HI-TECH C® Compiler with Mid-Range PIC® MCU Family

Hi-Tech PICC Workshop v2.0

Rev 2.0 August 2011 Modified by Richard yang

課程說明

- 此教育訓練課程不是 **C** 的基礎課程
- 我們認定你已經有一定使用 **C** 的能力
  - 使用過 **MPLAB IDE**
  - 使用過 **ANSI C** 或其它的 **C** 語言
- 本課程不在介紹如何撰寫 **C** 程式，而是介紹如何使用 **Hi-Tech PICC Compiler**
- 本教育訓練使用元件爲
  - **Mid-Range : PIC16F887**

Hi-TECH PICC v9.xx
Compiler Overview

Rev 2.0 August 2011 Modified by Richard yang

關於本課程之工具

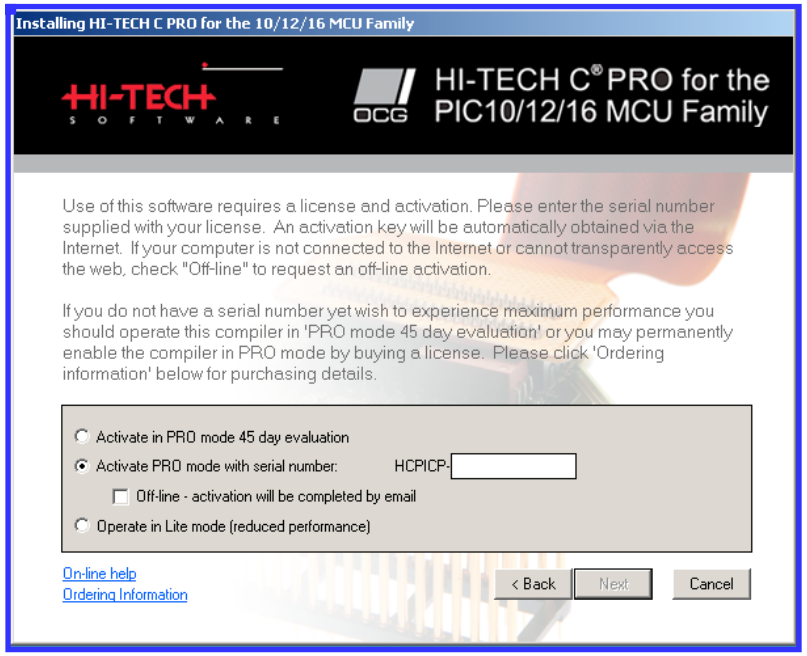
- 本教材在撰寫時所使用的軟體工具版本
  - **MPLAB IDE v8.70**
  - **Hi-Tech PICC v9.82**
- 軟體工具下載網站
  - http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=81
- 使用的元件：**PIC16F887**
- 實驗與練習
  - **MPLAB SIM** 軟體模擬
  - 硬體部分使用 **Proteus** 硬體電路模擬
  - 或使用 **APP001 + PIC16F887-I/P**

PICC 四種授權版本

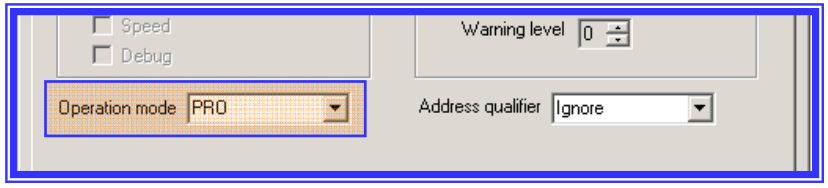
- **PRO 版本 (需購買)**
  - 全功能最佳化 **(optimized)**
- **Standard 版本 (需購買)**
  - 有限的最佳化功能
- **Lite 版本 (免費版本)**
  - 僅具有基本的最佳化功能，不受時間及編譯容量的限制
- **Evaluation 版本有效期爲 45 天的PRO 模式，超過期限後將轉爲 Lite 版本**

安裝 PICC 編譯器

- 在安裝時，程式會要求輸入完整的安裝序號
  - 如右圖所示，請在此時輸入所購買版本的序號。

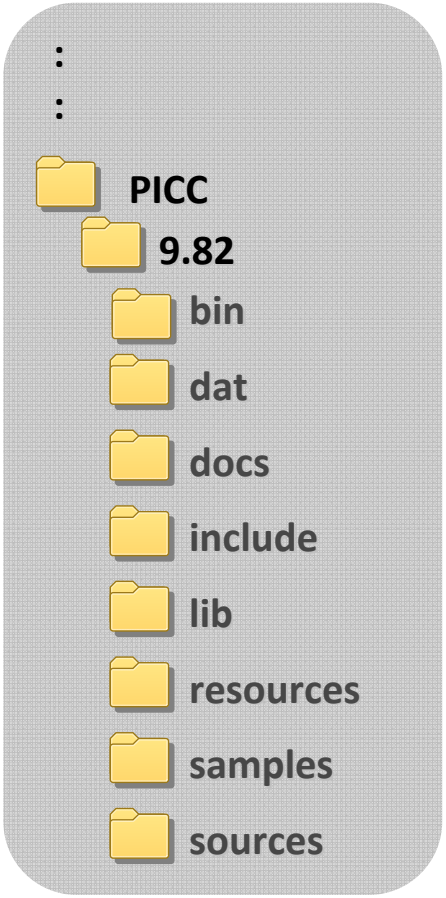


- 編譯模式可以在 **Project** 視窗下更改



安裝完成後的檔案路徑

- 編譯器相關路徑
  - **C:\Program Files\HI-TECH Software\PICC\9.82**



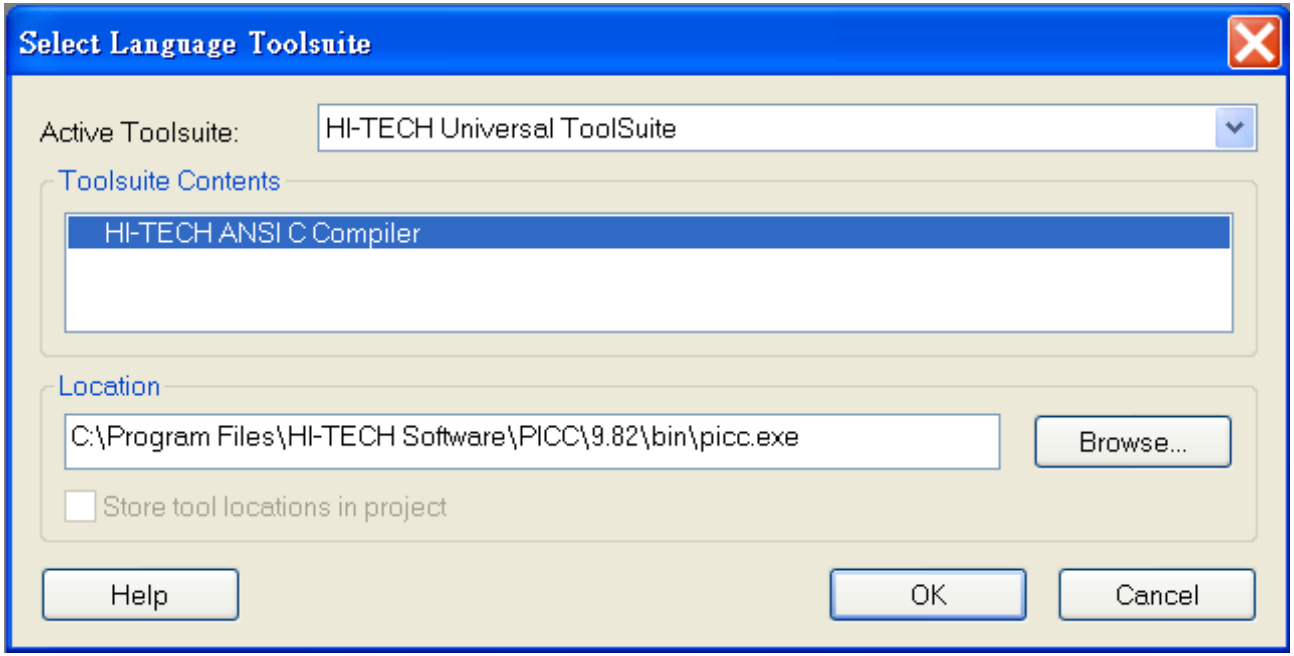
- **..\bin**：編譯器的執行檔 **(exe)**
- **..\dat**：組態設定檔 **(picc.ini)** 及圖庫
- **..\docs**：編譯器相關使用說明文件
- **..\include**：編譯器所提供的 **Head Files**
- **..\lib**：函數庫
- **..\resources**：安裝或修復支援
- **..\samples**：範例程式
- **..\sources**：函數庫原始程式

MPLAB IDE 設定語言工具

Hi-Tech PICC

1. 使用 “Project → Select Language Toolsuite” 來設定語言工具爲 **HI-TECH Universal ToolSuite**
   - 安裝於預設的路徑，如下圖所示：

     C:\Program Files\HI-TECH Software\PICC\9.82\bin\picc.exe



PICC 執行檔的說明(一)

- **Hi-Tech PICC** 是由多個主要的　應用程式組合而成
- 這些應用程式包含:
  - **CPP.exe - The pre-processor**
    - 先行處理前置處理器的虛指令及移除程式的註解說明
  - **P1.exe - The parser**
    - 轉換原始程式爲字母標記的助憶符號，檢查並分析原始程式的程式語法

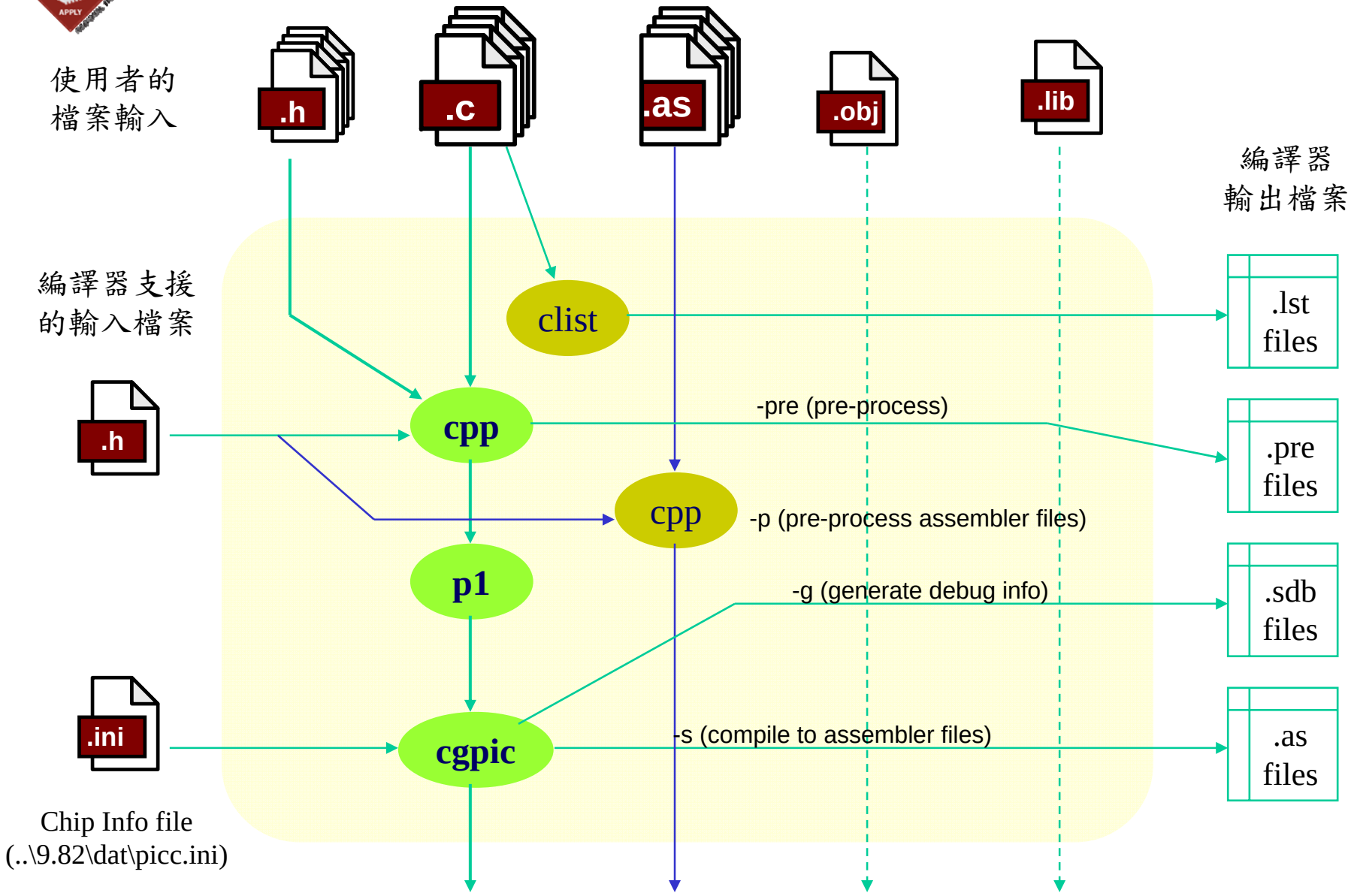
PICC 執行檔的說明(二)

- **CGPIC.exe - The code generator**
  - 將原始程式轉換成助憶符號的組合語言型態，其內容包含各個獨立的程式節區
- **ASPIC.exe - The assembler**
  - 將組合語言轉換成可重新定位址的目的 **(Relocatable object file)**
- **HLINK.exe - The linker**
  - 安排變數在 **RAM** 的實際位址
  - 聯結資料庫，並將各個獨立的程式節區的目的碼重新排定執行位址

