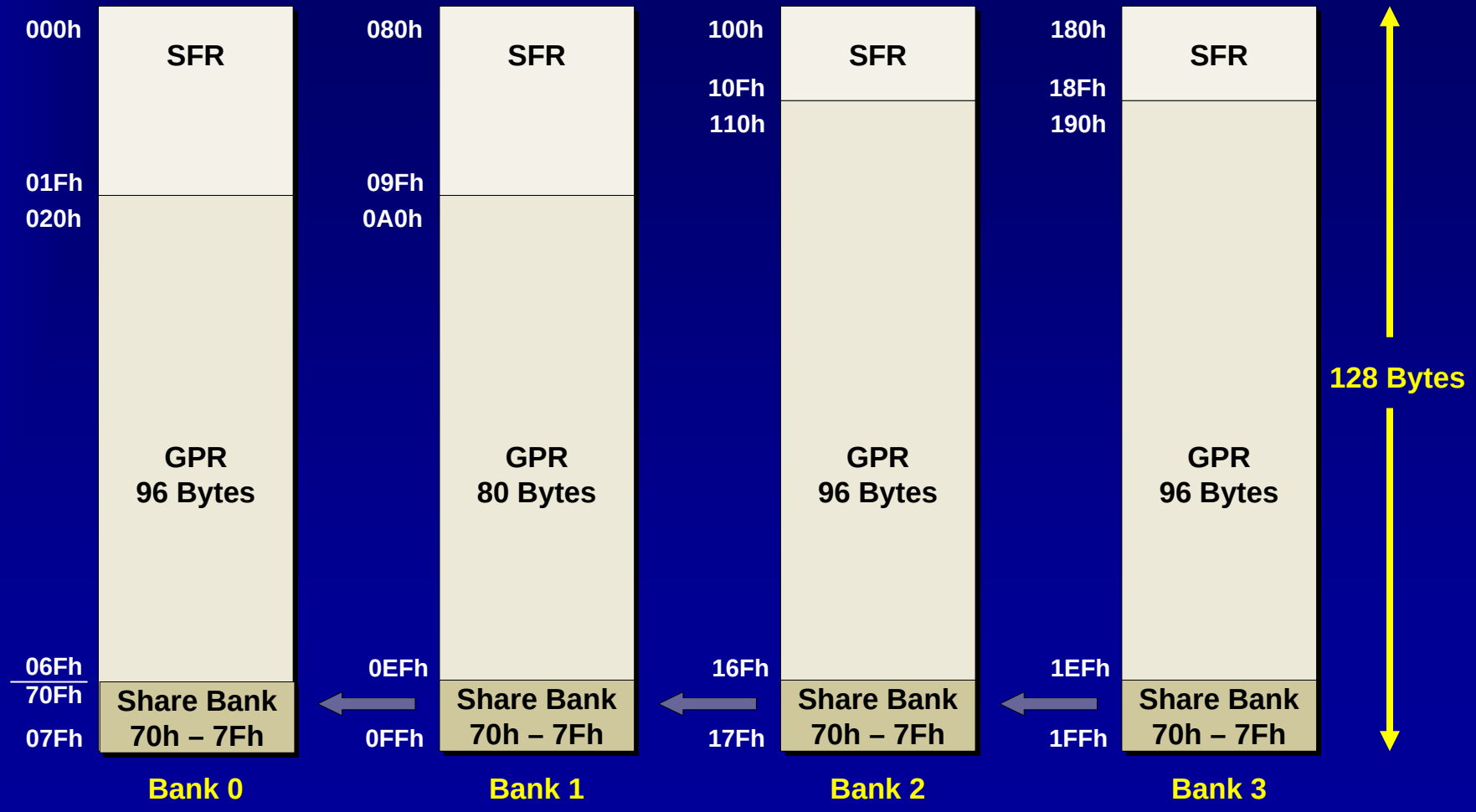
RAM 的分類

- **SFR : Special Function Register** 特殊功能暫存器
  - 泛指一般的周邊暫存器及核心暫存器
- 一般使用的 **RAM**
- 共用 **RAM : Share Bank RAM**
  - **Share RAM** 大小會因元件而異
  - **PIC16F887A** 有 **16 Bytes Shard RAM**，實際位址在 **0x70 ~ 0x7F**
  - **Bank1 0xF0 ~ 0xFF**，**Bank2 0x170 ~ 0x17F**，**Bank3 0x1F0 ~ 0x1FF** 等位置直接對應到 **Bank1 0x70 ~ 0x7F**
  - **Share RAM** 無須做 **Bank** 的切換，最適合中斷資料的處理與重要的變數儲存

Data Memory 架構

PIC16F876/877 Register File Map

368 Bytes of General Purpose RAM Plus Special Function Registers



SFR 特殊功能暫存器

	Bank 0		Bank 1		Bank 2		Bank 3
000	INDF	080	INDF	100	INDF	180	INDF
001	TMR0	081	OPTION_REG	101	TMR0	181	OPTION_REG
002	PCL	082	PCL	102	PCL	182	PCL
003	STATUS	083	STATUS	103	STATUS	183	STATUS
004	FSR	084	FSR	104	FSR	184	FSR
005	PORTA	085	TRISA	105		185	
006	PORTB	086	TRISB	106	PORTB	186	TRISB
007	PORTC	087	TRISC	107		187	
008	PORTD	088	TRISD	108		188	
009	PORTE	089	TRISE	109		189	
00A	PCLATH	08A	PCLATH	10A	PCLATH	18A	PCLATH
00B	INTCON	08B	INTCON	10B	INTCON	18B	INTCON
00C	PIR1	08C	PIE1	10C	EEDATA	18C	EECON1
00D	PIR2	08D	PIE2	10D	EEADR	18D	EECON2

Device Specific Registers

PIC16Fxxx Mid-Range
定址模式

PICmicro Architecture

程式記憶體：立即數定址

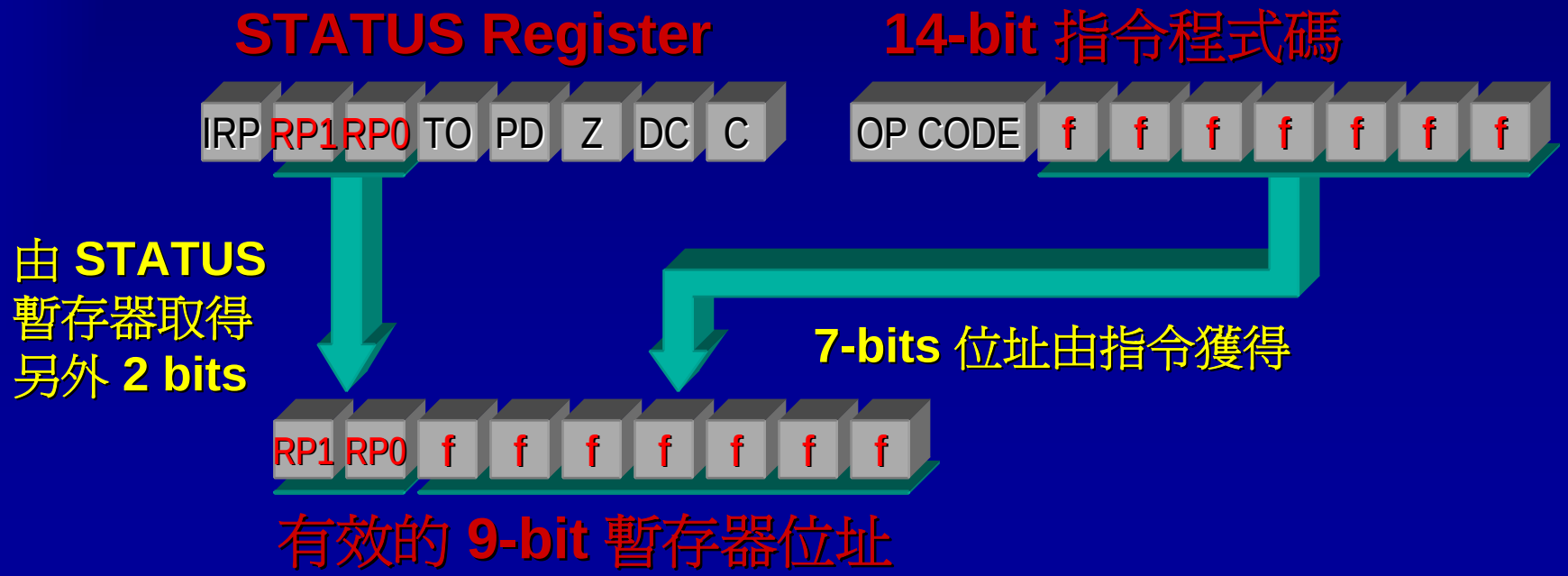
- 將 8-bit 的常數 (Literal) 直接置於指令的 bit-0 至 bit-7
- 配合不同的 OP Code 將常數做爲運算元
- 可直接將常數與 W 暫存器運算
- 使用於常數操作指令, 如 movlw, addlw, retlw,, 等

14-bit Instruction for Literal Instructions

PICmicro Architecture

資料記憶體：直接定址

- Mid-Range (14-bit 指令PIC) 的每一個暫存器庫爲 128 Bytes
- PIC16Fxxx 的最大DATA RAM大小爲 4 個 BANK (512 Bytes)
- 資料記憶體的有效位址爲 9 bits
- 7-bit 的位址直接來自於指令中
- 2-bit 由 STATUS 暫存器獲得，以定址到完整的 4 個 BANK



立即數定址與直接定址 – 範例

- 立即數定址：

```
MOVLW     h'80'   ; 將常數 0x80 載入 W 暫存器中
ADDLW     h'3F'   ; 將常數 0x3F 與 W 相加，其結果
                  ; 在放回 W 暫存器中
```

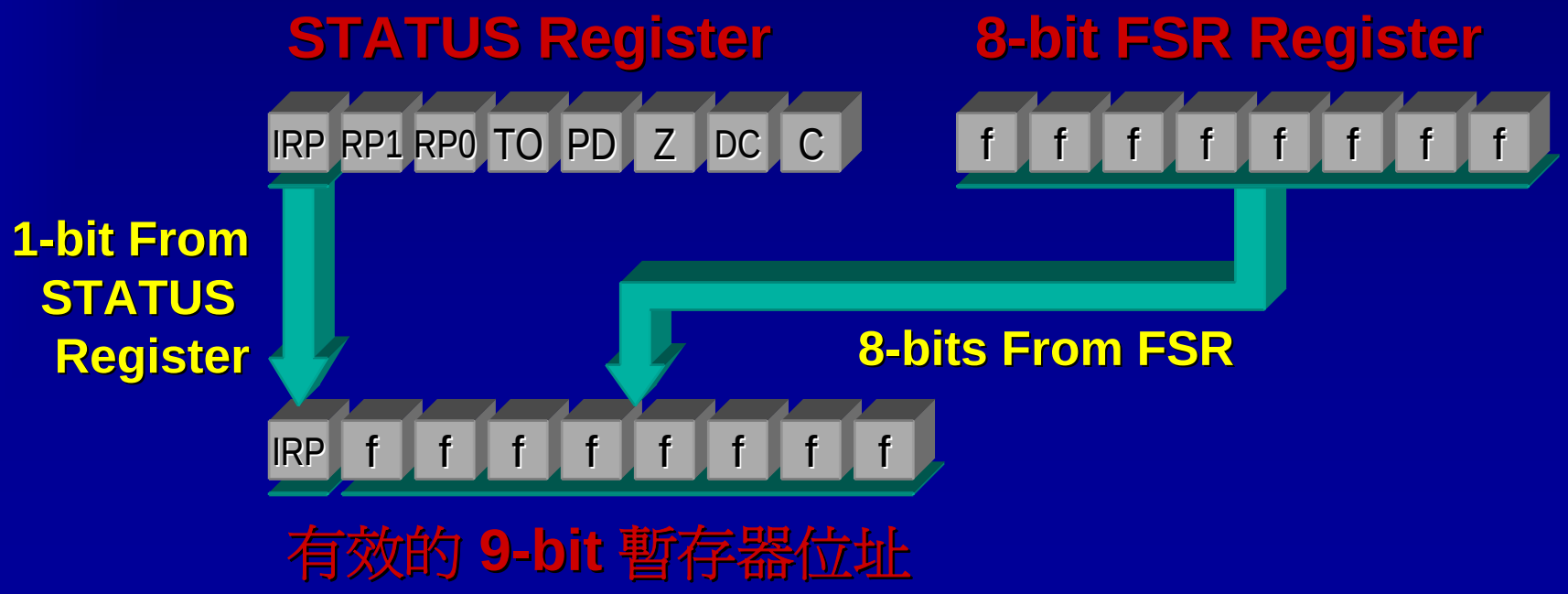
- 直接定址：

```
MOVWF     h'80    ; 將 W 暫存器中的內容值載
                  ; 入到位址 0x80 的 RAM 裡
MOVF      h'80',W ; 將 RAM 位址 0x80 的內容值載入
                  ; 到 W 暫存器中
```

PICmicro Architecture

資料記憶體：間接定址

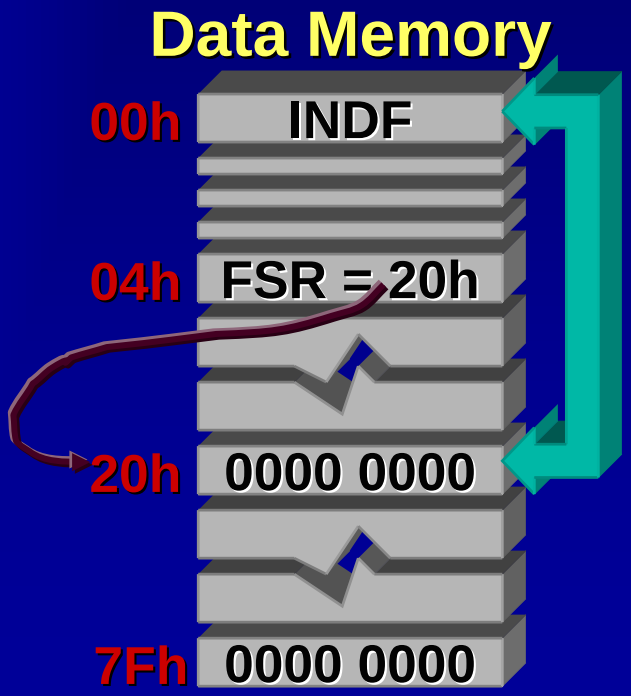
- Mid-Range (14-bit PIC) 的資料記憶體有效位址爲 9 bits，最大定址範圍是 512 Bytes (4 BANKs)
- 8-bit 的位址由 FSR (File Select Register) 獲得
- FSR 可提供 256 Bytes 的定址能力 (2 BANKs)
- 有效位址的第 9 位元由 STATUS 的 IRP bit 控制



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: 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，其基本得視野爲 2K 程式範圍；但程式範圍爲 8K (所以要考慮到 bit 12 & 13 的來源)
- 若範圍超過 2K 的程式頁範圍，需更新 PCLATH 的內容

```
MOVLW   (HIGH)  TARGET_ADDR   ; 取得 bit 12 & 13 位址
MOVWF   PCLATH                ; 更改 PCLATH 的值
CALL    TARGET_ADDR           ; 合併成 13 bits 的位址 (8k定址)
```

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” 可以協助設定 Program Page 及修改 PCLATH 暫存器
- 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：要前往的位址爲 TARGET_ADDR
  - 使用下列的敘述

```
PAGESEL     TARGET_ADDR
CALL        TARGET_ADDR
```

  - 相當於

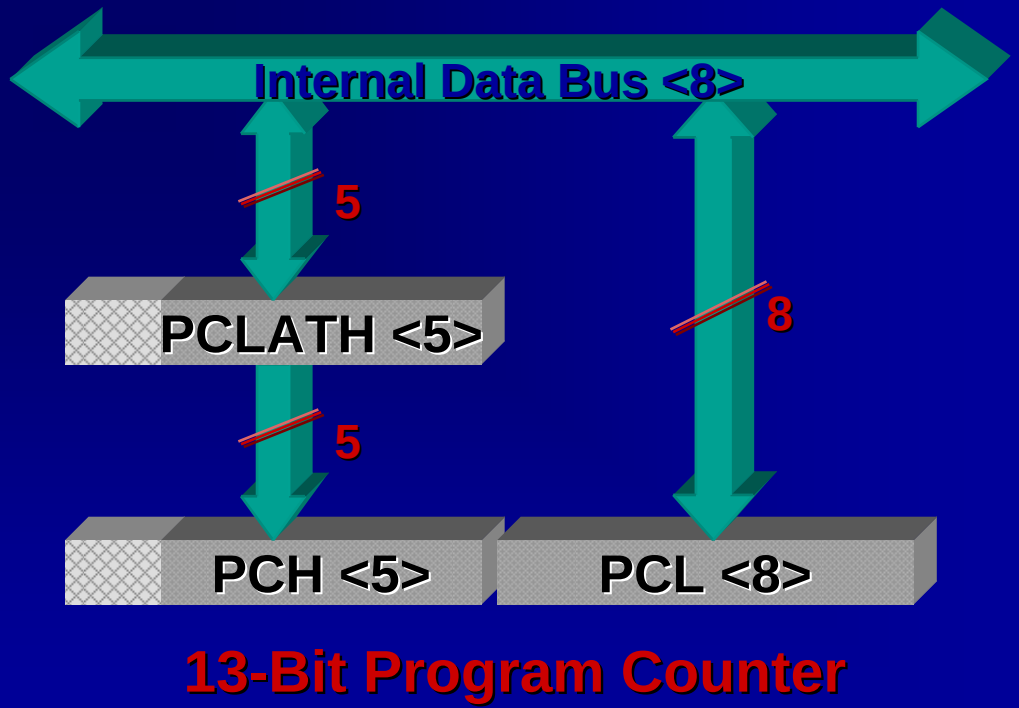
```
MOVLW       (HIGH)  TARGET_ADDR
MOVWF       PCLATH
CALL        TARGET_ADDR
```

PICmicro Architecture

程式記數器(PC) 的相對定址 (查表)

- 先將查表的最高位址(High Byte) 寫入 PCLATH.
- 再將 low byte 寫入 PCL，如此將會把整個13-Bit 的值一次載入到程式記數器 (PC)

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

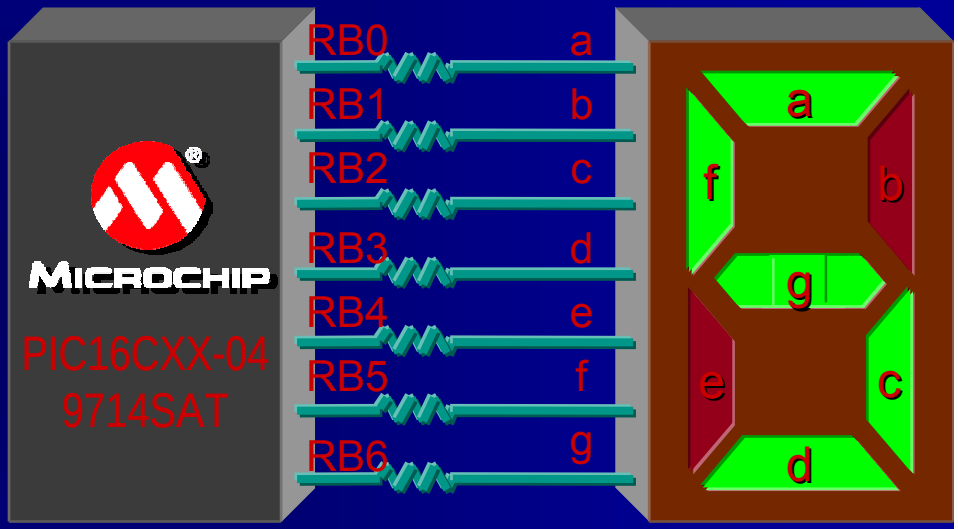


PICmicro Architecture

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
```



W Register

I/O Port B

PIC16Fxxx 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 MCU 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
- 有整體及各別周邊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 PIC
- 中斷延遲固定爲 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
```