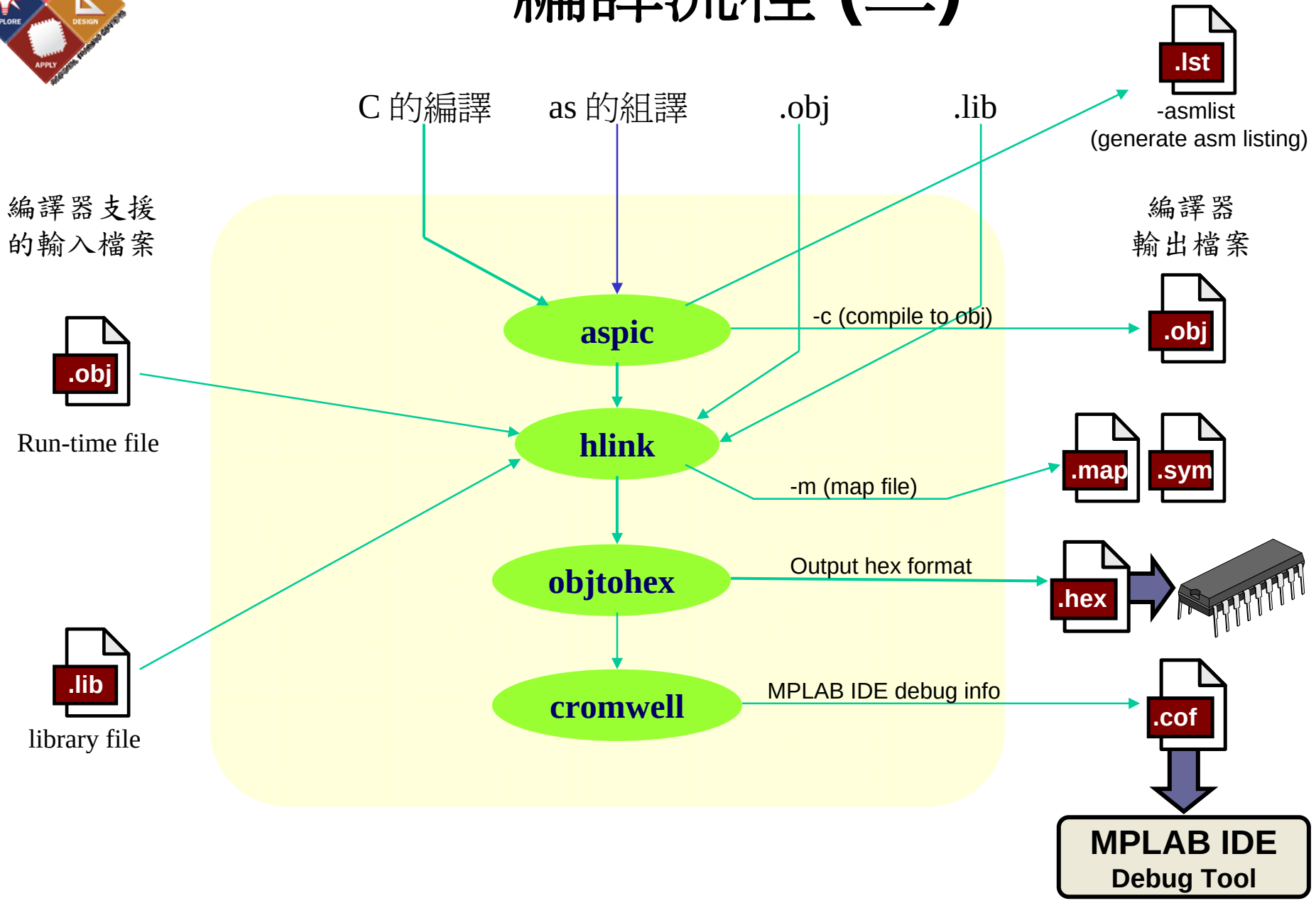
PICC 執行檔的說明(三)

- **OBJTOHEX.exe - The output file converter**
  - 產生一個可執行的 **Hex** 檔案
- **CROMWELL.exe - The re-formatter**
  - 產生原始程式除錯的相關訊息檔案
- 其它：
  - OBJTOHEX.exe : Conversion utility to create HEX files
  - LIBR.exe : Librarian
  - CLIST.exe : Text file formatter
  - DUMP.exe : Object file viewer
  - CREF.exe : Cross reference utility

編譯流程 (一)



編譯流程 (二)



Hi-Tech PICC 輸入檔案

附加檔	名稱	檔案內容說明
.c	C source file	C 原始程式
.h	Header file	C 或 組合語言的宣告定義檔案
.as	Assembler file	Hi-Tech PICC 的組合語言程式
.obj	Object file	Hi-TECH PICC編譯或組譯過所產生的可重新定位 object 檔案
.lib	Library file	HI-Tech 資料庫格式的檔案

其它檔案說明

附加檔	名稱	檔案內容說明
.lst	C listing file	C source code with line numbers
.map	map file	symbol and *psect* relocation information
.lst	Assembler listing	C source with corresponding assembler instruction
.sdb	Symbolic debug file	object names and types for module
.sym	Symbol file	absolute address of program symbols

元件的記憶容量及位址設定
..\PICC\9.82\dat\picc.ini

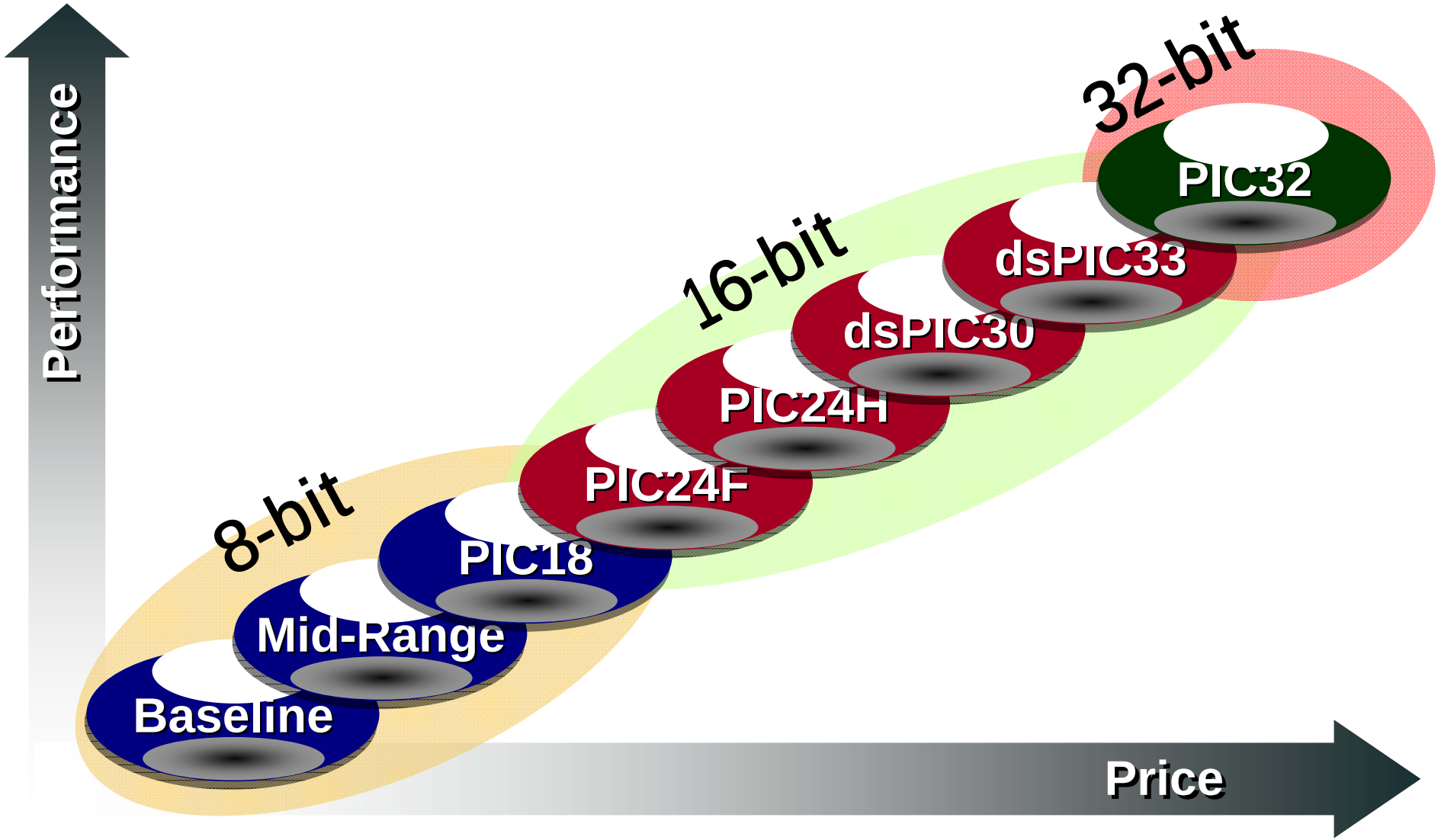
- 每一顆 **PIC** 都會有該元件的記憶體容量及位址設定，**hlink.exe** 在做連結時需要使用
- 底下以 **PIC18F887** 爲例，所有的設定項的說明請參考 **picc.ini** 第一頁的說明

```
[16F887]
MAKE=MICROCHIP
ARCH=PIC14                                  // Mid-Range Core
PROCID=887F
FLASH_READ=1
FLASH_WRITE=8
FLASH_ERASE=10
FLASHTYPE=READWRITE_A
IDLOC=2000-2003
CONFIG=2007-2008
EEPROMSIZE=100
ROMSIZE=2000
BANKS=4
RAMBANK=20-7F,A0-EF,110-16F,190-1EF
COMMON=70-7F
DATABANK=2
ICD2RAM=70-70,1E5-1F0
ICD2ROM=1F00-1FFF
ICD3RAM=70-70,1E5-1F0
ICD3ROM=1F00-1FFF
```

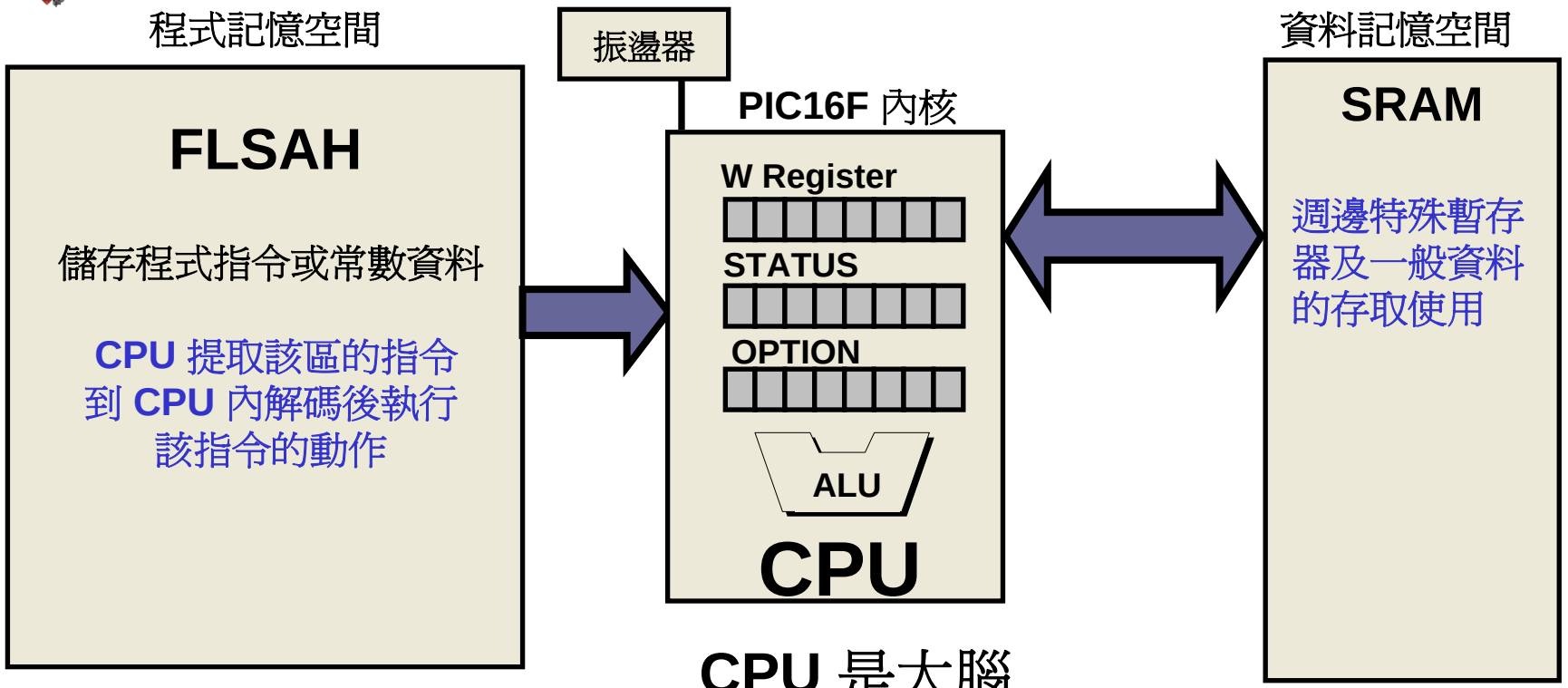
Mid-Range Family
基本架構

Rev 2.0 August 2011 Modified by Richard yang

PIC® 家族分類



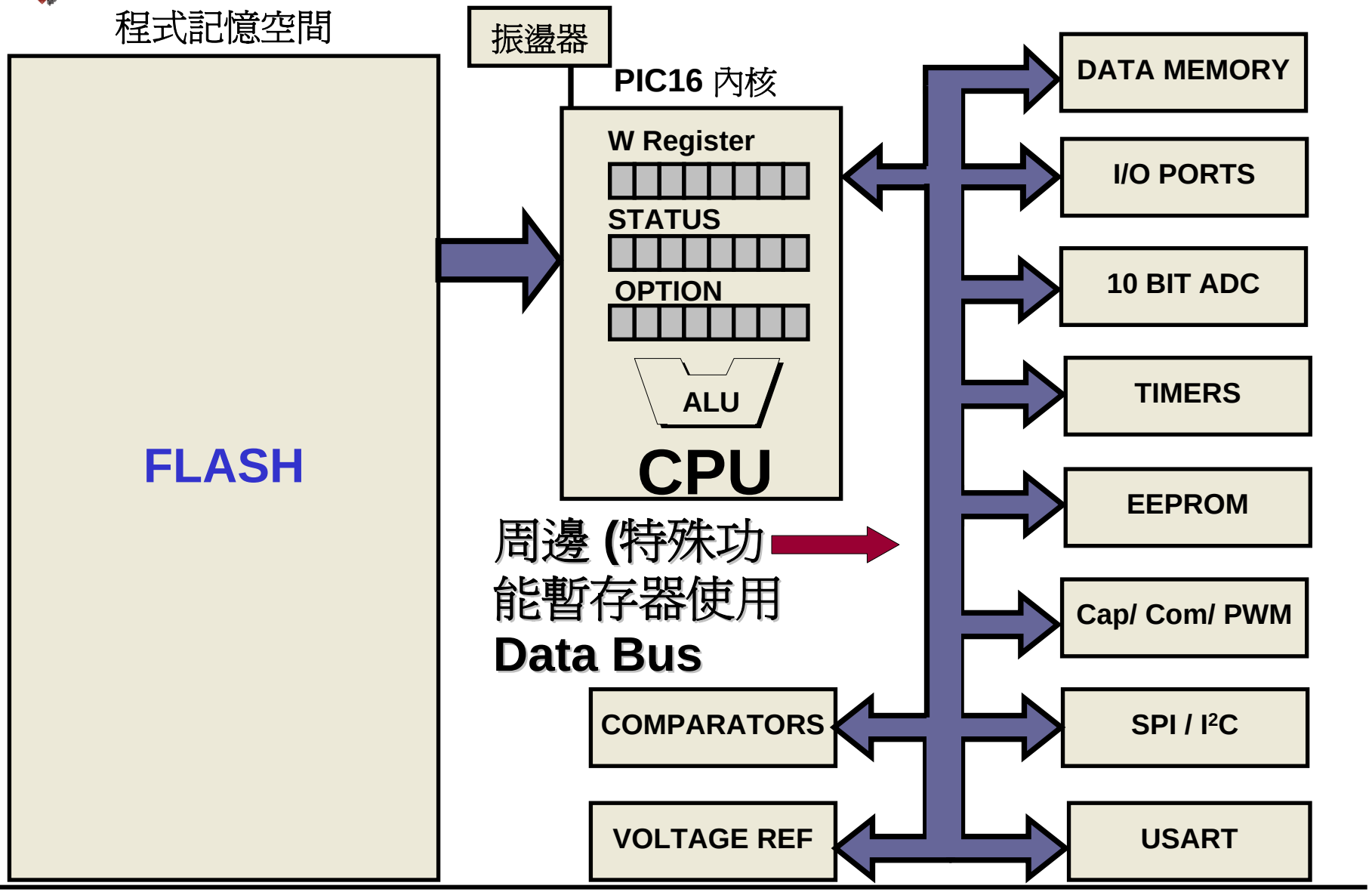
Mid-Range PIC 方塊圖 (一)