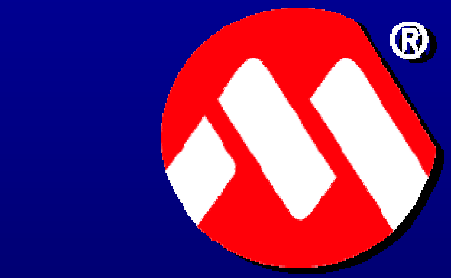
- MCS-8X51 @ 12 MHz
  - 每一指令需時1 to 4 cycle
  - 指令周期 = clk/12

```
                                ; 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	7
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

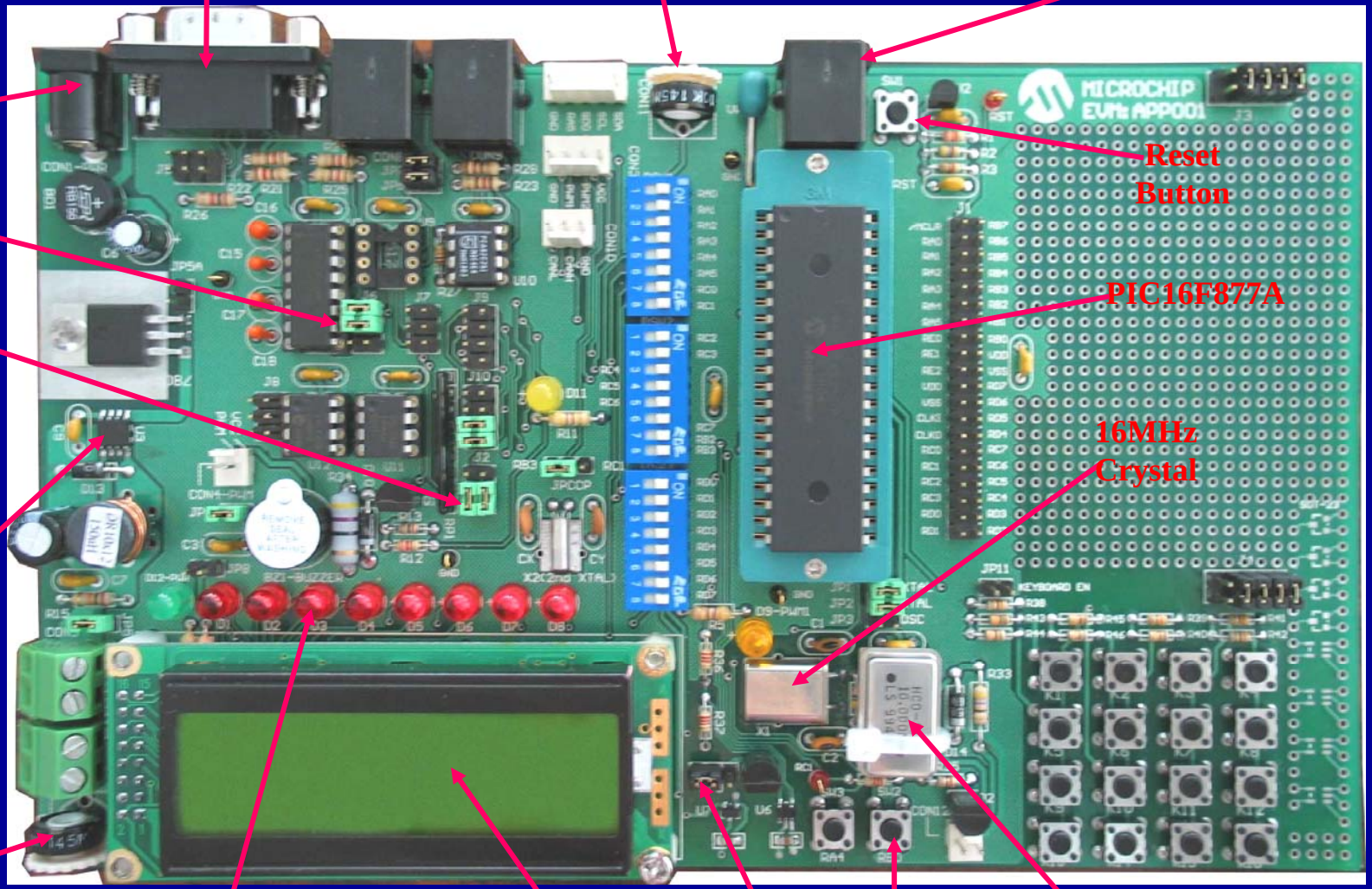
APP001 多功能實驗版

MPLAB IDE v8.xx

MPLAB ICD2, ICE2000

MPLAB Real ICE

APP001 v1.0 實驗板



APP001 實驗板功能

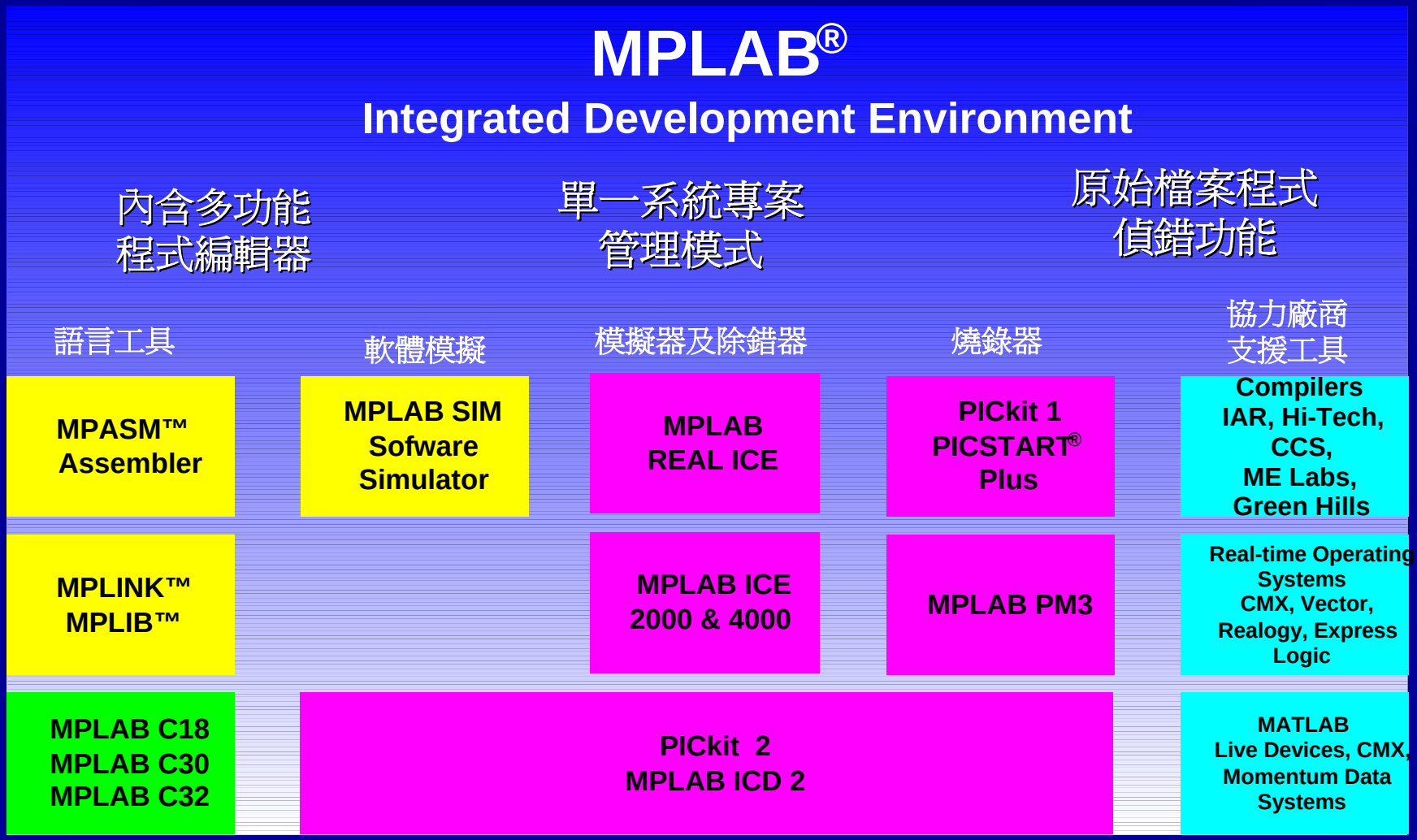
- 練習 **PIC16F877A** 內部周邊的使用方法及典型的應用方式
- **40 Pin** 的 **PIC16Fxxx**與 **PIC18Fxxx** 腳位相容
- 直接與 **MPLAB-ICD2** 連接 , 不須經由 **Header Board** 轉接
- 可練習的主要外接電路有:
  - **LCD Module ,** 按鍵處理，**I²C EEPROM**， **I²C**溫度存取
  - **SPI**，**RS-232 , RS-485 , CAN Controller**
  - **AD** 轉換 **, PWM** 與 **I/O** 控制
- **101ASP** 課程使用的主要資源
  - **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

- 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：
  - **ICE2000**
  - **Real ICE**
  - **MPLAB ICD2**
  - **MPLAB ICD2 LE**
  - **PICKit2 Debugger**
- 程式燒錄工具：
  - **PM-III**
  - **PICSTART Plus**
  - **MPLAB ICD2**
  - **PICKIT2 Debugger**

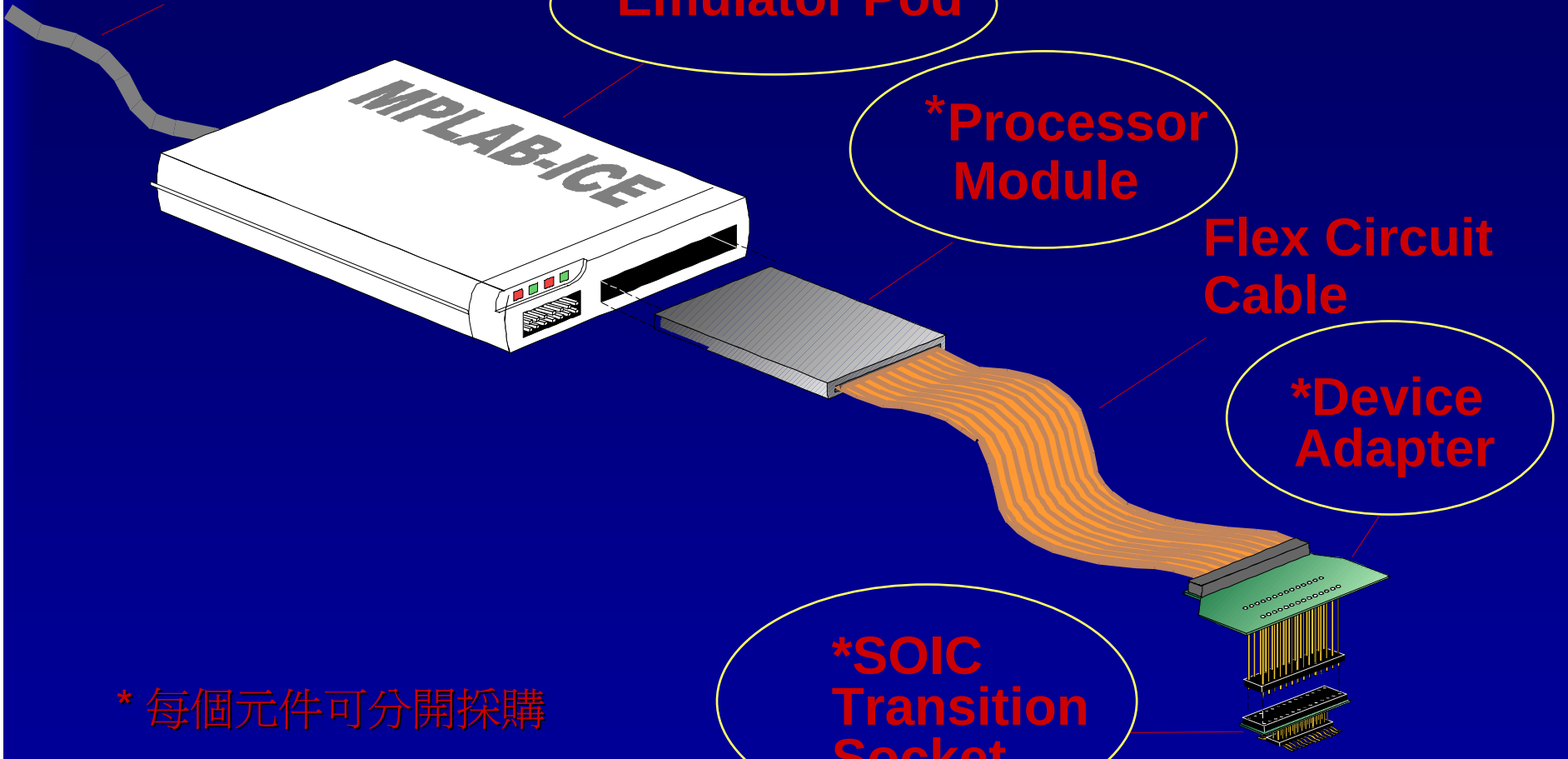
MPLAB *IDE* - 整合式的發展環境



Uniquely supporting 8, 16 and 32 bit MCUS within one integrated development Environment!

MPLAB - ICE2000

連接 Host 到 Pod 的Cable



*Emulator Pod

*Processor Module

Flex Circuit Cable

*Device Adapter

*SOIC Transition Socket

* 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

- 支援 **FS-USB** 介面
- 支援 **PIC16F** 裡具有 **ICD** 功能之 **MCU** 的除錯
- 支援 **PIC18F**， **PIC24**， **dsPIC30**，**dsPIC33 & PIC32**
- 可設定中斷點， 單步執行以及進行變數的觀察
- **32K Bytes (PIC18F4520)** 的程式可於 **3** 秒內燒錄完成

Real ICE 基本配備

傳統 ICE 技術上的瓶頸

- ICE 專用晶片價格昂貴
  - 量少，難以大量生產
  - 需與量產元件同步上市，困難度高
- 信號傳輸上時序與速度的誤差
  - CPU/記憶體 超過 20MHz 時的存取限制
  - ICE 實際的線路距離與元件之間的規格差異
- ICE 價格
  - 速度越快，價格越高
  - 價格昂貴，需要高速的外接硬體邏輯電路、 FPGA & SRAM

REAL ICE 功能

- “基本” 功能
  - 完全整合在 **MPLAB IDE** 的環境下
  - 開發環境下可作爲燒錄器
  - 中斷功能
  - 監看變數視窗功能
  - 程式執行控制

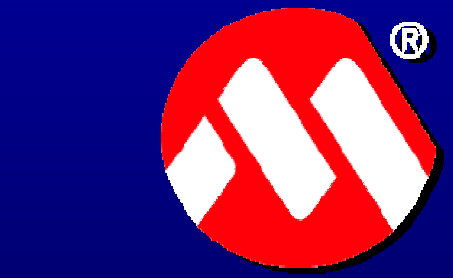
    - Run
    - Halt
    - Single step
    - Step into
    - Step over
    - Reset

REAL ICE 進階功能

- **Advanced Features**