CPU 是大腦

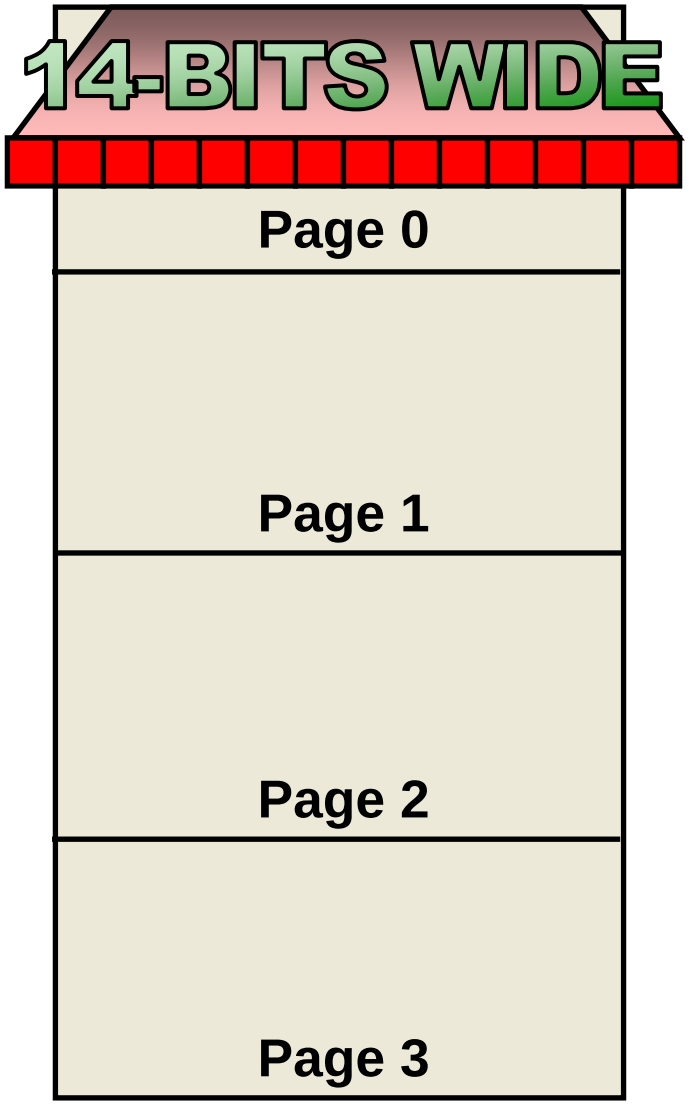
掌控所有程式的
執行、邏輯運算、
數學處理及資料
的搬移

Mid-Range PIC 方塊圖 (二)



Flash Program Memory

程式記憶空間

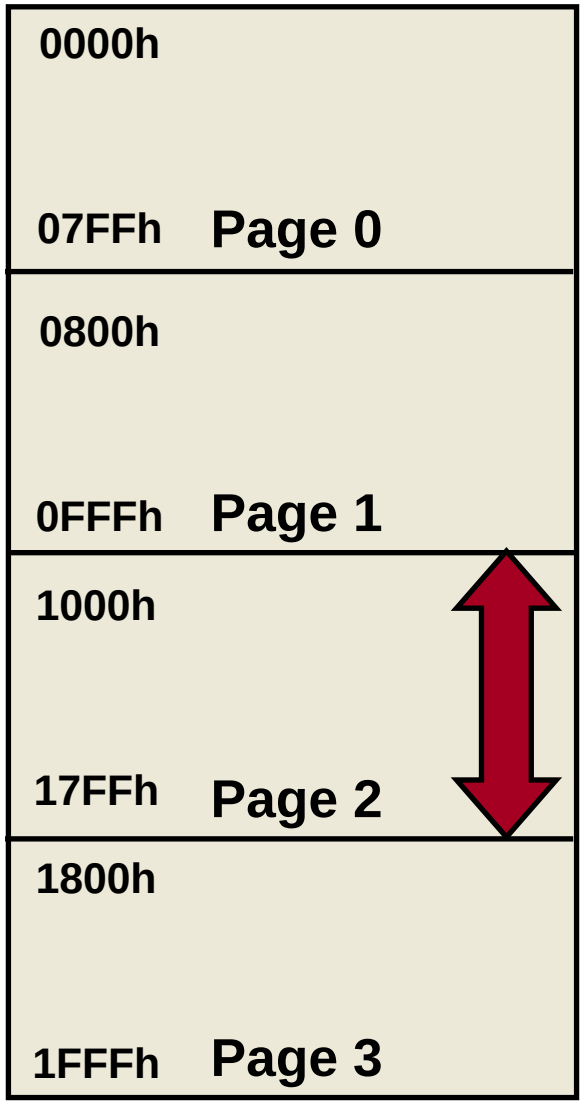


- 主要作爲程式的儲存
- 記憶容量隨元件而定
- **PIC16F** 指令寬度爲 **14-bits**
  - **1** 個指令佔用一個記憶位址
- **PIC16F** 系列元件因指令長度關係，執行 **CALL & GOTO** 指令時，視野只有 **2K (11-bit Address)** 的範圍，超過 **2K** 容量就會有**Page** 的切換動作
  - **PIC16F887** 有 **8KW** 的程式，所以有 **4** 個 **Page**

Flash Program Memory

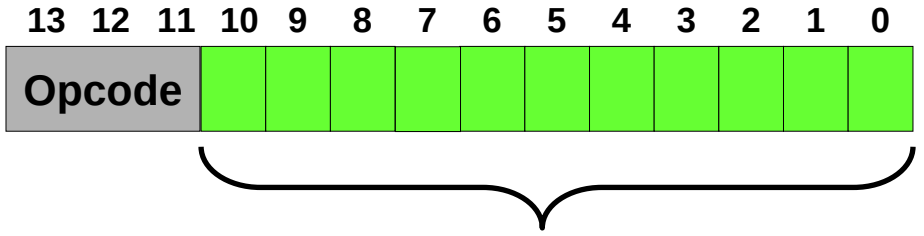
爲何需要切換程式的 Page

程式記憶空間



- 在 **14-bit** 的指令寬度下，只有 **11-bits** 的有效位址可以放在單一指令裡
- **GOTO K** 的解析
  - **0 < K <= 2047**
  - **K → PC<10:0>**
  - **PCLATH<4:3> → PC<12:11>**

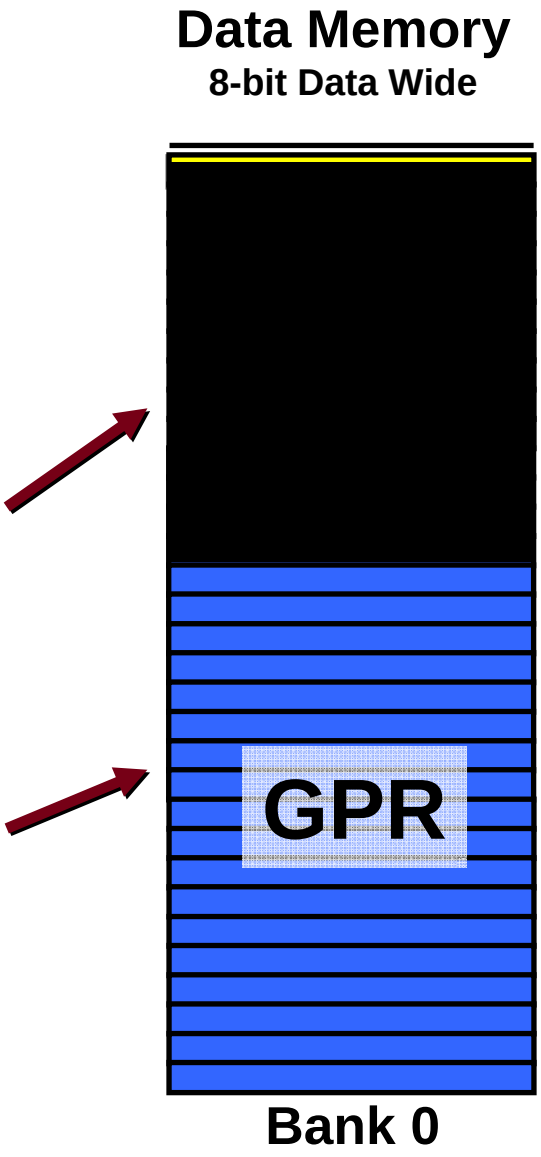
GOTO & CALL 指令的編碼方式



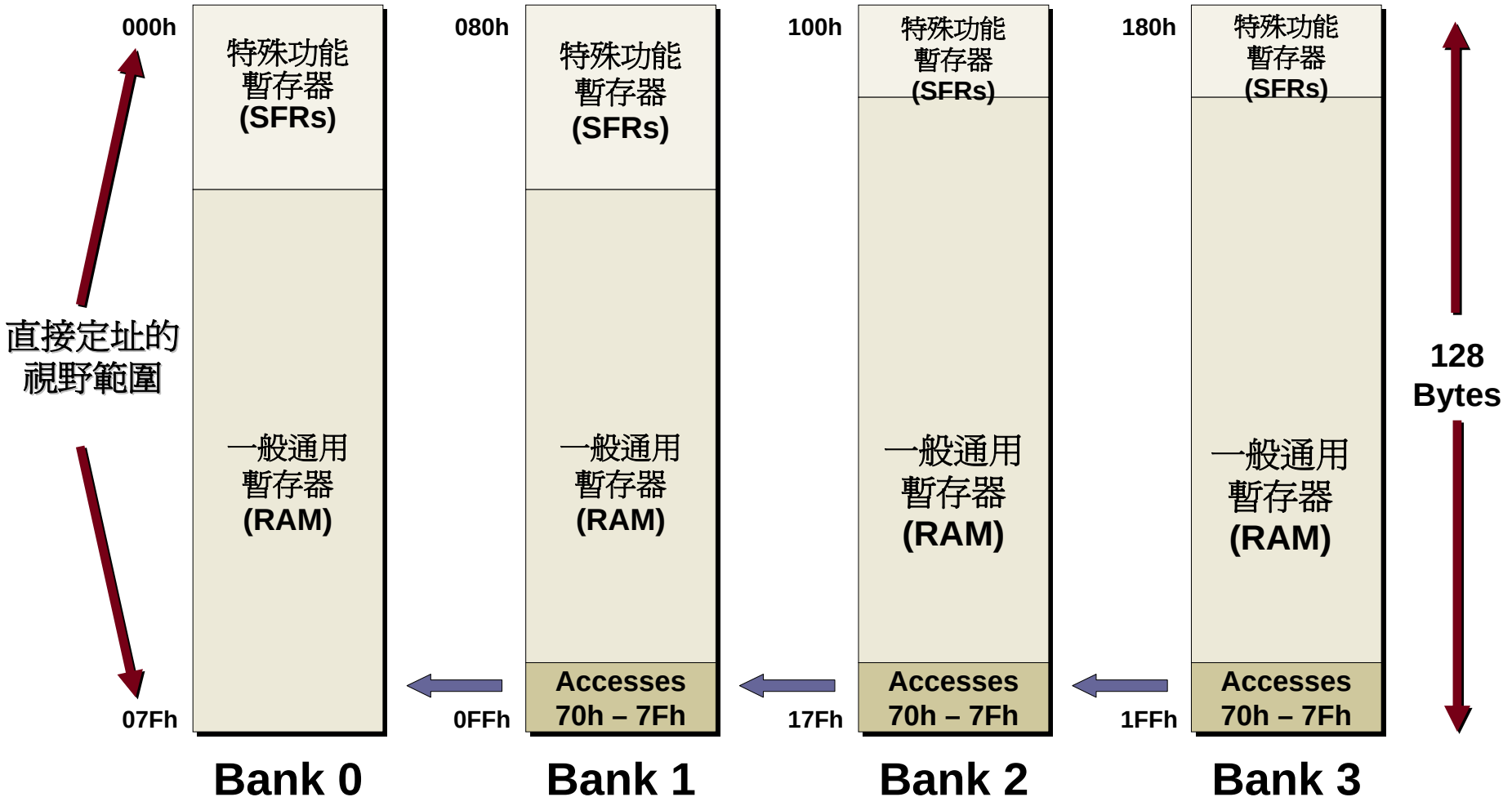
$2^{11} = 2048$ addresses

SRAM Data Memory

- 最大有 **512 Bytes** 空間，每 **128 Bytes** 被劃分爲一個 **Bank**
  - **Bank** 數目取決於元件
  - 最多有 **4 banks (max)**
- **2** 種類型的暫存器族群:
  - ***Special Function Registers*** **(SFR)**
    - **CPU** 控制 **&** 週邊暫存器
  - ***General Purpose Registers*** **(GPR)**
    - 用來做資料儲存或變數使用

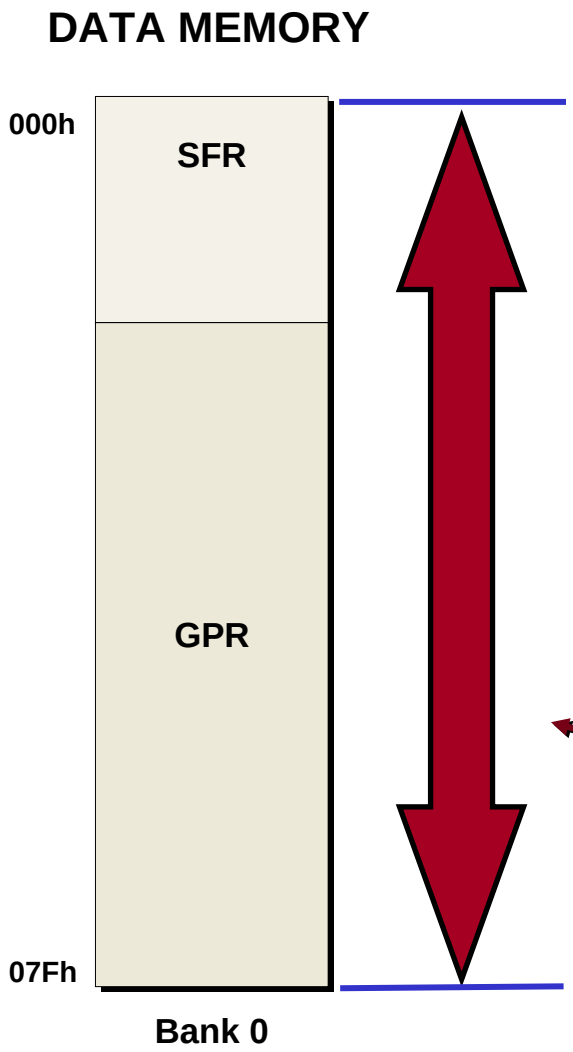


Data Memory Banks

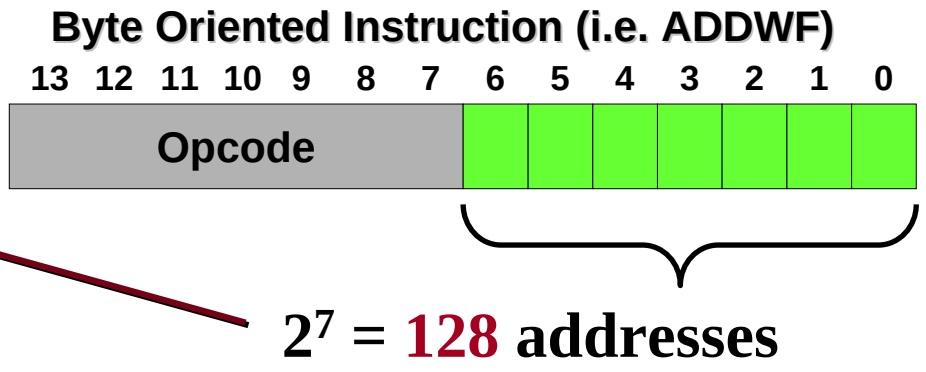


Data Memory Banks

會何需要做 Bank 的切換



- 在 **14 bit** 的指令寬度裡，使用直接定址模式時，只有 **7 bits** 可以指定 **RAM** 的直接位址；其視野範圍到 **128 Bytes (0x00 ~ 0x7F)**
  - 狀態暫存器 **(STATUS)** 有兩個位元 **(RP0, RP1)** 用來選擇其中一個 **Bank** 來操作