  - **Breakpoints (軟體與硬體中斷設定)**
  - **Stopwatch (程式執行計時碼表)**
  - **Real time watch (即時變數觀測)**
  - **Trace (追蹤)**
    - **Program execution (追蹤程式執行)**
    - **Variable values (追蹤變數值)**

REAL ICE 功能一覽表

	PIC10F PIC12F PIC16F	PIC18F	dsPIC30F	PIC24F	PIC24H dsPIC33F
Run, Halt	√	√	√	√	√
Single Step	√	√	√	√	√
Hardware Breakpoints	1	1-3	1-4	1-6	1-6
Peripheral Freeze on Halt	√	√	√	√	√
Break on Data Fetch or Write		(√)	√	√	√
Break on Stack Error		√	-	-	-
Stopwatch		(√)	√	√	√
Pass Counter		√	√	√	√
Break address or data match		√	√	√	√
WDT Overflow		(√)	√	√	√
Real Time Watch		(√)	√	√	√
Software Breakpoints		1K	1K	1K	1K
Trace Data and Program Flow		√	√	√	√



使用 MPLAB IDE v8.xx

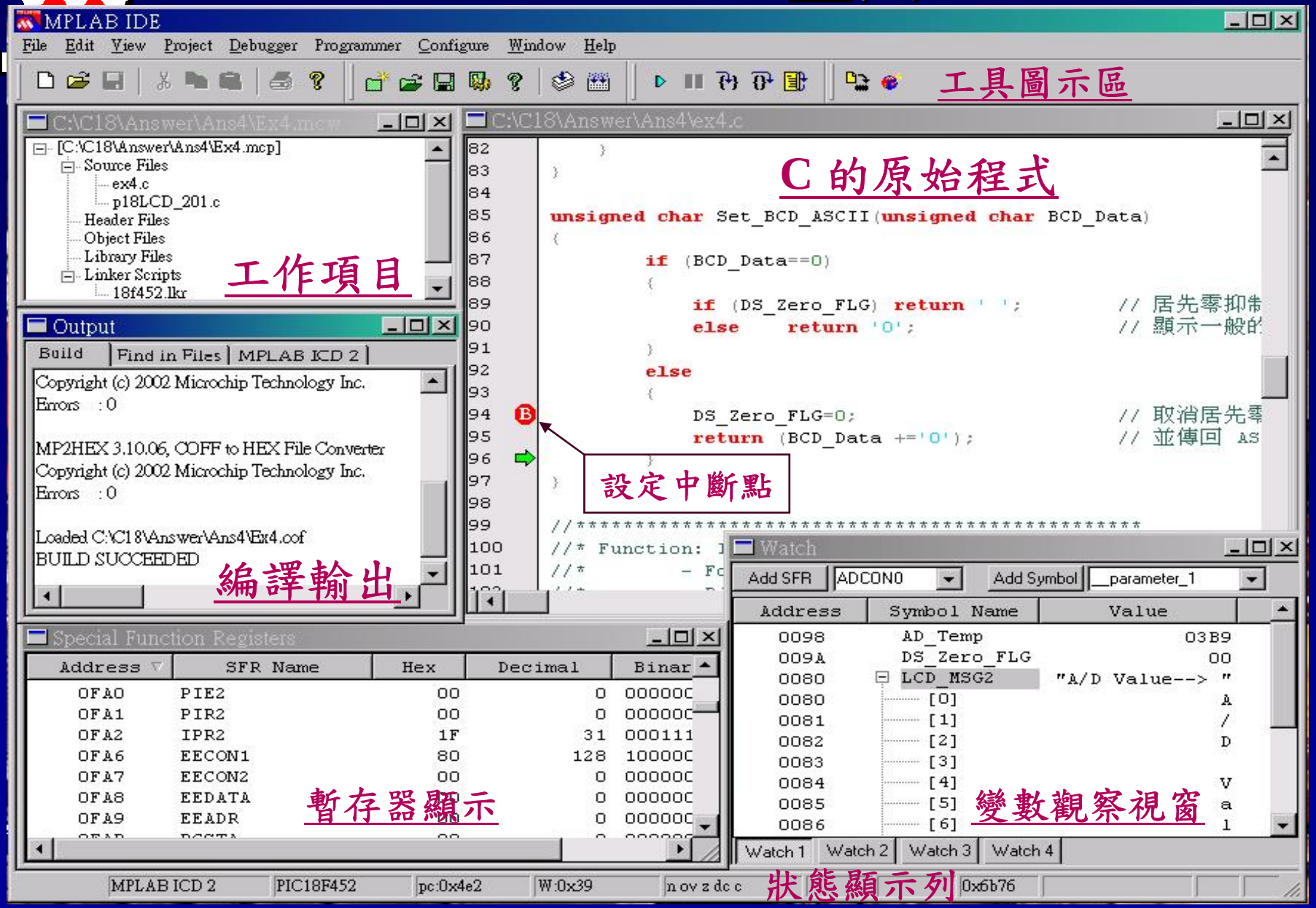
如何使用 MPLAB IDE v8.xx

- 詳細 MPLAB IDE 使用說明請參閱：
  - MPLAB IDE User's Guide
  - "www.microchip.com"
- ICD2 不知如何安裝 USB 驅動程式及使用：
  - 使用 ICD2 除錯時，要安裝 ICD2 驅動程式，其路徑如下：
  - C:\Program Files\Microchip\MPLAB IDE\ICD2\Drivers
    - 安裝方法請閱讀 clnicd2.htm & ddicd2.htm 檔案內之說明
  - 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 (MPASMWIN.exe) ：
  - 已內建在 MPLAB IDE 裡，安裝完成後就可使用 MPASM 組譯器
  - C:\Program Files\Microchip\MPASM Suite

安裝 MPLAB IDE

- 安裝 **MPLAB IDE** 注意事項
  - 之前已有安裝舊版的 **MPLAB IDE** ，需到控制台下移除舊版程式後再安裝新版本程式。切記！不可以直接使用檔案刪除的方式移除檔案。
  - 一般建議安裝在 **Microchip** 內定的路徑
  - 別忘了，還要安裝 **ICD2** 的 **USB** 驅動軟體
  - 如果有使用 **Real ICE** 還要再安裝驅動程式

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
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--> "
Watch 1
Watch 2
Watch 3
Watch 4
變數觀察視窗
Special Function Registers
Address
SFR Name
Hex
Decimal
0FA0 PIE2 00 0
0FA1 PIR2 00 0
0FA2 IPR2 1F 31
0FA6 EECON1 80 128
0FA7 EECON2 00 0
0FA8 EEDATA 0
0FA9 EEADR 0
暫存器顯示
MPLAB ICD 2
PIC18F452
pc:0x4e2
W:0x39
n ov z dc c
0x6b76
狀態顯示列

啓動 MPLAB IDE

- 有關 **MPLAB IDE** 的基本使用方式，請參考 **MPLAB IDE** 的使用手冊。
  - **MPLAB IDE User's Guide**
  - **http://ww1.microchip.com/downloads/en/DeviceDoc/51519B.pdf**
- 使用 **MPLAB IDE** 爲發展工具時，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。

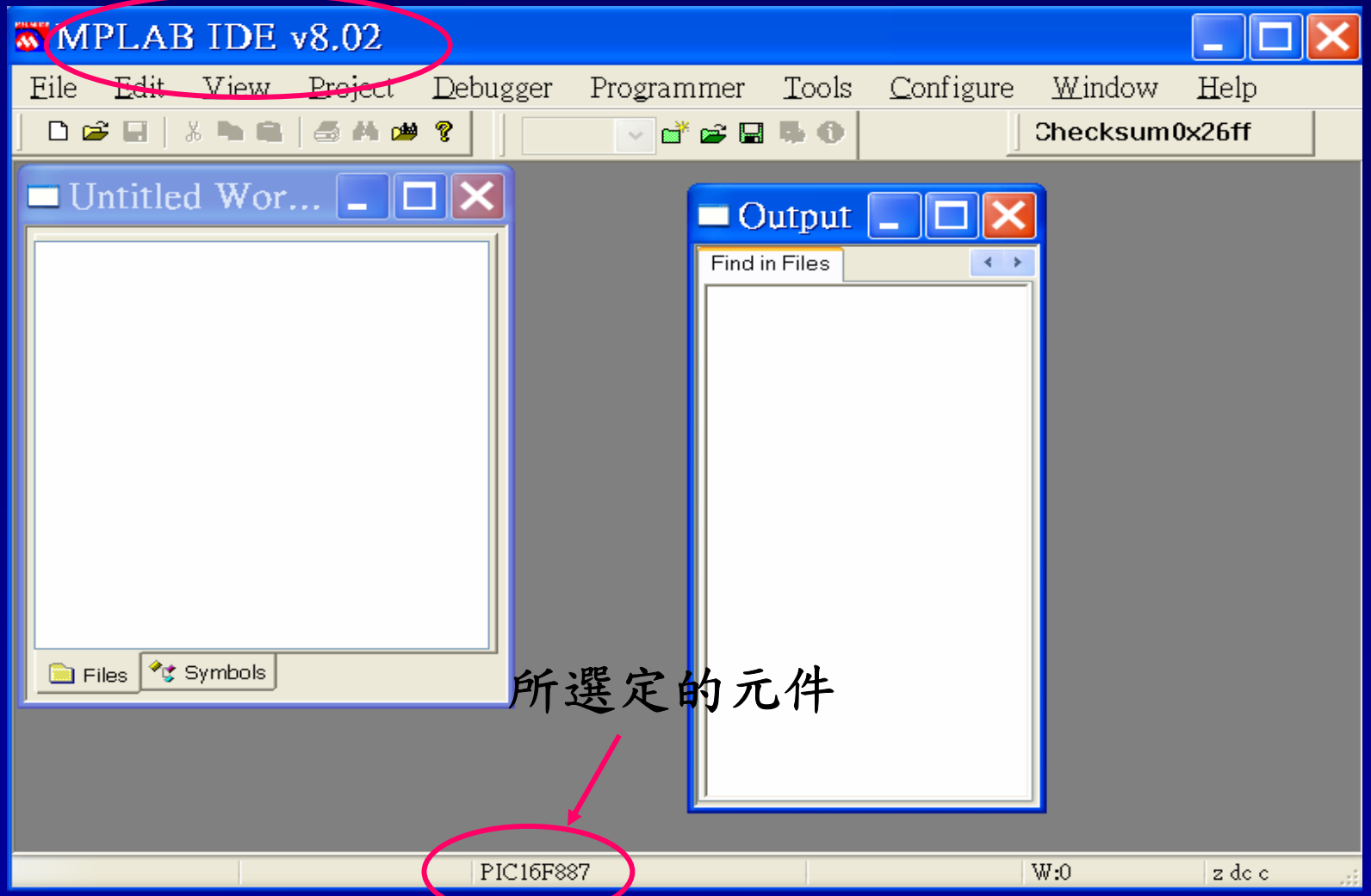
練習一

使用 MPLAB IDE
建立新的專案 (Project)

啓動 MPLAB® IDE