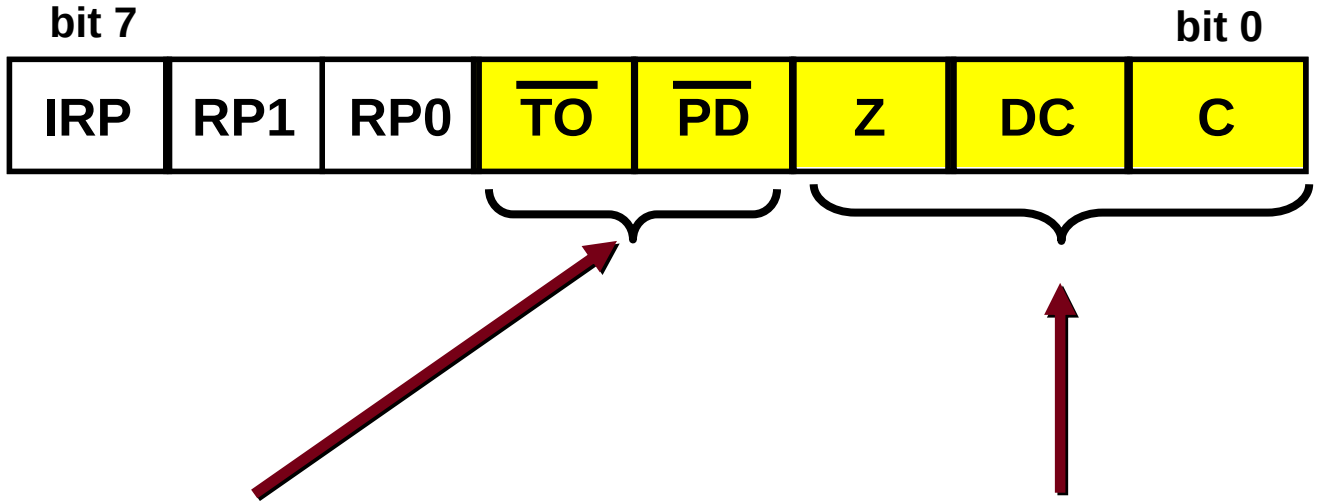
Data Memory SFRs

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

詳細的資料請參考該元件的 **Data Sheet**

狀態暫存器
Status Register



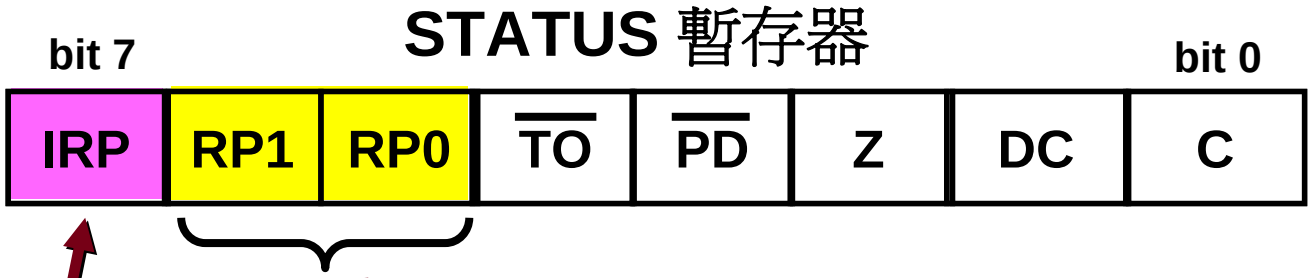
Reset Status Bits

$\overline{PD}$	電源狀態指示位元 (平時爲1，執行 Sleep 指令後設爲 0)
$\overline{TO}$	WDT 溢時狀態旗號 (電源開啓時設爲 1，WDT 溢時清爲 0)

ALU Status Bits

C	進位旗號 (Byte Overflow?)
DC	半進位旗號 (Nibble (4 bit) Overflow?)
Z	零旗號 (ALU 的結果爲 0)

Bank 的選擇位元



Bank 選擇位元
(索引定址模式下使用)

Bank 選擇位元
(直接定址模式下使用)

IRP	
0	BANKS 0 & 1
1	BANKS 2 & 3

RP1	RP0	
0	0	BANK0
0	1	BANK1
1	0	BANK2
1	1	BANK3

PIC16 組合語言指令集

- **35** 個指令
- 除了會改變程式路徑的指令外，其它都是單一指令週期執行時間

Byte Oriented Operations		
addwf	f,d	Add W and f
andwf	f,d	AND W with f
clrf	f	Clear f
clrw	-	Clear W
comf	f,d	Complement f
decf	f,d	Decrement f
decfsz	f,d	Decrement f, Skip if 0
incf	f,d	Increment f
incfsz	f,d	Increment f, Skip if 0
iorwf	f,d	Inclusive OR W with f
movf	f,d	Move f
movwf	f	Move W to f
nop	-	No Operation
rlf	f,d	Rotate Left f through Carry
rrf	f,d	Rotate Right f through Carry
subwf	f,d	Subtract W from f
swapf	f,d	Swap nibbles in f
xorwf	f,d	Exclusive OR W with f

Bit Oriented Operations		
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
Literal and Control Operations		
addlw	k	Add literal and W
andlw	k	AND literal with W
call	k	Call subroutine
clrwdt	-	Clear Watchdog Timer
goto	k	Go to address
iorlw	k	Inclusive OR literal with W
movlw	k	Move literal to W
retfie	-	Return from interrupt
retlw	k	Return with literal in W
return	-	Return from Subroutine
sleep	-	Go into standby mode
sublw	k	Subtract W from literal
xorlw	k	Exclusive OR literal with W

Header Files

pic.h

chip_select.h

pic16f887.h

pic.h

- 在程式中，加入 **#include <pic.h>**
- 查詢元件並加入元件的定義檔
- 定義特殊指令的巨集
  - **#define SLEEP( ) asm(“sleep”)**
  - **#define NOP( ) asm(“nop”)**
  - **#define CLRWDT( ) asm(“clrwdt”)**
    - 所以要進入睡眠省電模式只要輸入巨集指令
      - **SLEEP( );**
- 定義 **Configuration Word** 的巨集
- 定義 **ID Locations** 的巨集
- 定義 **EEPROM** 的巨集

pic.h

設定 Config. Word

- **CONFIG 的巨集宣告 (pic.h)**

```
#define  __CONFIG(x)  asm("\tpsect config,class=CONFIG,delta=2");\
                      asm("\tdw "___mkstr(x))
```

- **注意: pic.h 只定義 config 的巨集，名稱卻是在 pic16f877.h 裡定義的 (因爲各元件 Config. 不同)**

範例：**PIC16F877A (只有一個 Config. Word @0x2007)**

```
#include  <pic.h>

__CONFIG  ( HS & WDTDIS & PWRTEN & BORDIS & LVPDIS & UNPROTECT );

void main(void)
{
        //    ... your code
}
```

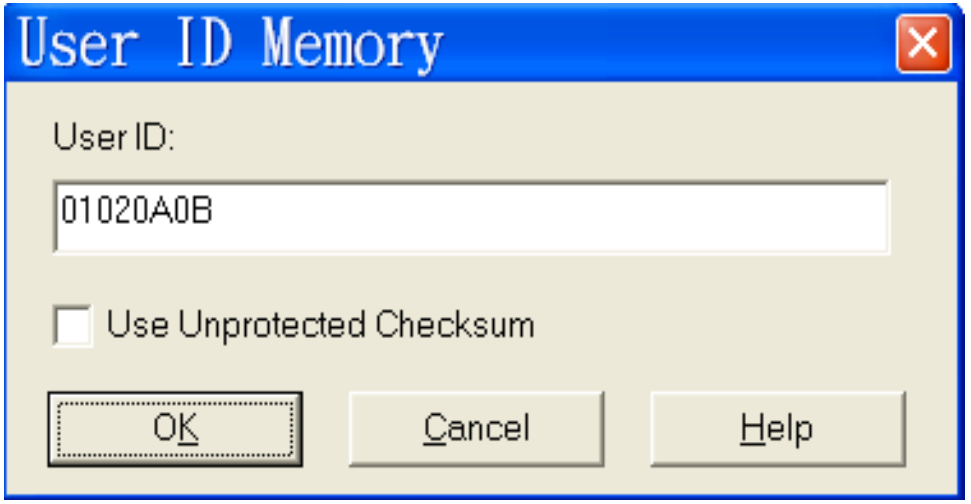
pic.h

設定 ID Locations

- **ID Locations 的巨集宣告 (pic.h)**
  - 使用 _ _ **IDLOC** 巨集
  - 輸入爲 **16** 進制的 **4** 個字元 **(0 ~ F)**

範例：PIC16F887

```
#include  <pic.h>

    __IDLOC  ( 12AB );

void main(void)
{
        //     ... your code

}
```



pic.h
設定 EEPROM

- **EEPROM 的巨集宣告 (pic.h)**
  - 使用 _ _ **EEPROM_DATA** 巨集
  - 每個巨集最多可輸入八個 **Bytes**

PIC16F887 範例：

```
__EEPROM_DATA    (0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07);
__EEPROM_DATA    (0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f);
__EEPROM_DATA    ('H','i','-','T','e','c','h',0x00,);
```

pic.h 會再含入 chip_select.h

- **pic.h & chip_select.h 的檔案在哪裡?**
  - **C:\Program Files\HI-TECH Software\PICC\9.82\include**

```
// Select device-specific header file
#ifndef _CHIP_SELECT_H_
#define _CHIP_SELECT_H_

#ifdef _16C558
#ifdef _LEGACY_HEADERS
#include <legacy/pic1655x.h>
#else
#include <pic16c558.h>
#endif
#endif

#ifdef _16C99C
#include <legacy/pic1699.h>
#endif
:
:

#ifdef _16F887      ← 此選用零件編號是由 MPLAB IDE 在編譯時傳入的
#ifdef _LEGACY_HEADERS
#include <legacy/pic16f887.h>
#else
#include <pic16f887.h>
#endif
#endif
```

chip_select.h
依據所選定的元件
來決定使用那一個
元件的定義檔案

關於特殊功能暫存器的的宣告

- **C** 可以直接使用所有的 **SFR**
  - **C** 使用 @ 的絕對定址符號定義 **SFR** 的位址
  - 使用此種宣告方式，該位址並不會被保留
  - 一但使用上述的定義後，在 **C** 程式中就可以直接存取 **SFR** 暫存器

- 有關 **PIC** 的特殊功能暫存器的定義內容已包含在編譯器所提供的 **header files** 裡
  - 使用時需將 **“#include <pic.h> “**加入程式中
  - **pic.h ➔ chip_select.h ➔ pic16f887.h**

pic16f887.h 裡的定義 (SFRs)

```
// Register: FSR
volatile unsigned char       FSR          @ 0x004;
// bit and bitfield definitions

// Register: PORTA
volatile unsigned char       PORTA        @ 0x005;
// bit and bitfield definitions
volatile bit RA0             @ ((unsigned)&PORTA*8)+0;
volatile bit RA1             @ ((unsigned)&PORTA*8)+1;
volatile bit RA2             @ ((unsigned)&PORTA*8)+2;
volatile bit RA3             @ ((unsigned)&PORTA*8)+3;
volatile bit RA4             @ ((unsigned)&PORTA*8)+4;
volatile bit RA5             @ ((unsigned)&PORTA*8)+5;
volatile bit RA6             @ ((unsigned)&PORTA*8)+6;
volatile bit RA7             @ ((unsigned)&PORTA*8)+7;
#ifndef _LIB_BUILD
volatile union {
   struct {
      unsigned          RA0          : 1;
      unsigned          RA1          : 1;
      unsigned          RA2          : 1;
      unsigned          RA3          : 1;
      unsigned          RA4          : 1;
      unsigned          RA5          : 1;
      unsigned          RA6          : 1;
      unsigned          RA7          : 1;
   };
} PORTAbits @ 0x005;
#endif
```

← 定義 **FSR** 暫存器及其絕對位址

← 定義 **PORTA** 及其絕對位址

定義 **PORTA** 裡各個位元的絕對位址

定義 **PORTA** 的位元結購 **(**語法與 **C18** 相容**)**

pic16f887.h 裡對 Config. 定義

- // Configuration mask definitions

- // Config Register: CONFIG1
- #define CONFIG1 0x2007

- // Oscillator Selection bits
- // RC oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, RC on RA7/OSC1/CLKIN
- #define FOSC_EXTRC_CLKOUT 0xFFFF
- // RCIO oscillator: I/O function on RA6/OSC2/CLKOUT pin, RC on RA7/OSC1/CLKIN
- #define FOSC_EXTRC_NOCLKOUT 0xFFFE
- // INTOSC oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN
- #define FOSC_INTRC_CLKOUT 0xFFFD
- // INTOSCIO oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN
- #define FOSC_INTRC_NOCLKOUT 0xFFFC
- // EC: I/O function on RA6/OSC2/CLKOUT pin, CLKIN on RA7/OSC1/CLKIN
- #define FOSC_EC 0xFFFB
- // HS oscillator: High-speed crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
- #define FOSC_HS 0xFFFA
- // XT oscillator: Crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
- #define FOSC_XT 0xFFF9
- // LP oscillator: Low-power crystal on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
- #define FOSC_LP 0xFFF8
- // Watchdog Timer Enable bit
- // WDT enabled
- #define WDTE_ON 0xFFFF
- // WDT disabled and can be enabled by SWDTEN bit of the WDTCON register
- #define WDTE_OFF 0xFFF7

PIC16F887 的 Configuration 設定

- **PIC16F887 有兩個 Config. 暫存器要設定**
  - **CONFIG1 (0x2007) 及 CONFIG2 (0x2008)**
  - **Configuration 的設定巨集：_ _ CONFIG (x)**

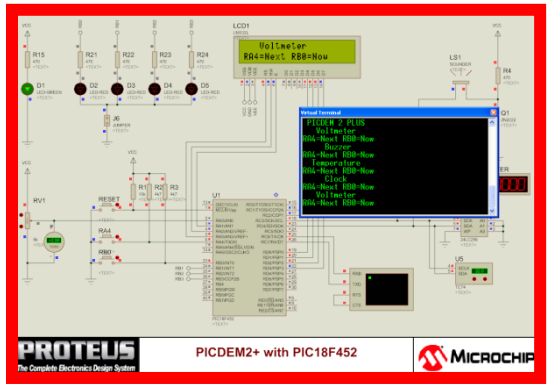
- **設定範例:**

```
// ************ 設定 PIC16F887 Configuration Bits  ******
// 有關此設定的定義字請參考 pic16f887.h 檔的說明
//

__CONFIG ( FOSC_XT & WDTE_OFF & PWRTE_OFF & MCLRE_ON & CP_OFF & CPD_OFF
           BOREN_ON & IESO_OFF & FCMEN_OFF & LVP_OFF );

__CONFIG ( BOR4V_BOR21V & WRT_OFF );
```