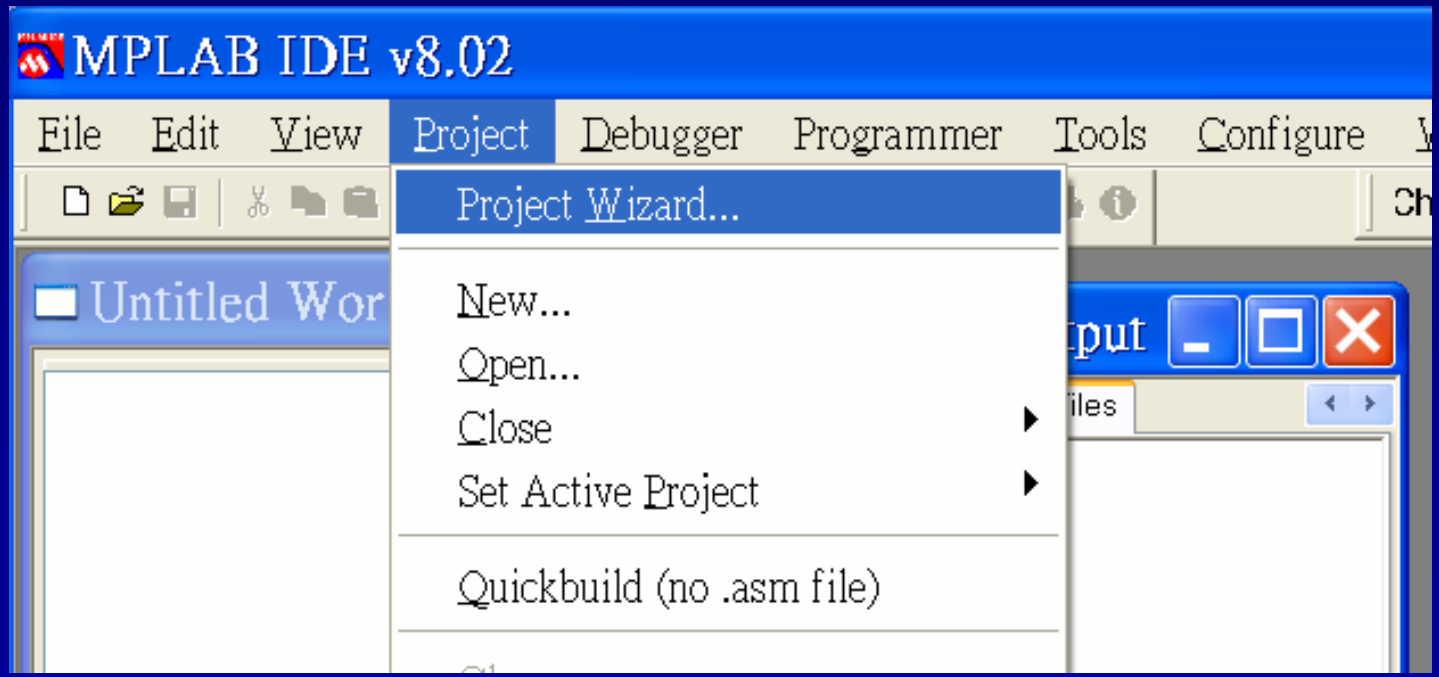
步驟 1

目前的版本



建立一個 New Project 步驟 2

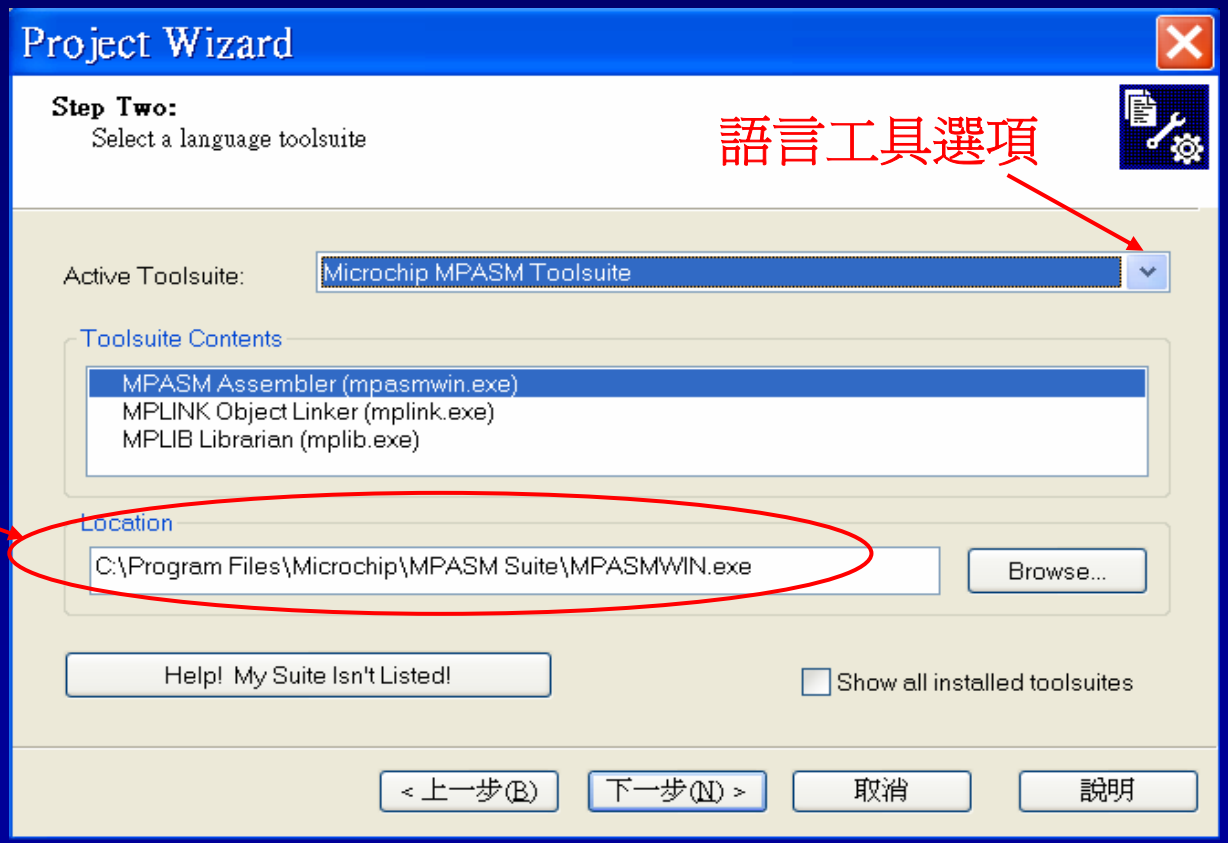
- 利用 **Project Wizard** 建立一個 **New Project** 的步驟：
  - **Project Wizard** 建立 **Project** 視窗



Project Wizard 設定

步驟 3

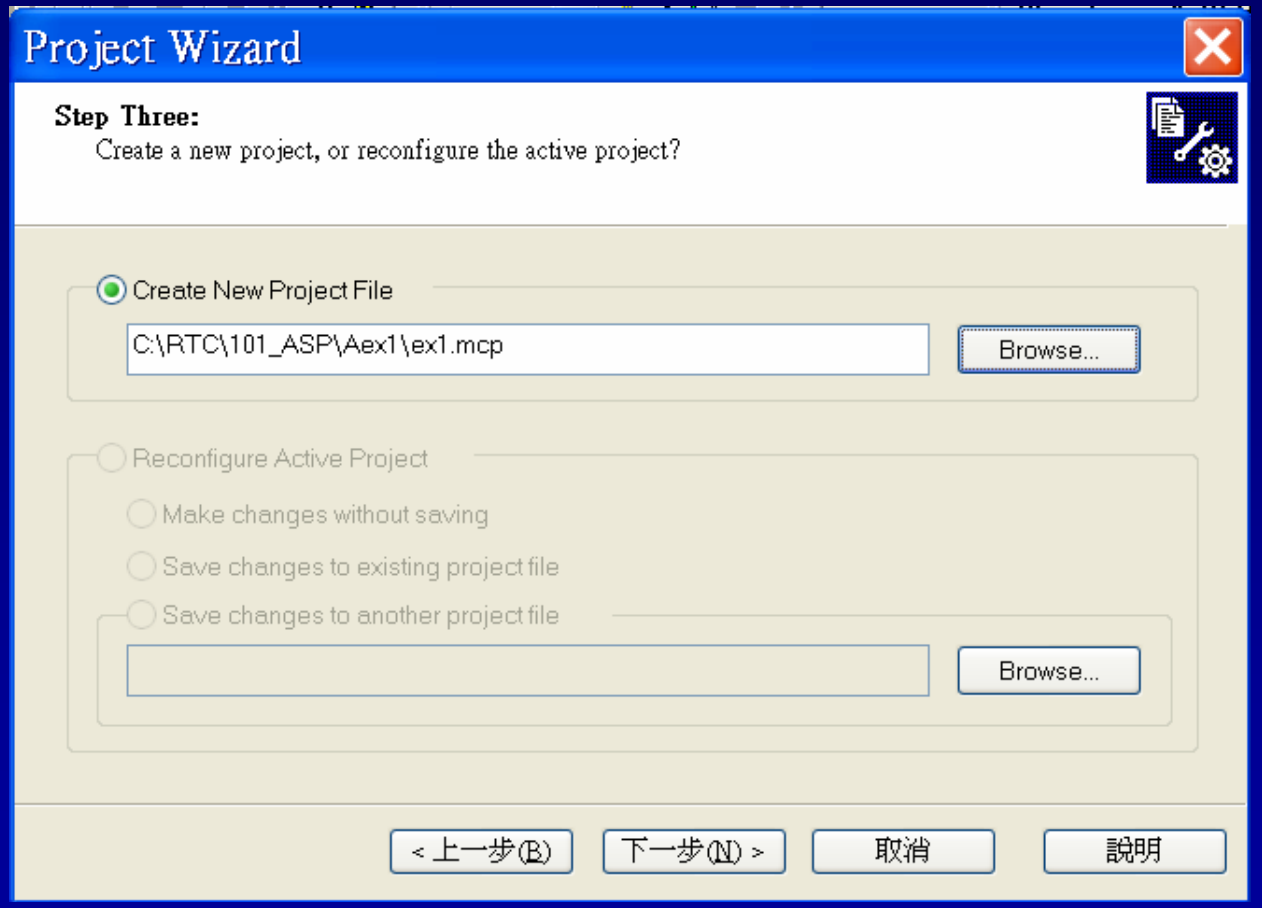
- 選擇所使用的元件為：**PIC16F877A**
- 選擇語言工具：**Microchip MPASM Toosuite**
- 確定 **MPASM** 組譯工具的執行路徑與程式 **(MPASMWIN.exe)**



組譯器的路徑

Project Wizard 設定　步驟 4

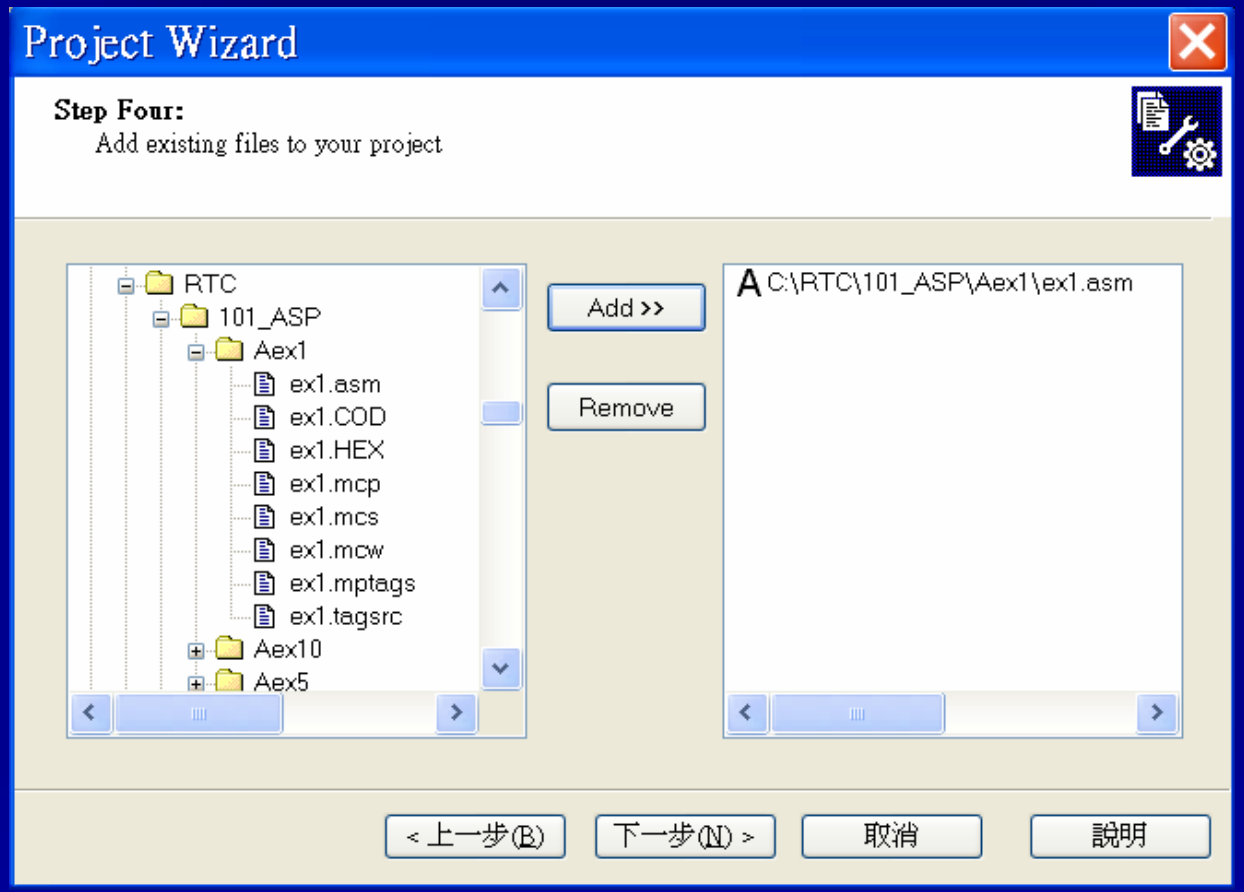
- 設定 **Project** 的檔案路徑與名稱：**ex1.mcp**



Project Wizard 設定

步驟 5

- 將 **ex1.asm** 加入 **Project** 裡



Project Wizard 設定

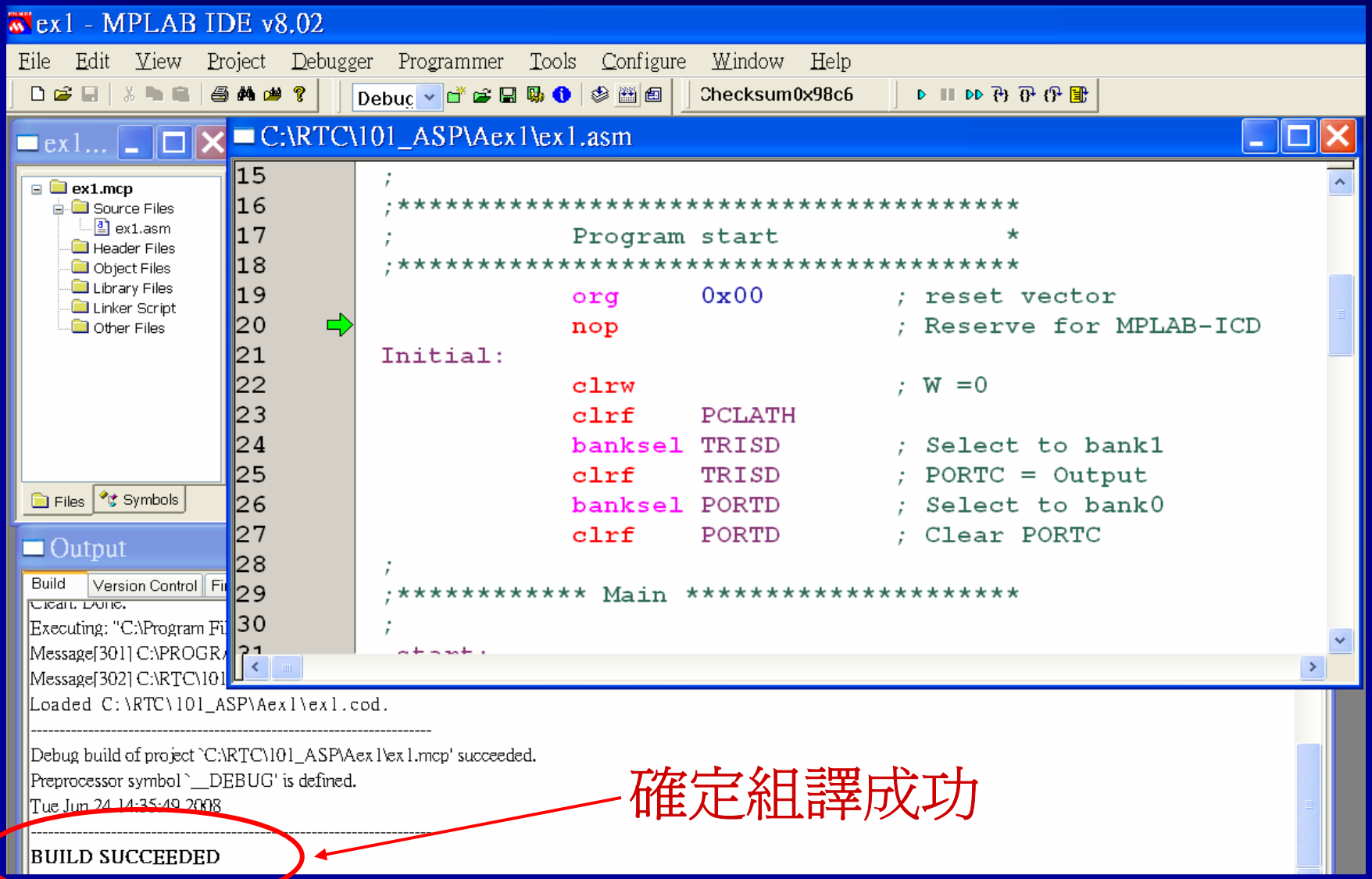
步驟 6

- **Project** 建立完畢，最後確認。

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？

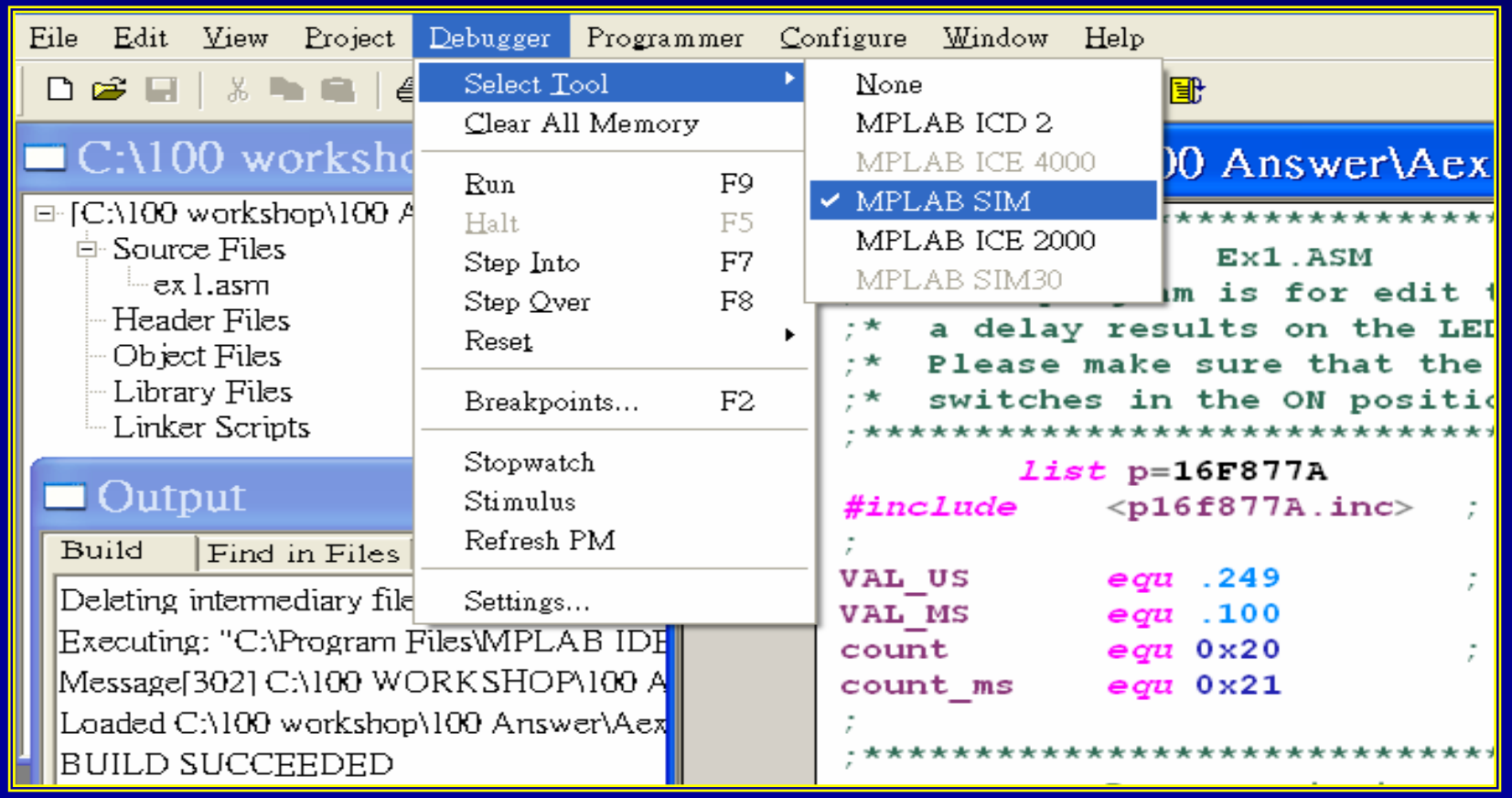
組譯程式 – Build All



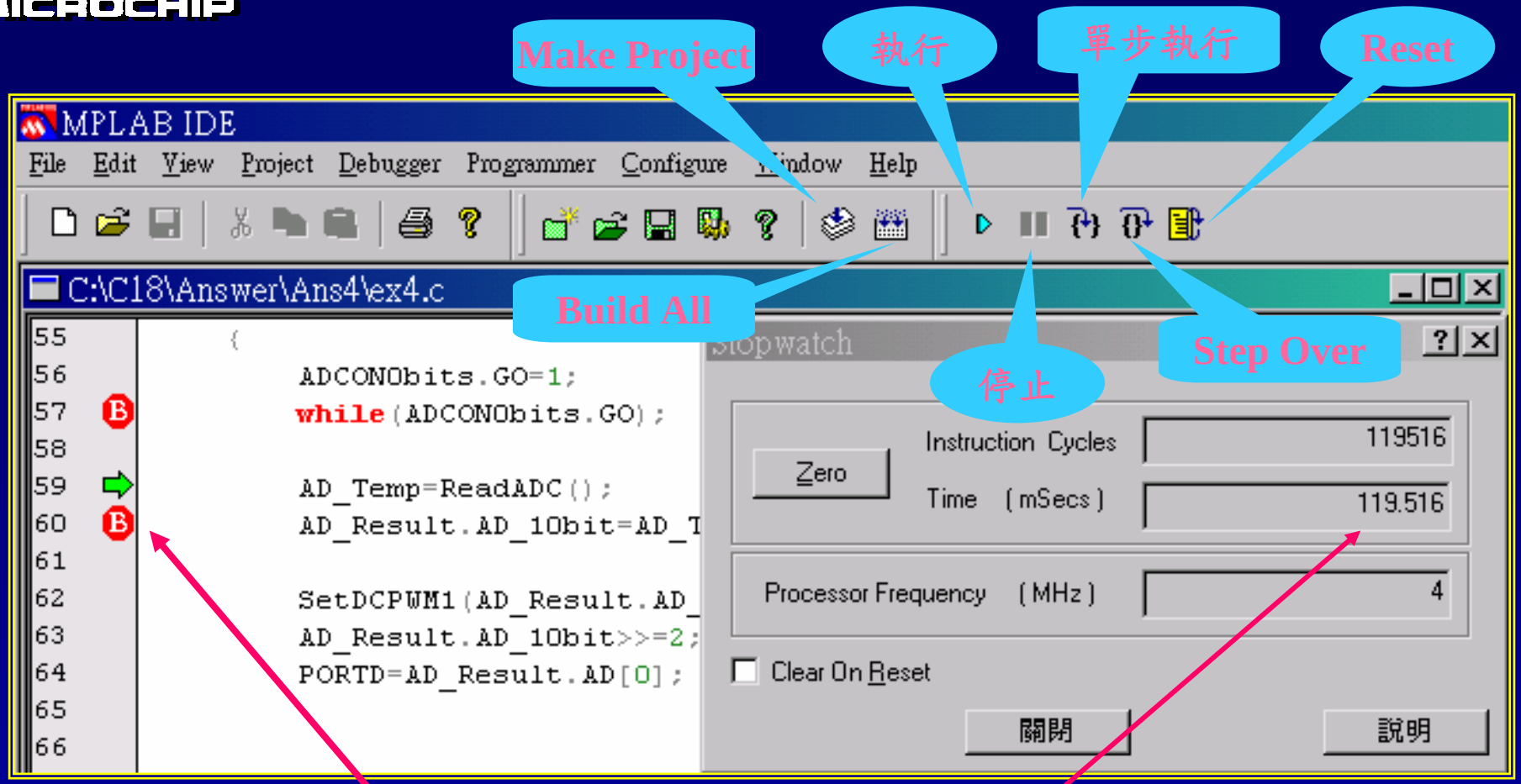
確定組譯成功

啓動基本 MPLAB-SIM

- 點選 **“Debugger → Select Tool → MPLAB SIM”** 以啓動內建的軟體模擬程式



MPLAB SIM 的基本除錯功能

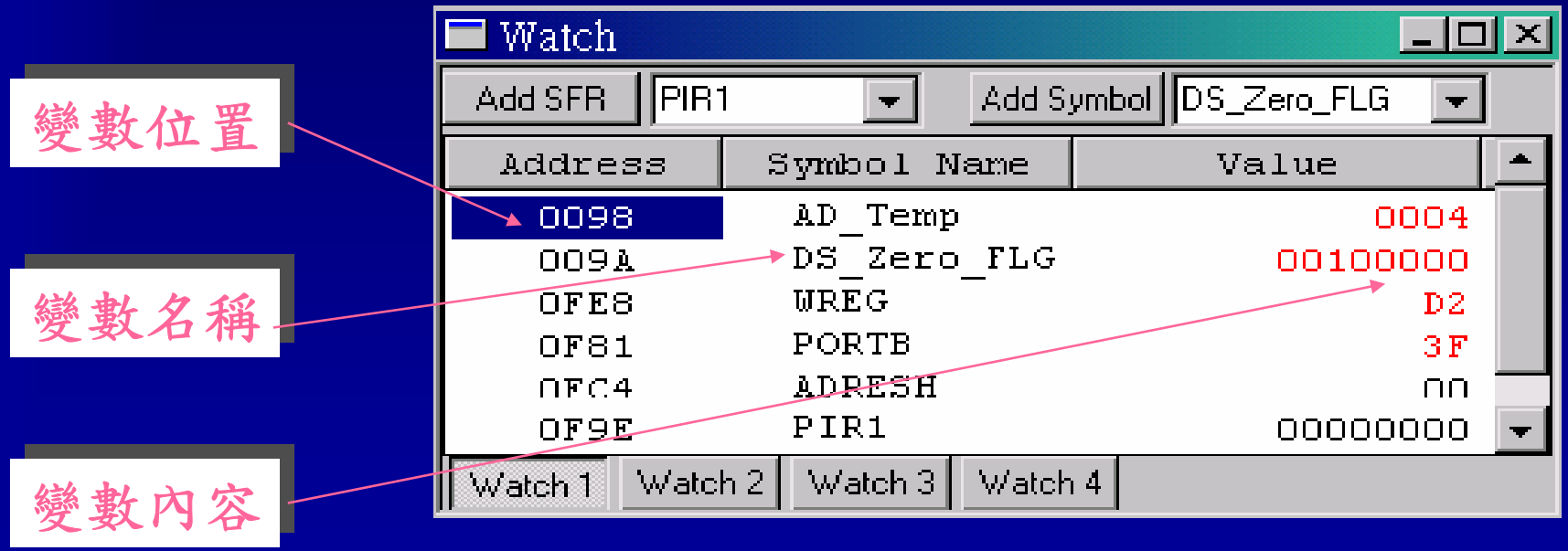


利用mouse的右鍵來設定中斷點

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 (View →Watch)



PIC16Fxxx

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)

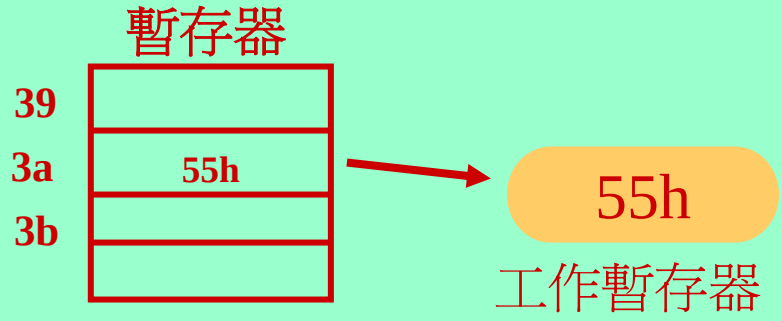
MOVF

- 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。



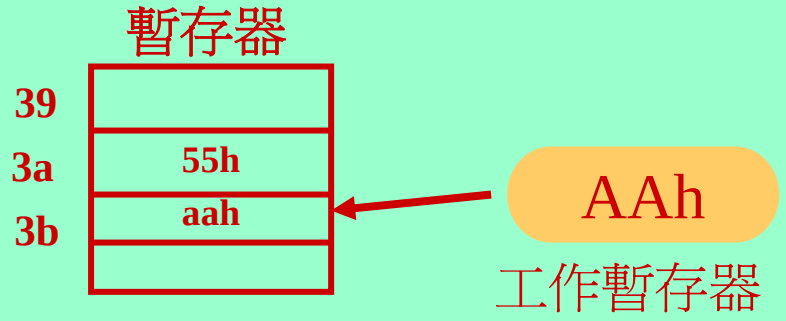
MOVWF

- 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爲 $3B_{(16)}$ 的暫存器中。



資料遞減指令
(DECF , DECFSZ)

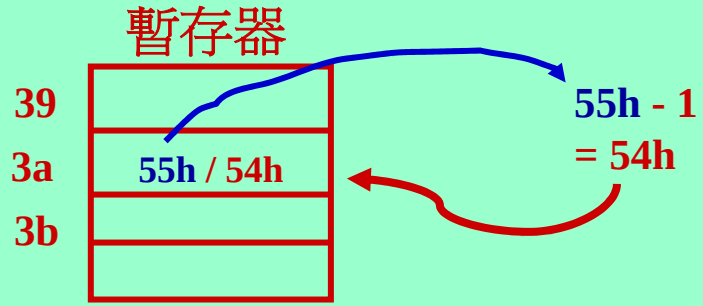
DECF

- 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爲$3A_{(16)}$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) 個位元設定爲 0。

例:

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

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

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

將位址 $3a_{(16)}$ 的暫存器的 bit 7 清除爲“0”。

BSF

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

例:

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

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

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

將位址 $3a_{(16)}$ 的暫存器的 bit 7 設定爲 “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
```