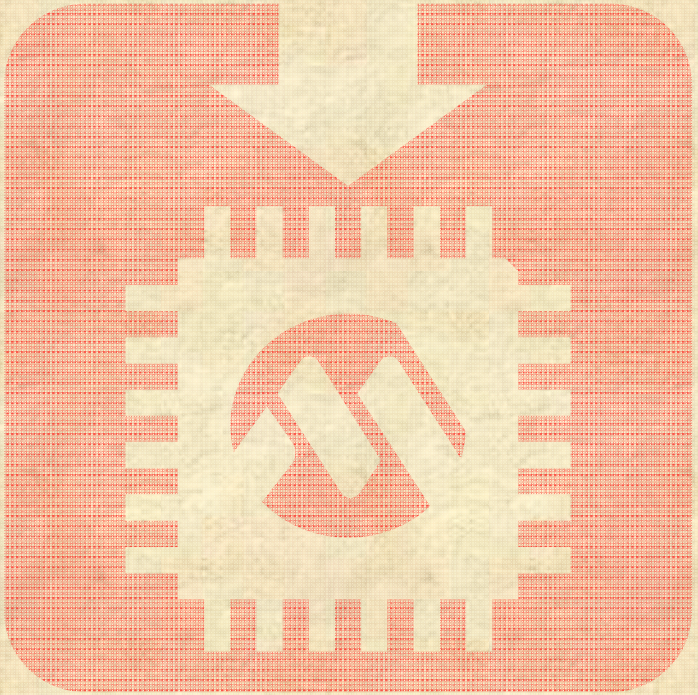
Labcenter Proteus VSM
(Virtual System Modelling)

- 以軟體方式模擬實際的硬體電路，可支援所有的 PIC MCU
- 完整的嵌入式系統模擬，支援類比及數位電路
- 可以對韌體及硬體電路做實質的除錯
- 測試及驗證所設計的硬體電路，降低硬體與韌體設計的風險

www.labcenter.com



Schematic simulation of Microchip PICDEM 2 board



Lab Exercise 1

建立一個 C 的範本程式

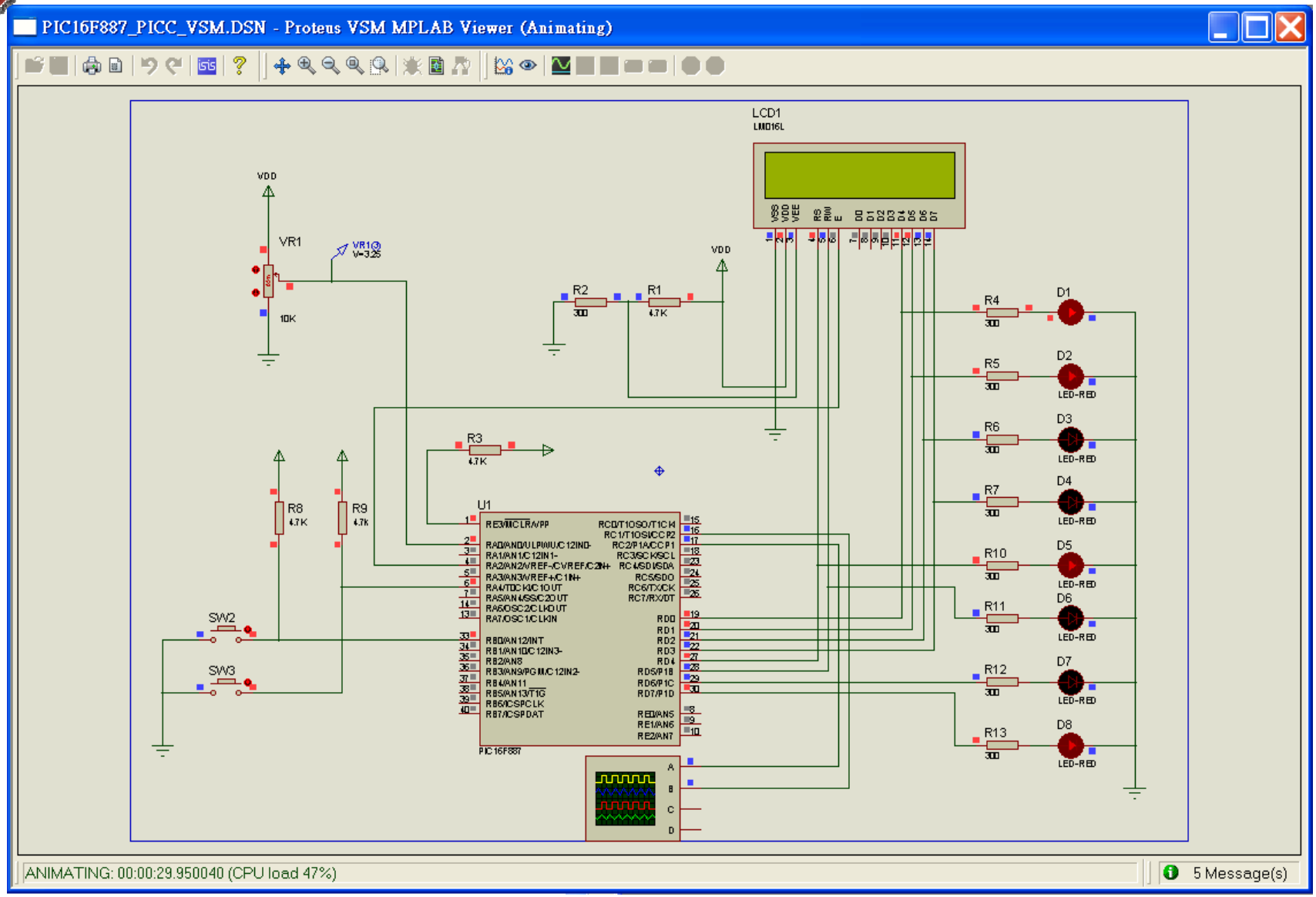
設定 Config. EEPROM & ID

LAB1 – Template File.c

- 本實驗將使用 **Proteus** 來模擬硬體電路，實驗中的電路已經畫好，只要開啓本練習的專案即可載入
  - **..\Hi-Tech PICC v2.0 LAB\lab1**

    LAB1 - Config Setting.mcp
- **Proteus** 對 **PIC16** 的元件支援
  - http://www.labcenter.com/products/pic16.cfm
- **Proteus** 的 **Demo** 軟體下載
  - http://www.labcenter.com/download/prodemo_download.cfm
- 或可以使用 **ICD3 + APP001 + PIC16F887**

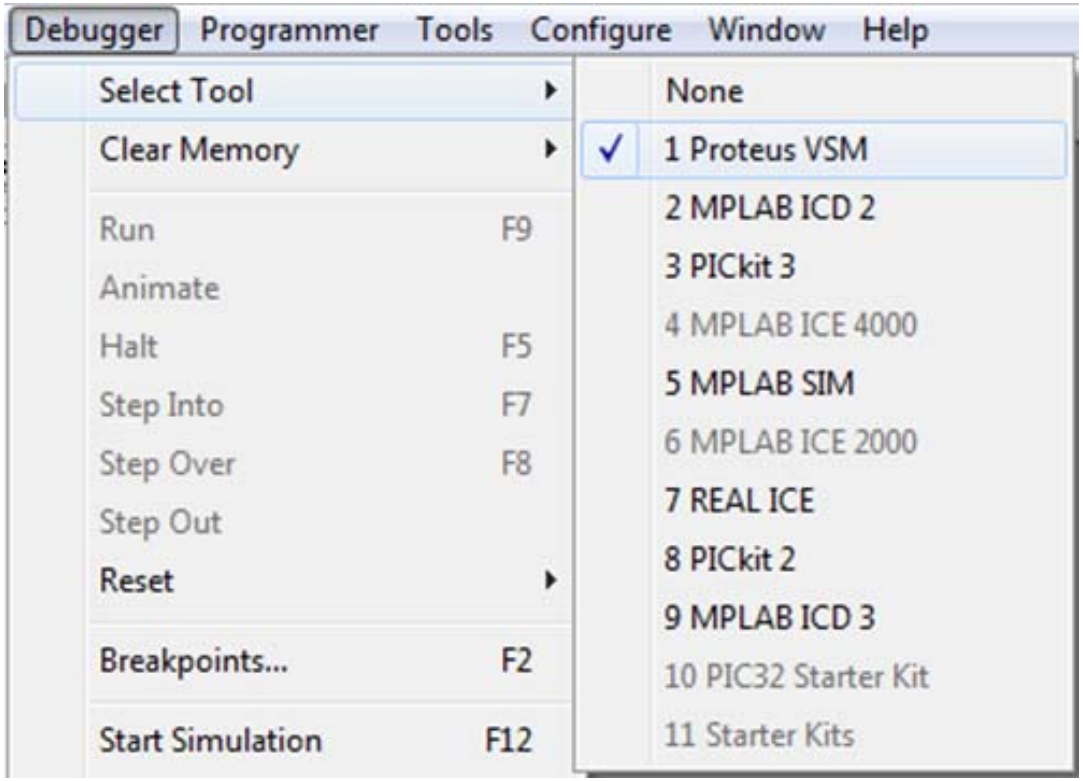
用 Proteus 畫的 LAB1 電路圖



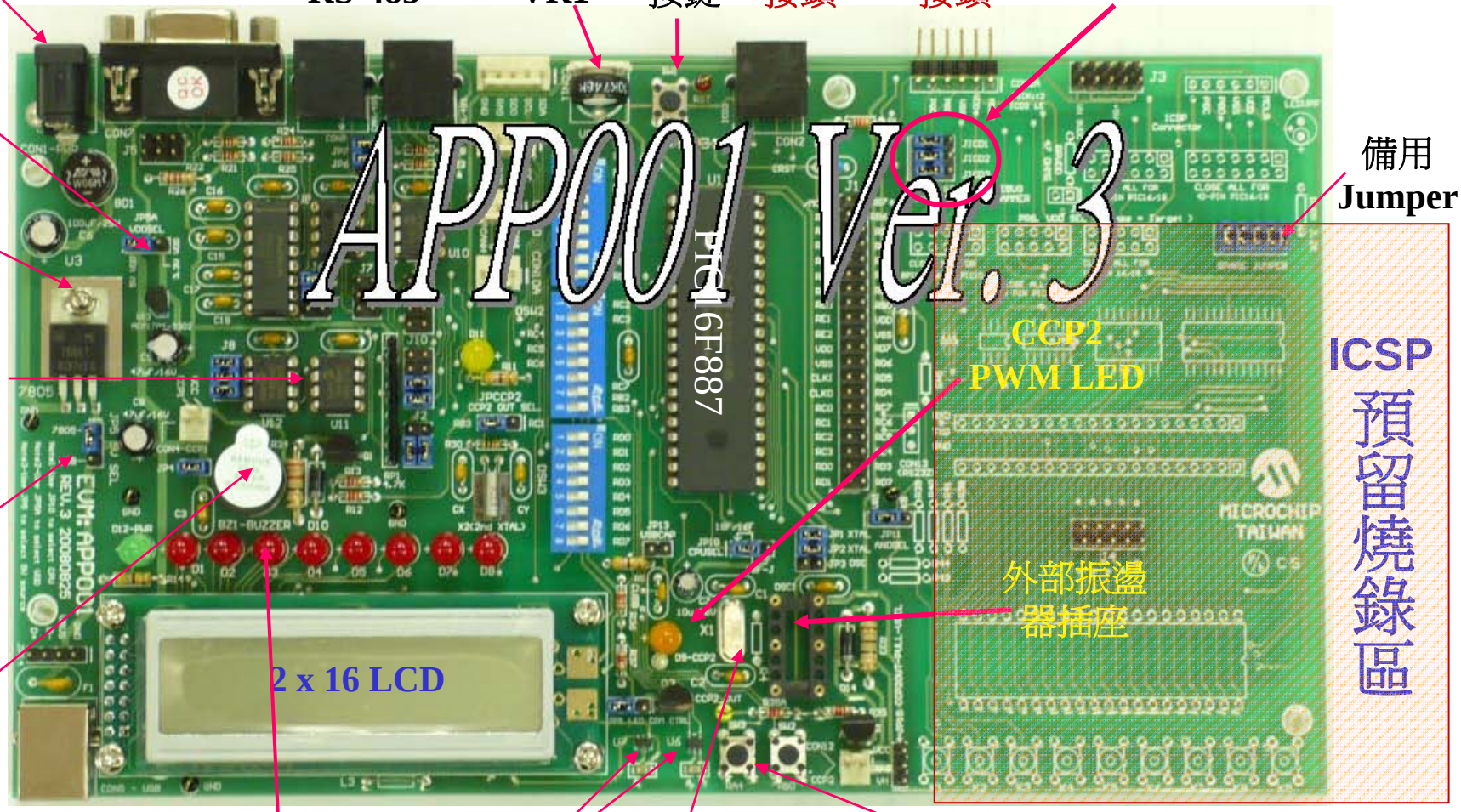
LAB1 使用 Proteus

Creating an MPLAB® C Based Project

- 要先安裝 **Proteus** 的軟體 **(試用版 PRODEMO.exe)**
- 在 **MPLAB IDE** 下啓用 **Proteus VSM**
  - Debugger ▸ Select Tool ▸ Proteus VSM
  - 在 Proteus 視窗下，用 Open File 開啓 **"PIC16F887_PICC_VSM.DSN"**



認識 APP001 REV.3a 實驗板



9V
電源輸入

RS-232

CAN
RS-485

AD 輸入
VR1

Reset
按鍵

ICD3
接頭

PICkit3
接頭

Debug & Prog.
選擇 Jumper

備用
Jumper

5V & 3.3V
供電選擇

5V
電源穩壓

I2C & SPI
EEPROM

9V & USB
供電選擇

CCP1
蜂鳴器

USB
接頭

PORTD
LED x8

溫度
感測器

16MHz
Crystal

按鍵開關
SW3 & SW2

使用 APP001 REV.3a 實驗板

- APP001 可以直接使用 ICD 3 或 PICkit 3
  - ICD3 連接 CON2，PICkit3 連接 CON2A
- 板子要單獨供電
  - JP5 選擇 USB 或 9V Adapter 供電
  - JP5A 再選擇 7805 的 5V 供電或 3.3V 供電
  - 上電前請確認 MCU 的工作電壓
- 三個 DIP SW 可以選擇是否使用板子上所提供的周邊，MCU 的腳位也可以自 J1 取得。

LAB1 – Template File.c

- 程式編譯成功後，開啓 **Configuration Bits**、**ID Memory** 及 **EEPROM** 三個視窗觀察其設定值
- 本實驗使用 **8MHz Internal FRC** 做爲振盪源請先用 **MPLAB SIM** 先做 **Delay** 的計算
- **Delay_1mS** 是如何得知爲 **1mS Delay ?**
- 啓用 **Proteus** 或使用 **APP001** 實驗板來觀察 **LED** 及 **SW2** 的動作

資料型別
變數種類
I/O 的定址

資料型別

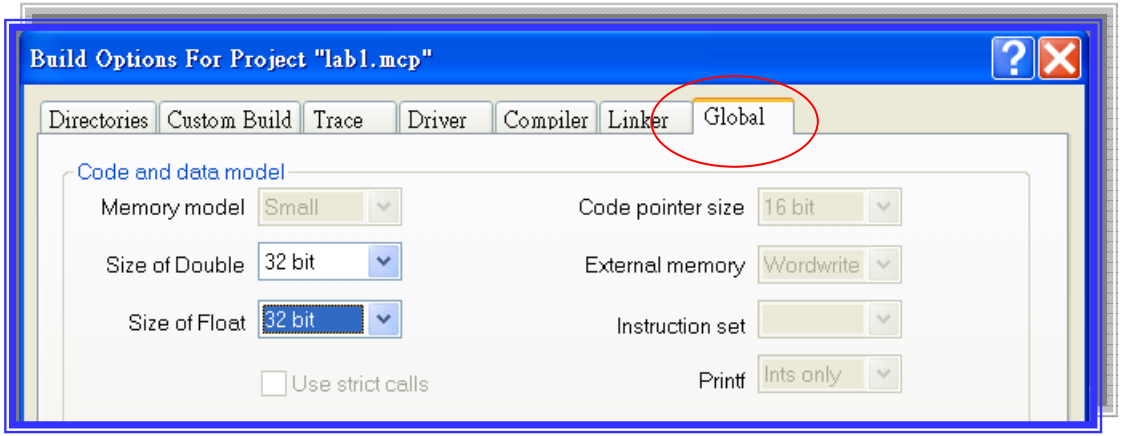
資料型別	記憶空間 (bits)	數值範圍
bit	1	boolean
signed char	8	-128 to 127 [a]
unsigned char	8	0 to 255
short	16	-32768 to 32767
unsigned short	16	0 to 65535
int	16	-32768 to 32767
unsigned int	16	0 to 65535
long	32	-2,147,483,648 to +2,147,483,647
unsigned long	32	0 to 4,294,967,925
float	24 or 32 [b]	real
double	24 or 32 [b]	real

a. char 的內定資料型別爲 unsigned
b. double & float 內定爲 24-bit 資料格式

Floating point 使用 IEEE 754 32-bit 的資料格式

內定的資料型別

- 標準的資料型別
  - 除了 **Char** 內定爲 **unsigned** 的型別，其餘若不指定均爲有號數的型別
  - **float**、**double** 內定 **24-bits** 的浮點數
  - 在 **Project Options** 下的 **Project** 對話視窗，選擇 **“Global”** 來指定爲 **32 bits** 浮點數



變數的等級

- ***Auto*** **–** 又稱爲 區域變數。一般是只在函數內宣告的變數，因函數的叫用而生，函數返回後而消失，屬於暫態的變數

- ***Static*** **- ANSI-C** 定義 靜態變數 是在函數宣告但保有變數的位址，且該位址不因該函數返回後而消失

- **Global –** 在 **main( )** 以外所宣告的變數稱之爲 公共變數，可以用來做函數之間的變數傳遞

- ***Extern*** **–** 指示該變數已經被其它程式宣告過，可以用 **Extern**來擴展視野並使用該變數名稱，但不可重複宣告

一般資料型態 (const)

- ***const*** **- const** 的保留字是用來告訴編譯器該內容的資料型態爲固定不變的常數值。任何企圖修改這個常數值的動作編譯器將會產生警告訊息。
  - **const** 的宣告其內容會被擺放在 **Program** 空間。

範例：給予陣列 tableDef[ ] 初始常數值，並放置在程式記憶位址 @0x100

```
const char tableDef[ ] @ 0x100 = { 0, 1, 2, 3, 4};
```

一般資料型態 (volatile)

- ***volatile*** **– volatile** 的保留字是用來告訴編譯器該變數的內容不一定要經過程式的執行後才會改變。它可以告訴編譯器在做最佳化處理的時候無需將該變數簡化
  - **SFR** 暫存器可以被硬體改變其內容，要加入 **volatile**
  - 中斷所使用到的變數或中斷函數會修改到的變數

範例：

```
volatile static unsigned int TACTL;
```

特殊資料型態

- ***persistent*** – 爲一個特殊的保留字，用來告訴編譯器該變數在重新啓動時(startup)無須被清除爲零。一般而言，使用此方式所宣告的變數會被存放在另外的節區。

  範例： static persistent int intvar; /* 必須加上 static */

- ***near*** **– 用來指定變數擺放在不用切換 bank 的共用區域 (0x70 ~ 0x7F)**

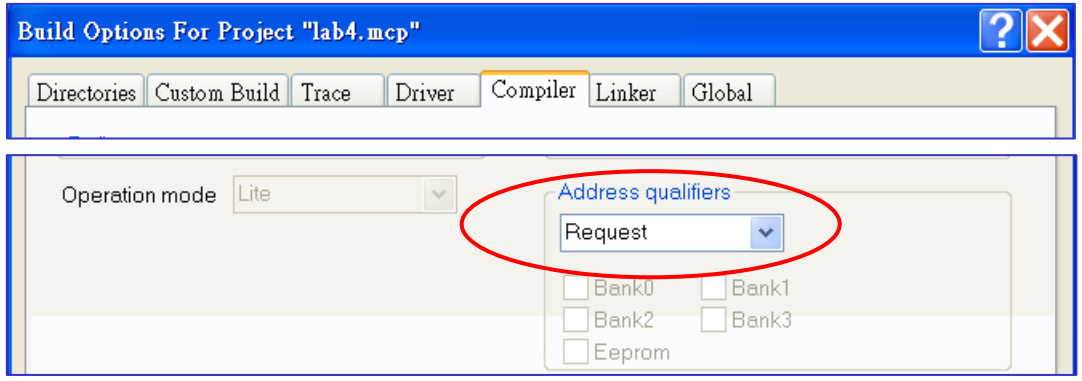
  範例 ：

  near unsigned char Time_Flag;

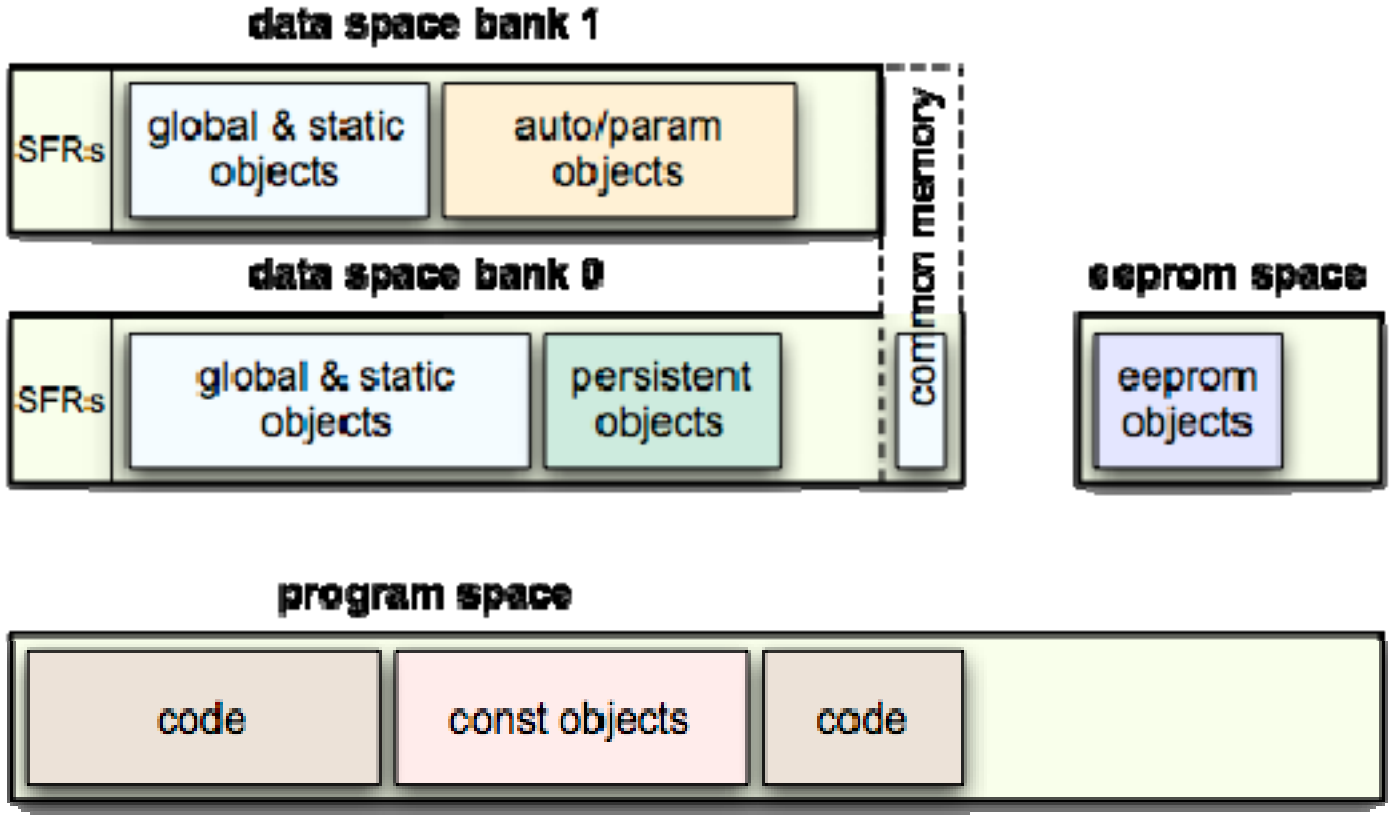
特殊資料型態

- **Bank0, bank1, bank2 and bank3 –** 另一種特殊的變數型態的宣告，用來指定變數存分別存放在 **RAM Bank 1, RAM Bank 2** 和 **RAM Bank 3** 。變數未加以指定 **RAM Bank** 時，基本上是會被放置在 **RAM Bank 0**。

  - **PICC** 編譯器對 **bankx** 的安排內定是忽略的 **(Ignore)**，要啓用此項指定需在編譯時加以設定爲 **“Request”**。
  - **Project Options → Project** 對話視窗下，選擇 **Compiler** 選項



記憶體的安排


data space bank 1
SFRs
global & static objects
auto/param objects
common memory
data space bank 0
SFRs
global & static objects
persistent objects
eeprom space
eeprom objects
program space
code
const objects
code

絕對位址變數

- ***Absolute*** **–** 絕對位址的定位方式可使用**“ @ address”**的方式。 編譯器不會保留該位址，使用時需注意該位址是否有其它的變數重複使用(編譯器與連結器不會檢查該絕對位址是否有與其它變數位址有重複)

範例 ：

volatile unsigned char PORTB @ 0x06 // **#include <pic.h>**

常數宣告在程式記憶體裡，並使用絕對定址方式

const int settings[ ] @ 0x200 = { 1, 5, 10, 50, 100 };

絕對定址使用範例

- 定義特殊功能暫存器 **(SFRs)** 的絕對位址 **(pic16f887.h)**

```
/*  bank 0 registers */
    volatile unsigned char      INDF            @ 0x00;
    volatile unsigned char      TMR0            @ 0x01;
    volatile unsigned char      PCL             @ 0x02;
    volatile unsigned char      STATUS          @ 0x03;
    volatile unsigned char      FSR             @ 0x04;
                                :
                                :
/*  bank 1 registers */
    volatile unsigned char      OPTION_REG      @ 0x81;
    volatile unsigned char      TRISA           @ 0x85;
    volatile unsigned char      TRISB           @ 0x86;
    volatile unsigned char      TRISC           @ 0x87;
                        :
```

使用位元變數定址方式

- 使用位元的絕對位址方式
  - **Hi-Tech C** 的專用語法
  - 用來定義特殊功能暫存器**(SFR)**內的獨立位元
  - **SFR**內的位元定義 **–** 需要使用 **pic16f887.h**
- 使用 **bit** 定義位元變數
  - **Hi-Tech C** 的專用語法
  - 最簡單方便的使用方式
- 位元結構方式
  - **ANSI C** 標準使用方式
  - 與 **MPLAB C18** 相容

PORTA 位元 在 pic16f887.h 的定義

```
// Register: PORTA
volatile unsigned char       PORTA          @ 0x005;          // PORTA 的宣告及指定位址
// bit and bitfield definitions
volatile bit RA0             @ ((unsigned)&PORTA*8)+0;
volatile bit RA1             @ ((unsigned)&PORTA*8)+1;
volatile bit RA2             @ ((unsigned)&PORTA*8)+2;
volatile bit RA3             @ ((unsigned)&PORTA*8)+3;
volatile bit RA4             @ ((unsigned)&PORTA*8)+4;
volatile bit RA5             @ ((unsigned)&PORTA*8)+5;
volatile bit RA6             @ ((unsigned)&PORTA*8)+6;
volatile bit RA7             @ ((unsigned)&PORTA*8)+7;
#ifndef _LIB_BUILD
volatile union {
   struct {
      unsigned          RA0          : 1;
      unsigned          RA1          : 1;
      unsigned          RA2          : 1;
      unsigned          RA3          : 1;
      unsigned          RA4          : 1;
      unsigned          RA5          : 1;
      unsigned          RA6          : 1;
      unsigned          RA7          : 1;
   };
} PORTAbits @ 0x005;
```

Hi-Tech 專用語法，
用來宣告單一位元的變數

ANSI C 標準語法，
使用位元結構方式來宣告。
此方式與 **C18** 相容

關於位元的絕對位址的設定

- 使用絕對位址定址方式
  - 位元定址的起始位址爲 **0**
  - 公式 **: (8 bits * SFR** 的位址 **) +** 偏移位元

範例： 存取 Z 旗號

```
static unsigned char STATUS   @ 0x03 ;
static bit  ZERO  @  (unsigned) & STATUS*8 + 2 ;
```

存取 PORTA 的 RA5

```
static volatile unsigned char PORTA @ 0x05 ;
static volatile bit RA5 @ (unsigned) & PORTA*8 + 5 ;
```

使用 SFR 位元變數

- **HI-TECH C** 已將各相關的位元名稱及相關位置定義完成 **(pic16f887.h)**
  - /*STATUS bits */
  - static volatile bit IRP @ (unsigned)&STATUS*8+7;
  - static volatile bit RP1 @ (unsigned)&STATUS*8+6;
  - static volatile bit RP0 @ (unsigned)&STATUS*8+5;
  - static volatile bit TO @ (unsigned)&STATUS*8+4;
  - static volatile bit PD @ (unsigned)&STATUS*8+3;
  - static volatile bit ZERO @ (unsigned)&STATUS*8+2;
  - static volatile bit DC @ (unsigned)&STATUS*8+1;
  - static volatile bit CARRY @ (unsigned)&STATUS*8+0;
- 若要判斷 **Z** 旗標，只要
  - **If (ZERO) ……**

位元 (bit) 變數定址
(自行使用 bit 定義位元變數)

- 可以使用 **bit** 宣告爲單獨的位元變數
- **Hi-Tech C** 會自動整理位元變數擺在同一個 **Byte** 裡以節省空間
- 建議使用此種簡單的方式

範例 ： 利用 bit 定義位元變數

```
static bit Count_Flag ;
static bit Buzzer_1_Flag ;

Buzzer_1_Flag = 1 ;
if (Count_Flag) Count_Flag = 0 ;
```