建議語法 →

```
                        addwf        var_count , W
                        addwf        var_count , F
```

必需要使用的虛擬指令

- **LIST - 目錄控制 (Listing Control)**
  - **list p=PIC16F877A, 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 爲 PIC16F877A
               #include    <p16f877a.inc>       ; 使用 16F887A 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
```

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

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
```

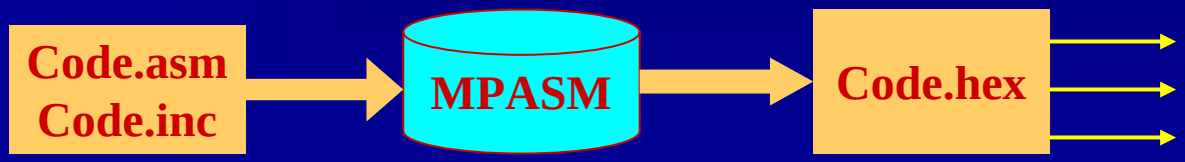
練習三：認識檔案

- 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** 爲名的 **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101ASP 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



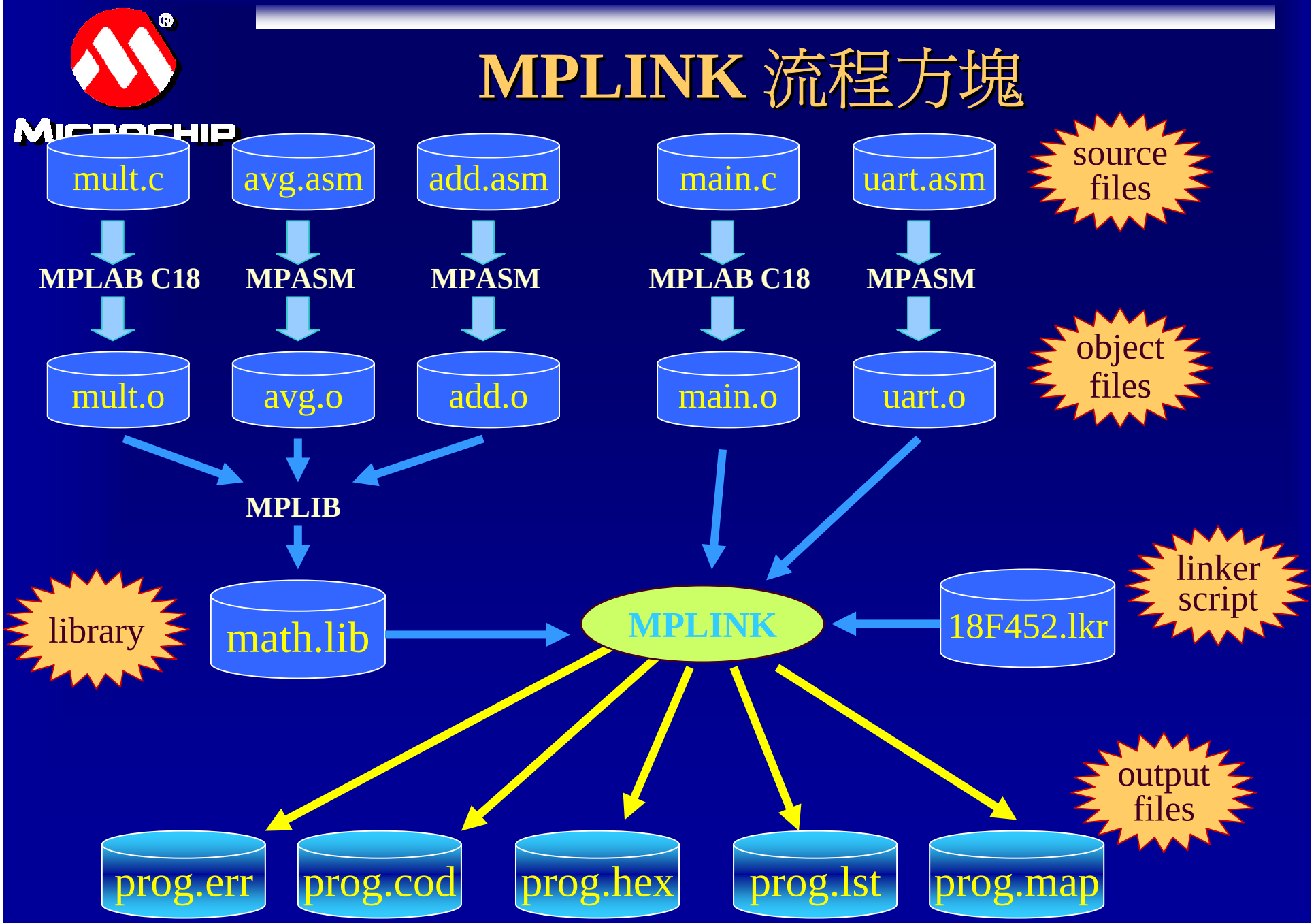
燒錄器 (PM-III)
軟體模擬 (MPLAB-SIM)
硬體除錯器 (Real ICE)
(ICD2)

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

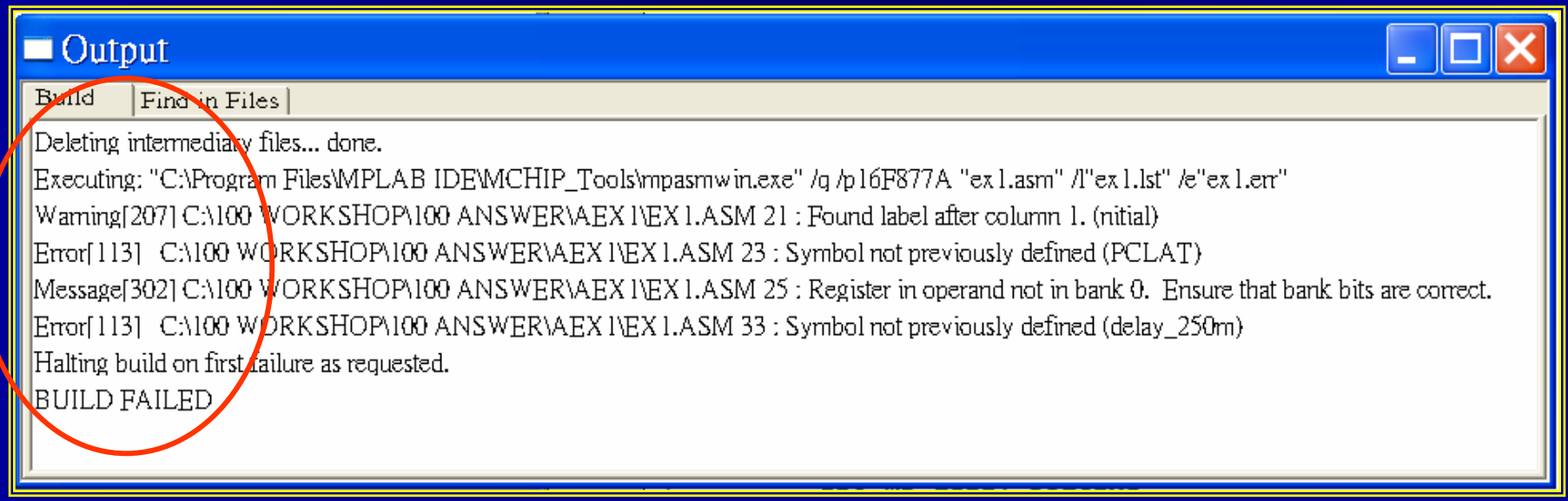
- MPLINK 能作做什麼？
  - 安排實際位址給程式(CODE)及資料(RAM)
  - 產生可執行擋 (HEX)
  - 安排堆疊位址及深度給 MPLAB C18
  - 提供 COD 檔以利程式偵錯
  - 讓程式更容易模組化
  - 連結資料庫 (Library)

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 ?



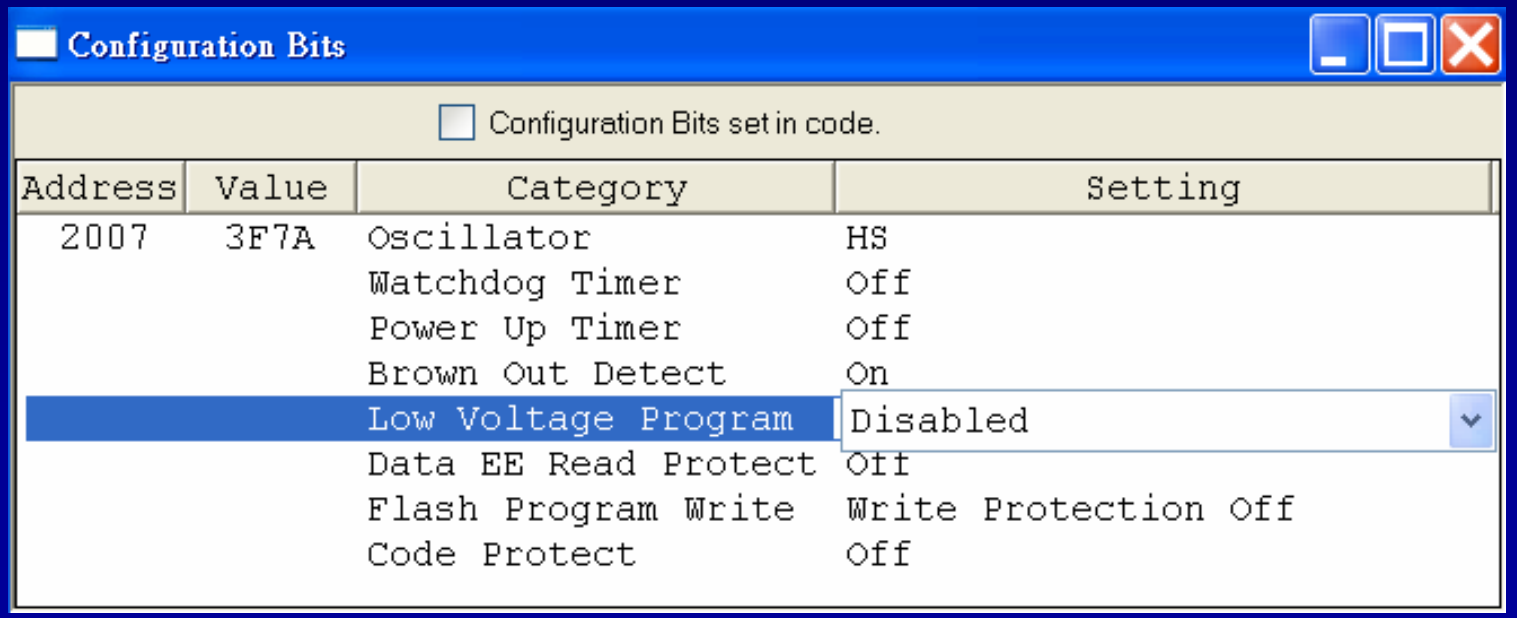
指令的執行時間

```
; 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                   2us
                :                                                +

delay_1ms:
                movlw   VAL_US          ; 1us                   1us
                                                                 +
                movwf   count           ; 1us                   1us
                                                                 +
dec_loop        nop                     ; 1us
                                                                 (1us+1us+2us)*248
                decfsz  count,f         ; count=0, 2us          + (1us+2us) =  995us
                                        ; count>0, 1us
                                                                 +
                goto    dec_loop ; 2us

                return                  ; 2us                   2us
                                                                 ---------
                                                                 1001us
```

該指令執行所需的時間

MPLAB SIM 環境設定

- 在 MPLAB IDE 下選擇 “軟體模擬”
  - Debugger → Select Tool → MPLAB SIM
- 設定系統執行的工作頻率爲 16MHz
  - Debugger → Setting 下設定模擬頻率
- 檢查一下 Configuration Bits 的設定



練習五：寫個 Delay 的副程式

Ex5.ASM

❶ MPALB SIM 下設定模擬的頻率爲 16MHz

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)

練習五 ex5.asm

```
; 請利用Watch Window的計時量測，填入延遲的值。
;
VAL_500US     equ     ???             ; 0.5ms delay value
VAL_10MS      equ     ???             ; 10mS delay value
VAL_200MS     equ     ???             ; 200mS delay value
;
```

1. EQU 內的常數要填入多少值才可以達到 Delay 的要求。
2. 先從 VAL_500US 的值先計算在填入，用軟體模擬達到 500uS 的延遲後再填入 VAL_10MS 的值。
3. 最後完成 VAL_200MS 的值。

解答在 ..\101_ASP\100 Answer\Aex5.asm

PIC16Fxxx
特殊功能概述

PIC16F877A 燒錄設定位元

CP	-	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	-	-	PWRTEN	WDTEN	FOSC1	FOSC0

bit 1 bit 0

- 置放在程式位址 **0x2007**
- 設定元件工作模式
  - **CP : Code Protection –** 程式碼保護
  - **DEBUG : Debug Mode –** 除錯模式
  - **WRT1~WRT0 :** 允許程式記憶體用軟體方式寫入
  - **CPD : Data EEPROM** 保護位元
  - **LVD : Low Voltage Programming –** 低電壓偵測
  - **BOREN : Brown Out Reset –** 電源異常偵測
  - **PWRTEN : Power Up Timer –** 電源上電延遲計時器設定
  - **WDTEN : Watchdog Timer –** 看門狗計時器啓動設定
  - **FOSC1~FOCS0 : Oscillator Mode –** 振盪模式設定
- 錯誤的設定將導致 **PIC** 無法正常工作

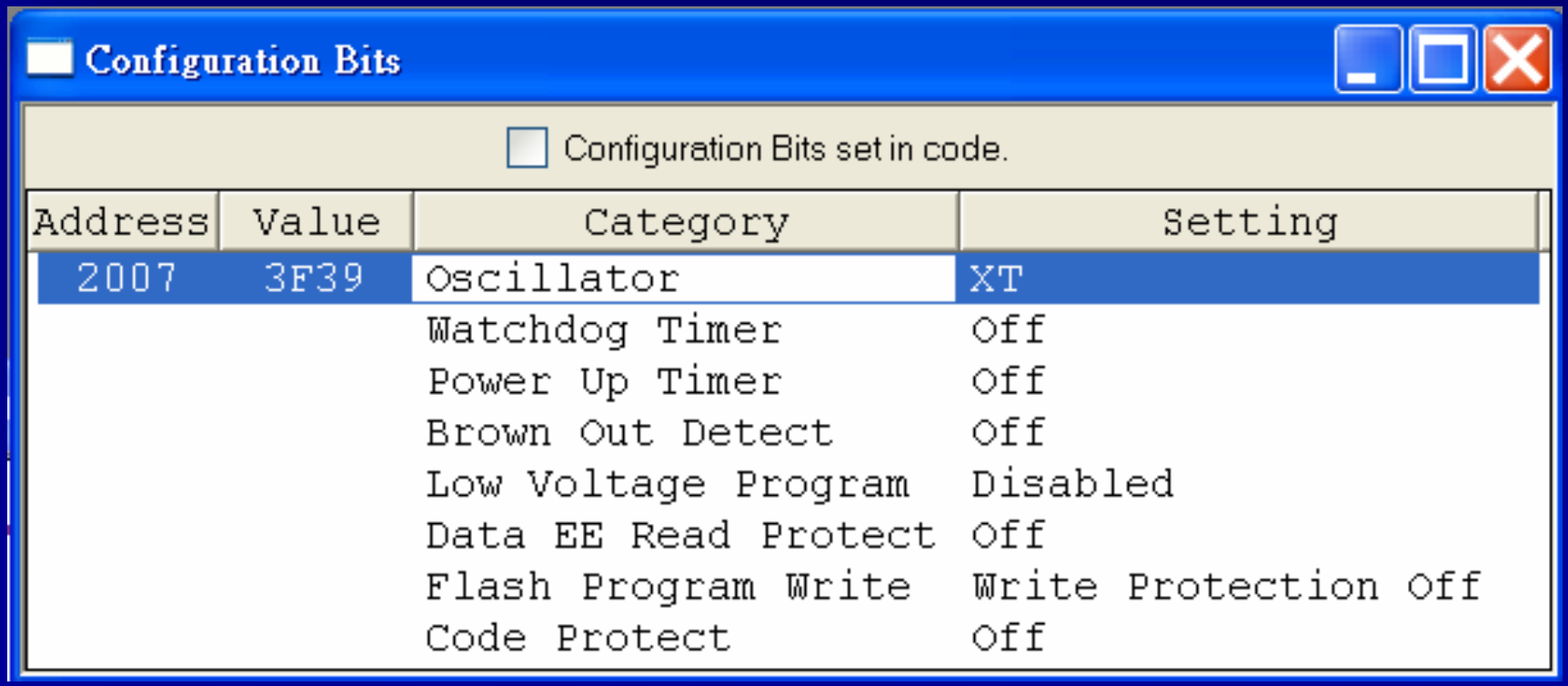
PIC16 振盪器的選項

XT	一般頻率石英振盪電路設定模式	100kHz - 4MHz
HS	高頻率石英振盪電路設定模式	4MHz - 20MHz
LP	低頻率石英振盪電路設定模式	5kHz - 200kHz
RC	使用外部 RC 振盪器	DC - 4MHz
INTRC	使用內建 RC 振盪器	4 or 8 MHz ± 2%

- 提供多樣的振盪器選擇方式，讓設計更加彈性：
  - LP 振盪模式提供了低功耗功能
  - RC 或 INTRC 振盪模式提供了低成本的效益
  - XT 提供一般使用的工作頻率
  - HS 提供需高速處理所需的工作頻率
- 速度範圍參考資料手冊所建議的範圍

設定 Configuration Word 的方式

- 直接在 **MPLAB IDE** 下設定
  - 先確定 **MPLAB IDE** 使用的元件名稱
  - **MPLAB IDE** 目錄下 **: Configure → Configuration Bits**
  - 取消 **Configuration Bits Set in code**



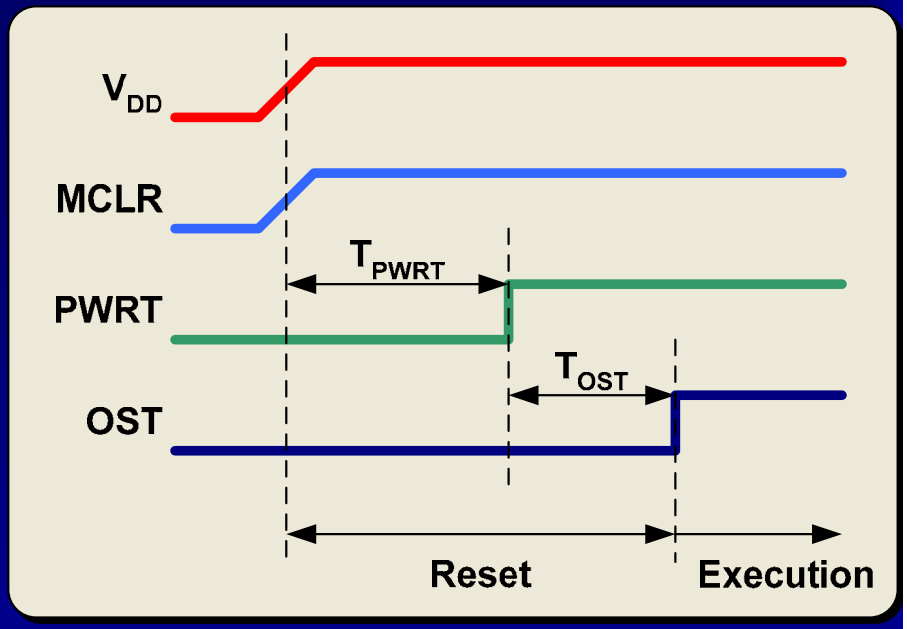
程式設定 Configuration Word

- 一般會建議利用程式內建 **Config.** 方式來設定
  - 編譯過後，**HEX** 檔會有設定值的存在
  - 避免燒錄時的人為設定上的選擇錯誤
- 使用虛指令 **_ _ config** 的設定指令
  - 參考 **C:\Program Files\Microchip\MPASM Suite\Template\Code** 裡的 **16f877a.tmp** 裡的範例
  - 相關設定的定義字參考 **inc** 檔

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

__config        _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON
                & _HS_OSC & _WRT_OFF & _LVP_OFF & _CPD_OFF
```

POR, OST, PWRT

- **POR: 電源開機重置**
  - MCLR 腳接到 V_{DD}，當 V_{DD} 上升時會產生重置動作 (Reset)
- **PWRT: 電源上電延遲計時器**
  - 偵測到電源開機後(POR)，自動延遲72mS以確保MCU可以在工作電壓下執行
- **OST: 振盪器穩定計時器**
  - HS，XT，LP 三種振盪模式的啓動穩定延遲，自動計數 1024 的 Cycles 後送出震盪信號給 MCU。(不適用在 RC 模式) 只在 POR啓動後及喚醒睡眠模式下會做此動作確保頻率的穩定

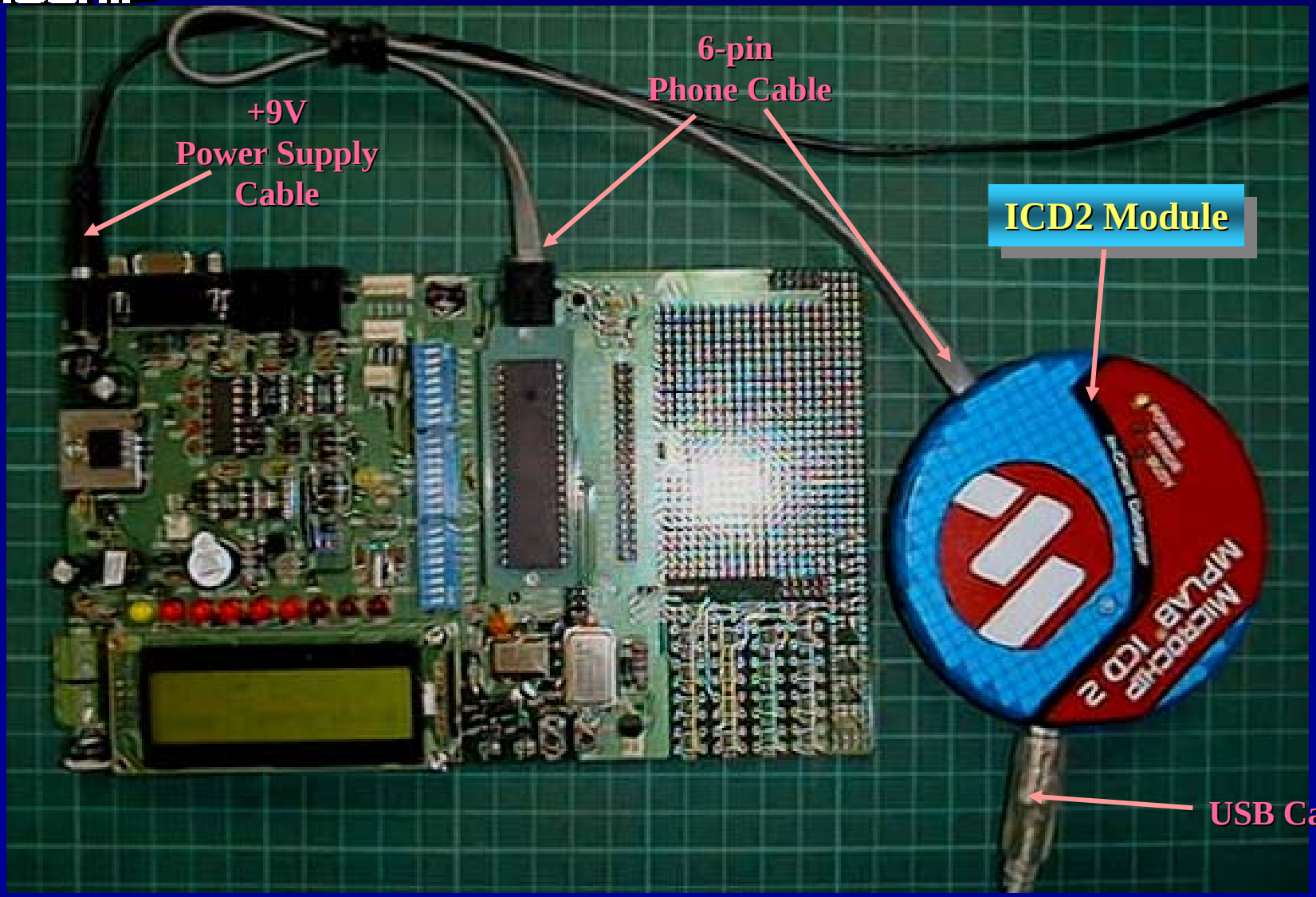


使用 MPLAB ICD2 除錯

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**

MPLAB-ICD2 配線圖



MPLAB-ICD2 注意事項

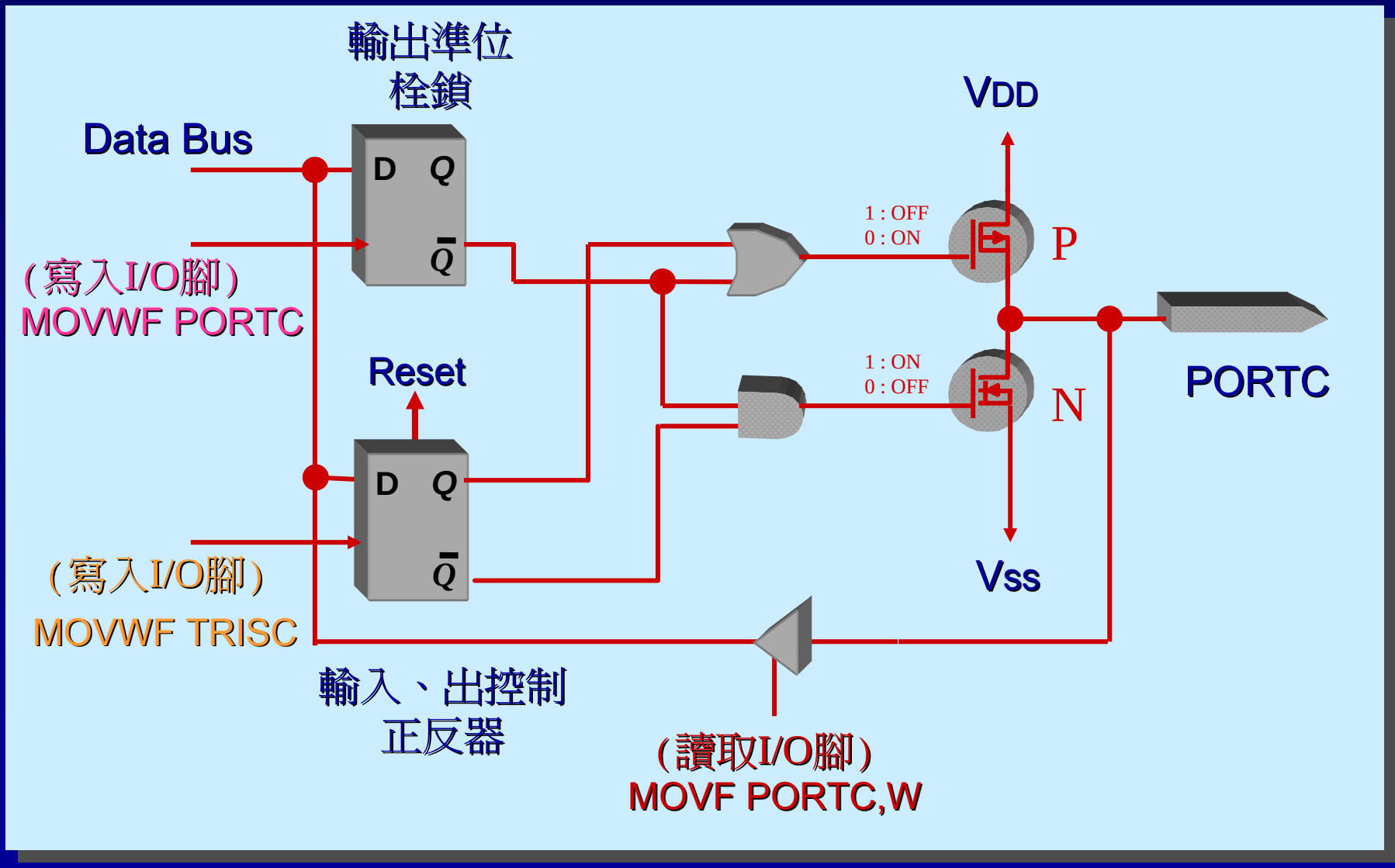
- ICD2 佔用一層堆疊, 使用者只可使用七層堆疊
- 程式位址 (0x1F00-0x1FFF) 保留給監督程式使用
- 程式位址 0x0 必須填入“NOP” 指令
- RB6 & RB7 保留給 ICD 做除錯用
- Data Memory 0x70 , 0x1EB – 0x1EF 等六個 RAM 將被佔用 !!
- MCLR pin 會出現 13V 的電壓 (thru a 1kohm resistor)

使用 MPLAB-ICD2

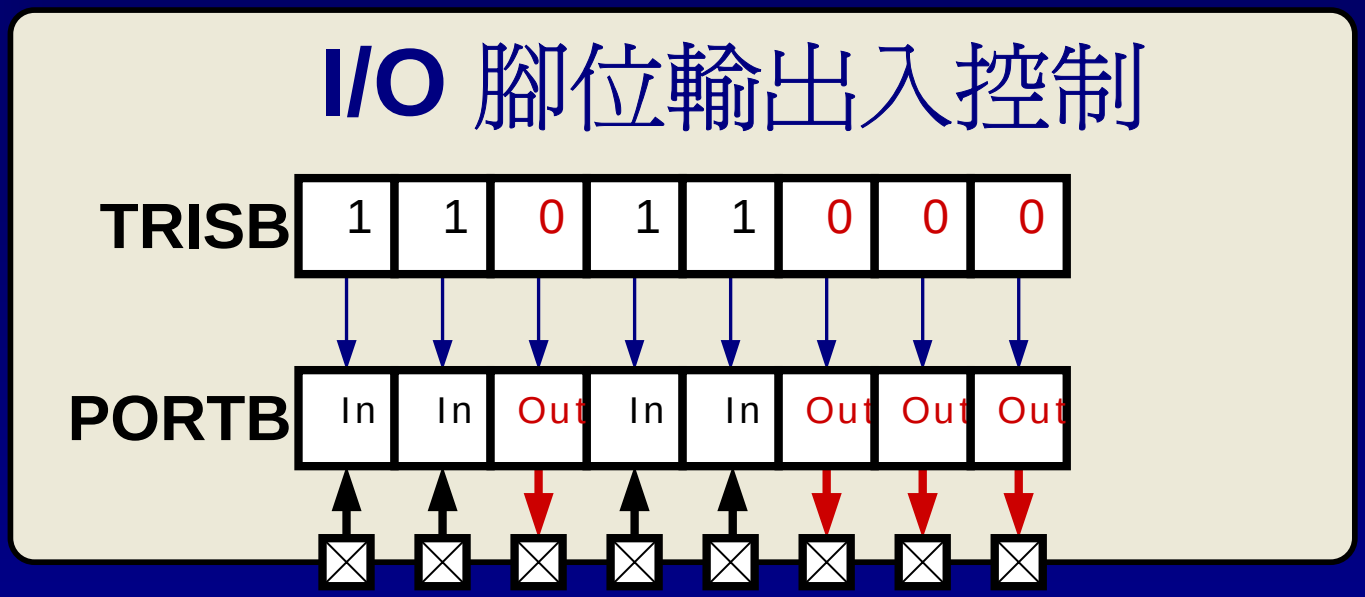
- 有關使用 **ICD2** 的詳細步驟，請參閱：
  - **MPLAB IDE v6.10** 中文使用手冊，第五章
- 使用 ICD2 時，務必重新載入目前所使用MPLAB IDE 版本的 ICD2 Firmware 以確保 ICD2 能與 MPLAB IDE 相容
  - 先點選 **“Debugger → Select Tool → MPLAB ICD2”**
  - 再點選 **“Debugger → Download ICD2 Operating System”**

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
VDD
1 : OFF
0 : ON
P
1 : ON
0 : OFF
N
Vss
PORTC
(讀取I/O腳)
MOVF PORTC,W

I/O Ports



- **TRISx 暫存器控制著相對應的 PORTx 的輸出入功能，而且是對應到每一個位元控制**
- **1 = Input, 0 = Output**

設定 I/O Pin

- 試著將 I/O (PORT) 視爲一個暫存器 (RAM)來操作
- TRISx 爲 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 爲輸出時，則將相對應的 TRISx 設定爲 “0”
  - 要設定 PORTx 爲輸入時，則將相對應的 TRISx 設定爲 “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 基本電氣規格
  - IC V_{DD} pin 最大電流：250mA
  - IC V_{SS} pin 最大電流：300mA
  - 每個 I/O pin 最大驅動 / 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

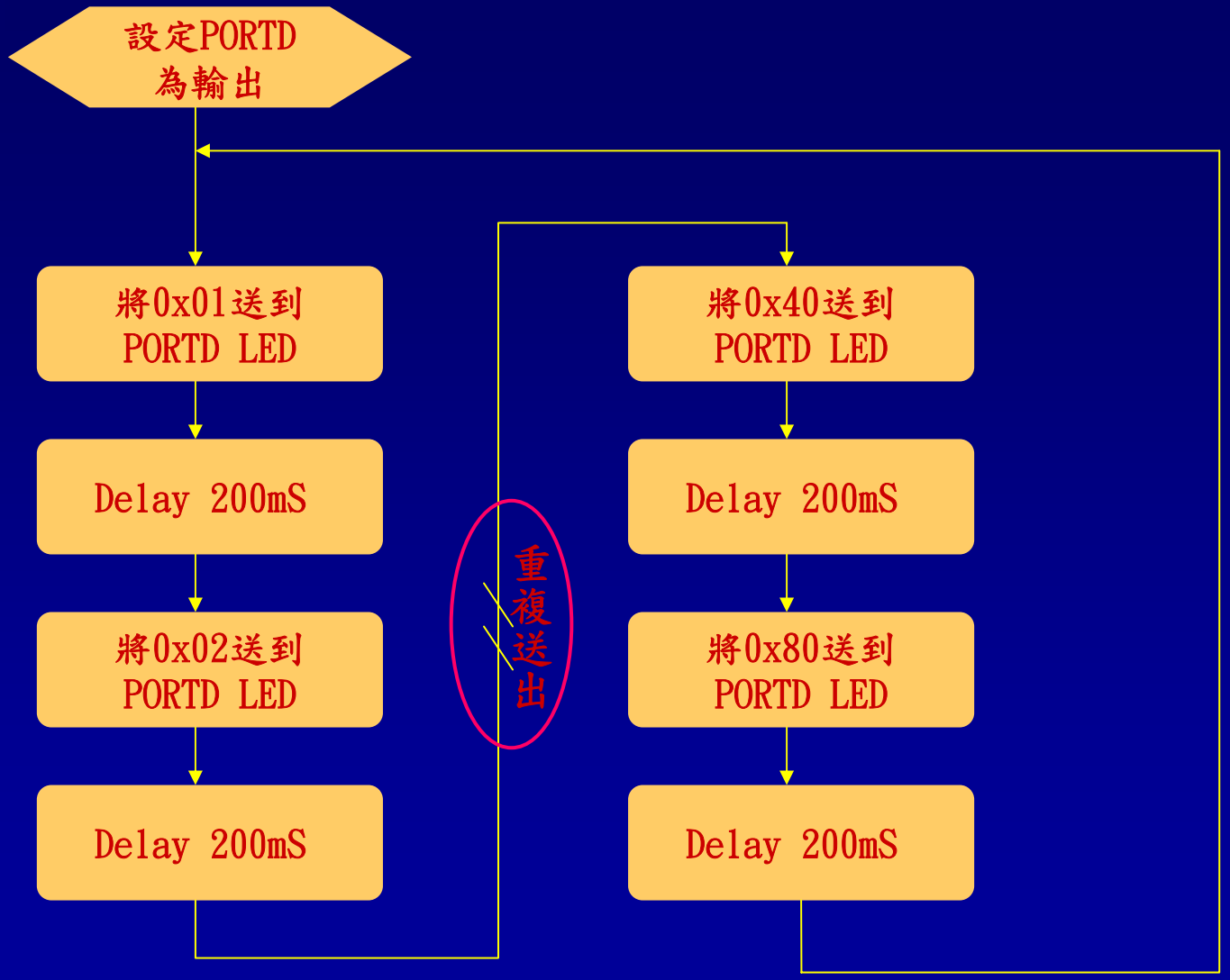
- 確定選擇 HS 振盪模式，使用 16MHz Crystal
  - 16 MHz Crystal 上方有三個 Jumper
  - JP1 (XTAL) & JP2 (XTAL) 短路，JP3 (OSC) 需空接
- 將練習五完成的 200mS 延遲副程式，作爲延遲時間的基準
- 設計一個程式能將 PORTD 的 LED 從“零”開始每隔 200mS 自動加一
- 利用 ICD2 來進行除錯
  - 設中斷點、變數觀察視窗

練習六 ex6.asm