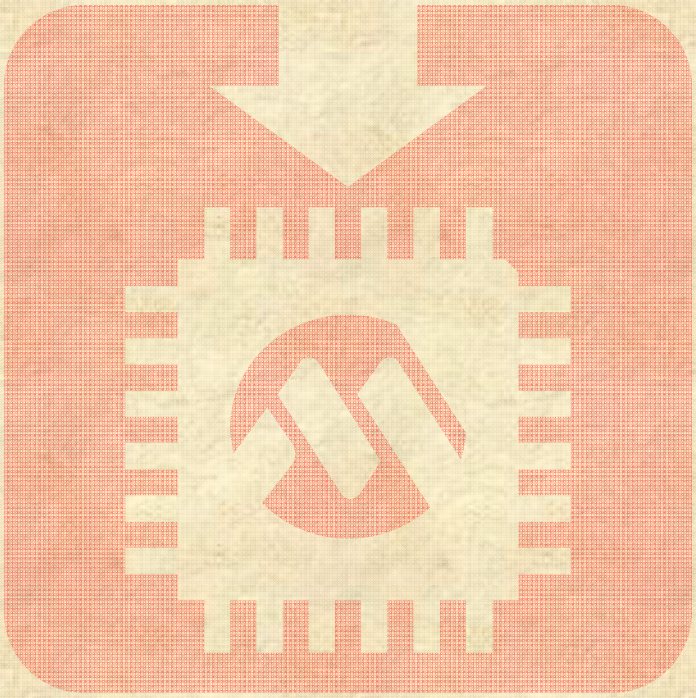
使用位元結構方式
(與 C18 & C30 相容)

將 **PORTD** 的 **bit3 & bit5** 設定爲輸出腳並設定輸出爲 **High**：

```
TRISDbits.TRISD3 = 0;          // 設定 RD3 爲輸出腳功能
TRISDbits.TRISD5 = 0;          // 設定 RD3 爲輸出腳功能
PORTDbits.RD3 = 1 ;            // 將 RD3 輸出 High
PORTDbits.RD5 = 1 ;            // 將 RD5 輸出 High
```

另一範例：

```
#define   SW1      PORTAbits.RA0
#define   LED1     PORTDbits.RD0
TRISDbits.TRISD0 = 0;          // 設定 RD0 爲輸出腳功能
TRISAbits.TRISA0 = 1;          // 設定 RA0 爲輸入腳功能
:
If (!sw1) LED1=1;              // SW1 被按下，點亮 LED1
else LED1=0;                   // SW1 放開，LED1 熄滅
```



Lab Exercise 2

程式裡使用 pic16f887.h 的定義
周邊名稱及位元結構

準備 LAB2
關於 LCD 顯示函數

- **LCD Functions (mid_lcd.c)**
  - **void OpenLCD (void) ;**
  - **void putsLCD(char *) ;**
  - **void putrsLCD(const char *) ;**
  - **void putcLCD(unsigned char) ;**
  - **void puthexLCD(unsigned char) ;**
  - **void SetCursorLCD(unsigned char , unsigned char) ;** // 設定游標位置於 (X,Y)

LAB2 - Drive LCD Module.c
基本 I/O 的 使用

- 本練習請開啓
  - **..\Hi-Tech PICC v2.0 LAB\lab2\ LAB2 - use the Bit Struct.mcp**
- 請使用 **PIC16f887.h** 所提供的**FSR**暫存器定義名稱，修改 **I/O Port** 以符合 **Hi-Tech PICC** 的語法
  - **LAB2 - Drive LCD Module.c :** 主程式
  - **mid_lcd.c , mid_lcd.h : LCD** 顯示函數
- 按下 **SW2** 或 **SW3** 時， **LCD** 會顯示加一或減一 **LCD** 模組顯示如下的字元

Hi-Tech PICC Ex2
Up: 01 Down: FE

結構變數
共用變數
位元結構

結構型態 (Structures)

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

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

結構名稱

變數成員

結構變數

使用基本的結構變數

- 欲使用結構內的成員，可使用“.”來組成:

  variable-name.memberx *（結構變數.成員）*

```
struct Comm_protocol
{
   char ID[6];
   char Data[10];
   char  Message[20];
   unsigned int CRC;
   unsigned char Repeat;
} Rec_Fram;
        :
        :
unsigned char j;
for(j=0;j<20;j++)
{
writeUSART(Rec_Fram.Message[j]);
}
```

Comm_protocol 在 RAM 的排列

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

存取結構中的陣列 (一)

範例 1 ：

```
struct filtered_data {
                char  Fbandgap[4];
                char  Frefhi[4];
                char  Freflo[4];
                char  Ftemp[4];
        } Fcount;
```

成員的存取使用小數點的運算符號 “ . “

例如：

```
        Fcount.Fbandgap[1] = 0x34;
```

“***Fbandgap***” *陣列的第二個成元會被存入* **0x34** 的值.

存取結構中的陣列 (二)
使用結構指標

範例 2 :

格式 1 : 結構變數 . 結構元素
格式 2 : 結構指標 -> 結構元素

使用範例 1 所定義過的結構名稱 **“filtered_data”**

宣告: **struct filtered_data *ptr;**

ptr = &Fcount;

ptr->Frefhi[0] = 0x87;

結構陣列**“Fcount.*Freghi*”** *的第一個元素會被存入* **0x87** 的值.

在宣告同時指定結構變數的初始值

範例 3 ：

```
struct temp
{
        int count;
        float freq;
        char sign;
} pwm = {
                0x1234,
                1850.5,
                0xFF
        };
```

結構陣列

範例 4 ：

```
struct control {
        char mode;
        char state;
        char sign;
} drive[3];
```

存取 “***mode***” 的第一個陣列值:

```
drive[0].mode = 0x33;
```

存取 “***sign***” 的第三個陣列值:

```
drive[2].sign = 0xFF;
```

共用型態（union)

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

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

共用型態名稱

共用元素成員

共用型態變數

共用型態的資料架構

```
union EE_tag
{
    int  Word;
    char Byte[2];
} TC_74;
```

Word	
Byte[0]	Byte[1]
RAM0	RAM1

EE_tag 佔 2 個 Bytes

範例：

```
union
{
        int Word;
        char Bytes[2];
} TC_74;

for (i=0;i<5;i++)
   {
        EE_Addr=i;
        TC_74.Word = EERandomRead(0xA0,EE_Addr);
        if ( TC_74.Word >= 0 )
        Secu_Code[i] = TC_74.Bytes[0] ;
   }
```

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

**** EERandomRead( ) 的傳回值要 >= 0 才表示讀取中無錯誤 !!**

共用型態和成員的存取 (一)

範例 1 :

```
union u_tag {
        char abc;    // 8 bits
        int value;   // 16 bits
} utemp;
```

成員的存取使用 '.' 運算元

- *utemp.abc* = 'A';
- *utemp.value* = 0x3456;

範例 2 :

```
union u_tag  *uptr;
uptr = &utemp;
```

成員的存取使用 '->' 運算元

- *uptr->value* = 0x5678;

共用型態和成員的存取 (二)

- 一個結構型態或共用型態的宣告內，也可含有其它的共用型態或結構型態

範例 : 在整數(16-bit)中單獨存取其中一個 8-bit 值

```
union    {
         unsigned int var;
         struct    {
                   char var_lo;
                   char var_hi;
                   } hilo;
         } mix;

char a, b;
void main( void )
{
    mix.var = 0x1234;
    a = mix.hilo.var_lo;
    b = mix.hilo.var_hi;
}
```

合併使用 結構型態 & 共用型態

- 在浮點數中單獨取出指數部分

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

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

以 **Byte** 為單位的位元結構

範例 **1**：位元結構型態 **– Byte** 大小

```
struct  {
        unsigned int      : 3;  // bit padding
        unsigned int b3   : 1;  // bit 3
        unsigned int b4   : 1;  // bit 4
        unsigned int      : 3;  // bit padding
} PCLATHbit  @ 0x0A;
```

底下 **C** 的程序說明如何使用位元結構存取 **“b3”** 的位元成員，其結構變數直接宣告在 **0x0A** 的**RAM**位址：

```
PCLATHbit.b3 = 1;
```

說明：假如位元元素已指定大小，其結構變數採用絕對位址方式宣告 **(**結構變數 @ 絕對位址**)**， 編譯器將不會保留其特定位址。

以 Word 為單位的位元結構

範例 2：位元結構型態 – Word 大小 (16 Bits)

```
struct  status {
        unsigned int  high  : 1;   // LSb
        unsigned int  low   : 1;
        unsigned int        : 5;  // bit padding
        unsigned int  dir   : 1;
        unsigned int  rate  : 1;
        unsigned int        : 6;  // bit padding
        unsigned int  fault : 1;  // MSb
} pressure;
```

底下 C 的程序說明如何使用位元結構存取位元元素 **"*dir*" :**

```
pressure.dir = 1;
```

說明： 第一個元素名稱會是這個結構變數的最低位元，其中各元素所占的位元多寡均會依序安排位置，但不可超過16位元。

位元的使用範例(一)

範例 1:

```
union {
        unsigned char var;
        struct {
                unsigned int  bit0  : 1;
                unsigned int        : 6;  //padding
                unsigned int  bit7  : 1;
        } bits;
}  uvar;
```

現在你可以看到：

uvar.var → **8 bits** 的變數
uvar.bits.bit7 → 存取 **bit 7** 的位元

位元的使用範例(二)

範例 2:

宣告: **static bit Flow @ (unsigned) & PORTB*8 + 4;**

程序中 **PORTB** 的 **bit4** 可使用名稱爲 **"*Flow*"** 的助憶符號取代 **:**

例 **: Flow = 1;**

範例 3:

宣告: **static bit Motor @ (unsigned) & PORTC*8 + 3;**

程序中可以用 **"*Motor*"** 的助憶符號代替 **PORTC** 的 **bit3 :**

```
if (Motor)          {
                      // Statements when Motor == 1
                    }
else      {
            // Statements when Motor != 1
          }
```

位元的使用範例(三)

範例 **4:**

你也可以採用巨集的方式宣告:

```
#define PortBit(port,bit)  ((unsigned) & (port)*8 + (bit))
```

然後就可以使用下列的宣告方式指定任何暫存器的任一位元定址：

```
static bit led8     @ PortBit(PORTB,8);
static bit pulse    @ PortBit(PORTC,7);
```

一般較常用的方式是用 **#define** 來定義新名稱

```
#define S1_Button RA4;
#define S2_Button RB0;
```

位元的使用範例(四)

範例 **5:**

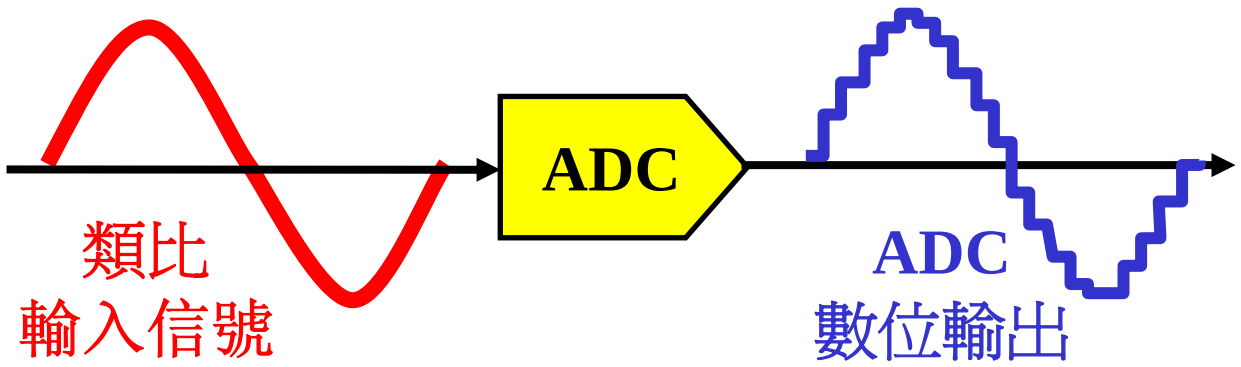
```
bit flag1;              // globally visible
bit flag2;              // globally visible

void main ( void )
{
   static bit flag3;    // locally visible
   flag3 = 1;

   . . .
   while(1);
}
```

類比轉換器（ADC）

Rev 2.0 August 2011 Modified by Richard yang

ADC 概述

- 類比轉換器模組
  - 將類比輸入信號轉換為 8 或 10 Bits 二進制值
  - 可選的內部或外部參考電壓
  - 轉換完成後可以產生中斷
  - 中斷可用於將PIC微控器從休眠模式喚醒



ADC 暫存器

- ADC 實現兩個控制暫存器
  - ADCON0和ADCON1

ADCON0

ADCS1	ADCS2	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON

位元	功能
ADCSx 位元	A/D 轉換時脈選擇位元 00 = Fosc/2，01 = Fosc/8，10 = Fosc/32， 11 = FRC （內部 RC 振盪器）
CHSx	類比通道選擇位元
GO/$\overline{\text{DONE}}$	1 = A/D 轉換正在進行 0 = A/D 轉換完成
ADON	啟動 ADC 模組

ADC暫存器

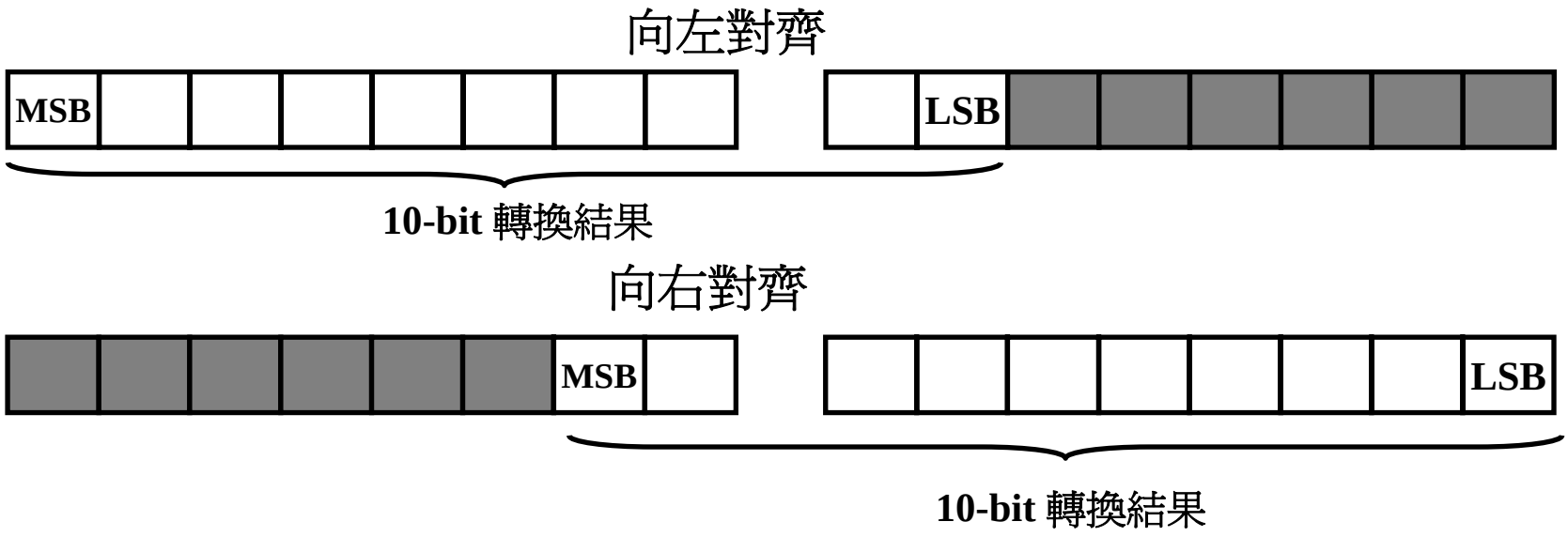
ADCON1

ADFM	---	VCFG1	VCFG0	---	---	---	---

位元	功能
ADFM	轉換結果暫存器對齊方式位 **1 = 向右對齊， 0 = 向左對齊**
VCFG1	負參考電壓 **1 = 來自Vref-接腳的外部電壓源， 0 = Vss**
VCFG0	正參考電壓 **1 = 來自Vref+接腳的外部電壓源， 0 = Vdd**