```
;************* Main ************************
;
start:
                banksel   TRISD           ; Set PORTD for output port
;               ?????
;               ?????
                banksel   PORTD           ; Clear PORTD
                ?????
;
Inc_LED
;               ????                      ; Call Delay 200mS subroutine
;               ????                      ; PORTD = PORTD + 1
                goto      Inc_LED
```

解答在 ..\101_ASP\100 Answer\Aex6.asm

簡單的跑馬燈控制流程


設定PORTD
為輸出
將0x01送到
PORTD LED
Delay 200mS
將0x02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

- 以練習六爲程式架構，仍以 **200mS** 延遲副程式作爲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

狀態暫存器 (旗號)

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

bit 7 bit 0

IRP: 索引定址時的 BANK 選擇位元 (索引時的單一BANK爲256Bytes)

0 = Bank 0, 1 1 = Bank 2, 3

RP1:RP0: 暫存器頁的切換選擇，配合直接定址模式使用

00 = Bank 0, 01 = Bank 1, 10 = Bank 2, 11 = Bank 3

TO: 看門狗計時器溢位旗號

0 = A WDT time-out occurred, 1= Power-up, CLRWDT or Sleep Instruction

PD: 掉電指示旗號

0 = SLEEP instruction executed, 1= Power-up, CLRWDT Instruction

Z: 零旗號

1 = Result of arithmetic operation is zero

DC: 半進位旗號

1 = Carry out of 4th low order bit occurred / No borrow occurred

C: 進位/借位旗號 (減法採 2's 運算)

1 = Carry out of MSb occurred / No borrow occurred

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
- **MPLAB IDE 的旗號顯示 (大寫表示爲 1) :**
  - C / c - 進位 或 借位旗號
  - Z / z - 零旗號
  - DC / 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爲零則 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

- 一般稱之爲“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**?**

練習八 ex8.asm

```
LED_Start       bcf     STATUS,C
                movlw   b'00000001'
                movwf   PORTD
;
;----------  利用旋轉指令改寫程式  ---------
;
LED_Next
;               ?????           延遲200mS
;               ?????           旋轉 PORTD
;               ?????           測試 C 旗號
;               ?????           如果C=0 的話
                goto    LED_Start
```

解答在 ..\101_ASP\100 Answer\Aex8.asm

練習九：設計一霹靂燈

Ex9.ASM

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

- 將顯示值 **“b’00000111”** 輸出到 **PORTD**，若 **RA4 = 1 (** 按鍵放開 **)** 則每隔 **0.1SEC** 向左移一位 **(** 一次移 **8** 個位元 **)**，**RA4 = 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 **?**

練習九 ex9.asm

```
start:
                    movlw     SHIFT_VAL
                    movwf     PORTD
;
test_rb0            btfss     RA4             ; Check RA4 press ?
                    goto      led_right       ; Yes, RA4 closed, b0-->C-->b7
;
; ------  判斷是左移或是右移  ---------
;
led_left
;                   ????                      ; No, RA4 is open, B7-->C-->b0
;                   ????                      ; Call 200mS delay routine
;                   ????
;
led_right
;                   ????                      ; Yes, RA4 closed, b0-->C-->b7
;                   ????                      ; Call 200mS delay routine
;                   ????
;
```

解答在 ..\101_ASP\100 Answer\Aex9.asm

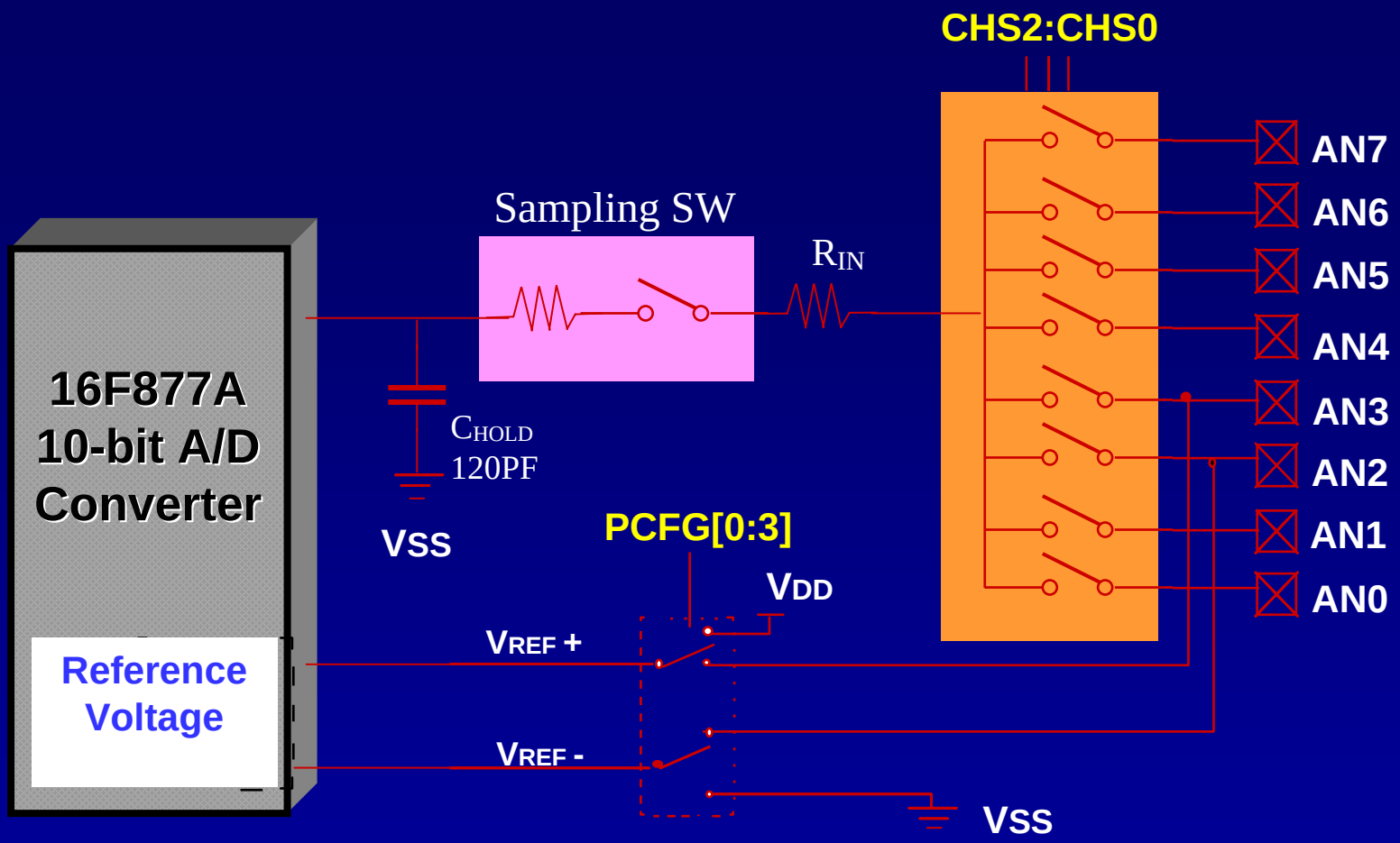
可調整速度的霹靂燈設計

附加進階練習

PIC16F877A 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



注意：我是 SAR 的架構，所以要有取樣時間的延遲來讓電容充飽電
不然我會不準 或 會有相互干擾的現象

See Data Book

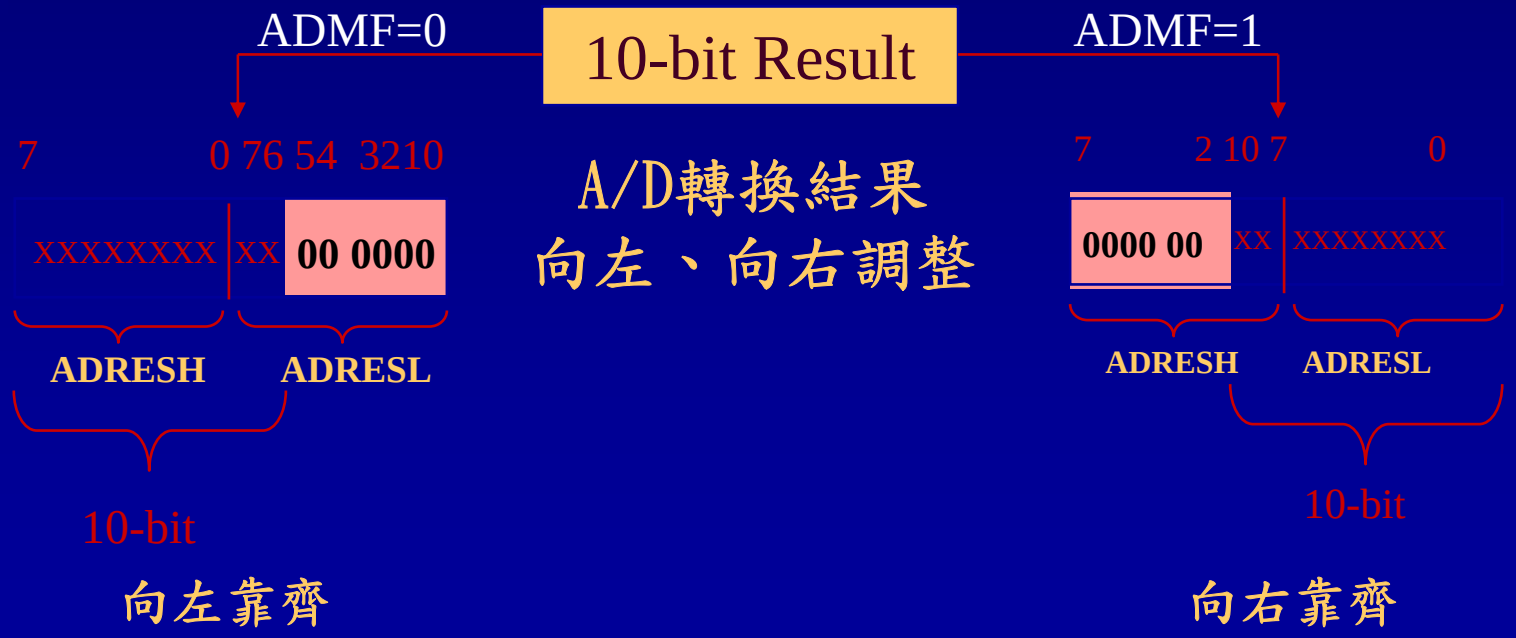
A/D 控制暫存器

ADCON0 Register

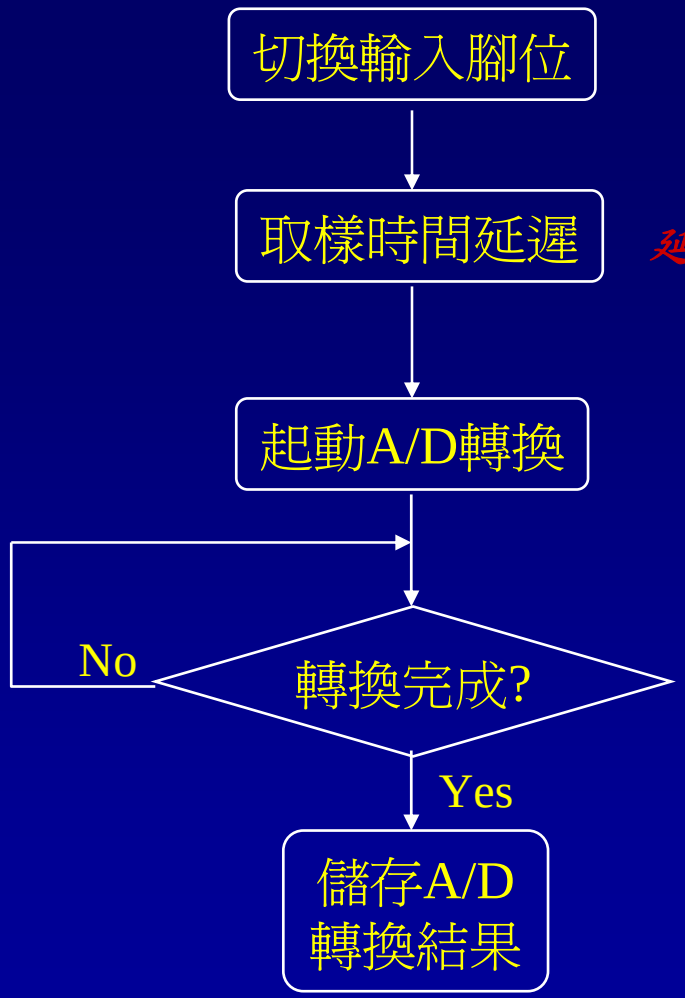
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	---	ADON
bit7							bit0

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
  - **CMCON [CM2,CM1,CM0]=111**

Thank You

別忘了填寫問卷調查表