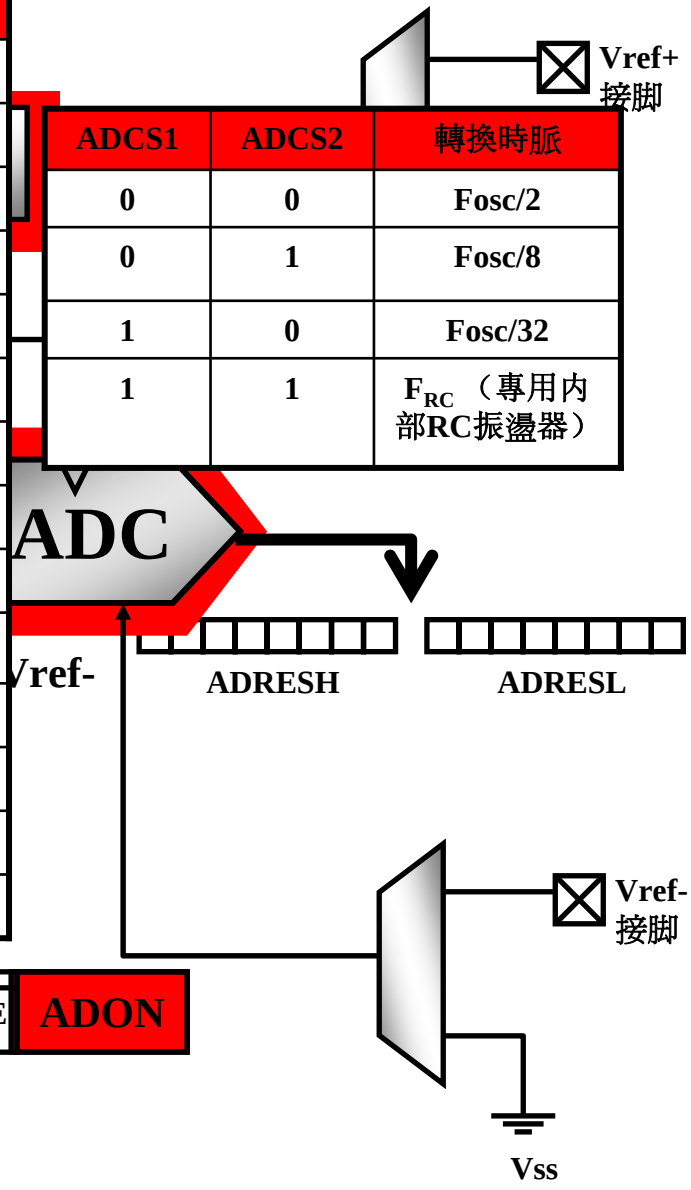
ADC 暫存器

- 轉換完成後，**ADC** 轉換結果被放到個結果暫存器 **ADRESH** 和 **ADRESL**中
- **10-bit ADC** 轉換結果可以向左對齊也可向右對齊



A/D模組方塊圖

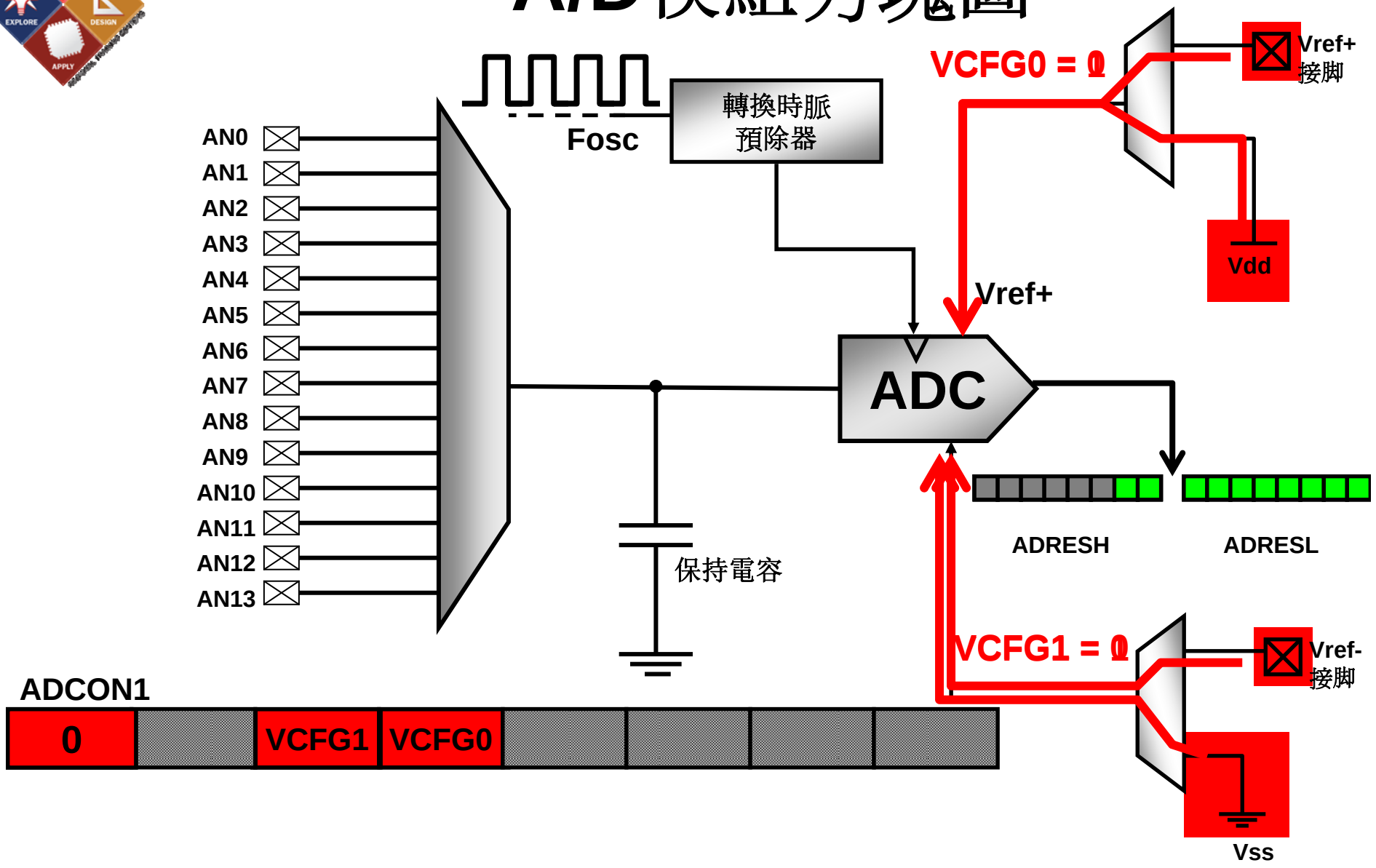
CHS3	CHS2	CHS1	CHS0	通道
0	0	0	0	AN0
0	0	0	1	AN1
0	0	1	0	AN2
0	0	1	1	AN3
0	1	0	0	AN4
0	1	0	1	AN5
0	1	1	0	AN6
0	1	1	1	AN7
1	0	0	0	AN8
1	0	0	1	AN9
1	0	1	0	AN10
1	0	1	1	AN11
1	1	0	0	AN12
1	1	0	1	AN13



ADCON0

ADCS1	ADCS2	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON

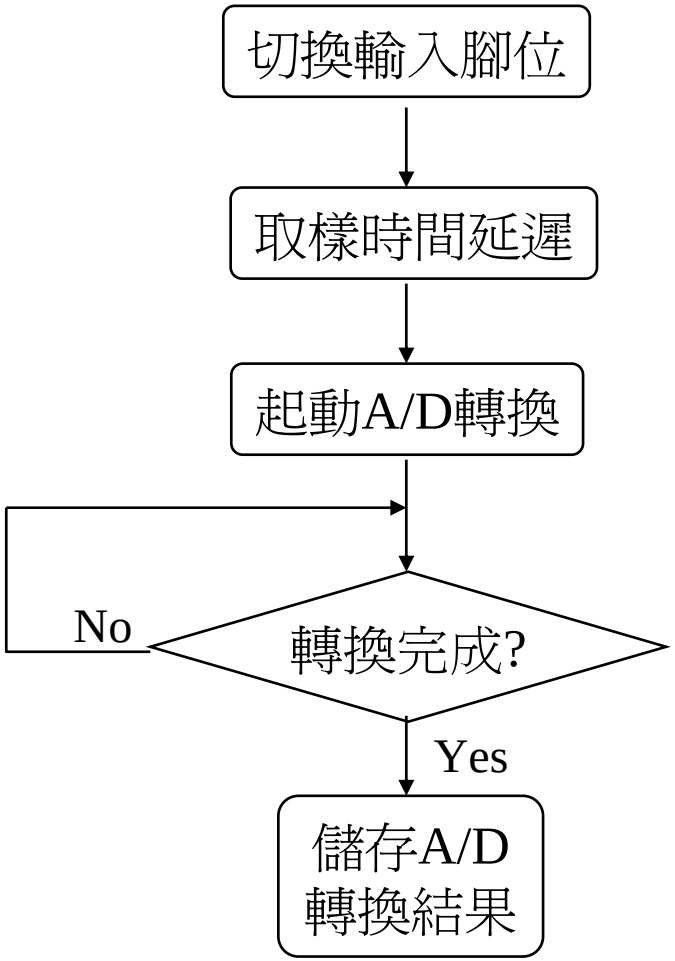
A/D模組方塊圖

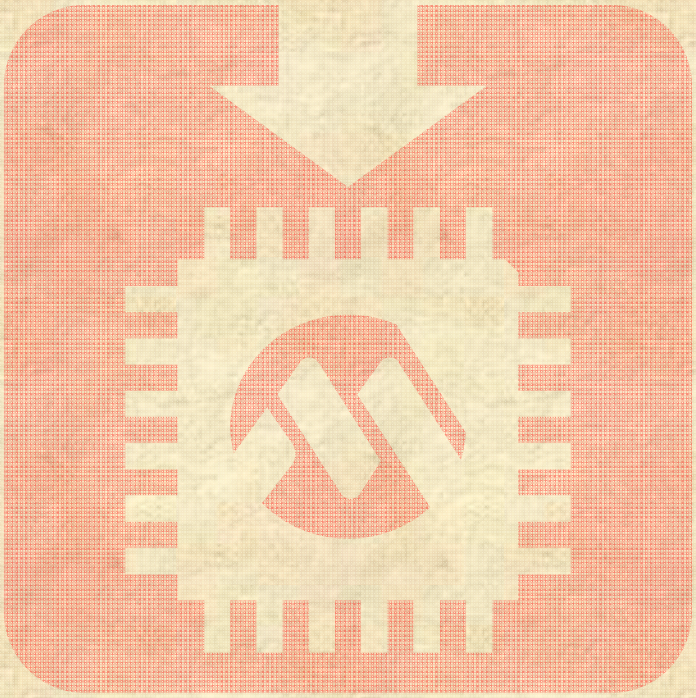


ADC 時序注意事項

- 選擇了一個類比轉換通道時，必須花費一定的時間使保持電容充電 (**11.5uS Typ.**)
- 用户必須根據系統的頻率選擇適當的 **ADC** 的工作時脈 (T_{AD})，且T_{AD} 的設定不可以少於 **Data Sheet** 裡的規定 **(PIC16F887** 爲 **1.6uS)**
- 所有 **10-bit** 轉換的完成需要 **11** 個 T_{AD}
  - 最快轉換時經爲 **17.6uS**
  - 連同 **Channel** 切換 **+ AD** 轉換共需 **29.1uS**
  - 使用內部 **A/D Internal RC** 做爲 T_{AD} 時需 **6.0uS (Max.) @** V_{DD} **= 5V**

A/D 轉換基本流程





Lab Exercise 3

了解 union & struct 的資料結構

10-bit AD 轉換器

LAB3 - Bit Struct.c

重要函數說名

- **void Init_IO(void)**
  - 設定 **I/O** 的輸出入
- **Void Init_PWM2(void)**
  - 設定 **Timer2** 及 **CCP2** 爲 **PWM** 輸出
- **void Init_Adc(void)**
  - 設定 **A/D** 工作模式
- **void A2D(unsigned char)**
  - 傳入欲轉換的 **Channel**，並啓動 **A/D** 轉換動作
- **void LCD_ItoA(unsigned int)**
  - 將整數轉換成**10**進制的**ASCII**後並顯示在 **LCD**
- **unsigned char Set_BCD_ASCII(unsigned char)**
  - 居先零的抑制

LAB3

- 專案開啓：
  - **..\Hi-Tech PICC v2.0 LAB\lab3\LAB3 - Bit Struct.mcp**
- **LAB3** 是利用 **10-bit A/D** 轉換結果利用 **union** 和 **struct** 來拆解資料
  - **10-bit** 的數值會轉換成 **10** 進制並做居先零的抑制後顯示在 **LCD Module** 的第一行
  - 另將 **A/D** 拆成兩個 **8-bit** 的資料，分別顯示在**LCD Module** 的第二行 **(2-bit MSB & 8-bit LSB)**
- **VR1** 做爲 **ADC** 轉換電壓的輸入，其電壓直可以在 **Proteus** 的電壓表看到方便查詢 **AD** 轉換的結果。
- **LAB3** 也使用到 **CCP2** 模組做爲 **PWM** 的輸出，輸出的 **Duty Cycle** 可以在 **Proteus** 的示波器上觀察其輸出變化或觀察在 **APP001** 上的 **D9-CCP2 LED** 的亮度變化 。

RAM & ROM 形態的指標和陣列

Rev 2.0 August 2011 Modified by Richard yang

使用指標

- 什麼是指標？
  - 使用指標時，有兩個重要觀念
    - 用來存放位址變數稱之爲指標變數
    - 指標變數的內容是位址

<Object's type and qualifiers>	*	<pointer's qualifiers>	<pointer's name> ;
char	*		cp ;
const char	*		cp ;
const char	*	const	cp ;

常見錯誤指標用法 (一)

- 將位址指定給不合型態的指標將帶來無法預估的災難

範例 1： 錯誤的 ROM 指標例

```
const char array[ ] = {0, 2, 4, 6};
char * cp, c;
cp =  array;
c = * cp;              // 指標 (*cp) 是對 RAM 作業而不是對 ROM
```

```
const char *cp;
char c;
正確宣告是將指標宣告為指到 ROM 的指標
```

常見錯誤指標用法 (二)

範例 2： 錯誤的 bank 宣告例

```
void process ( int * ip );  // 函數雛型宣告 (錯誤的宣告)

bank2 int value;            // 變數 value 是放在 Bank2
process ( &value );         // &value 處理時指標卻是指到 Bank0
```

```
void process( bank2 int * ip )
正確宣告是需加入相對的 Bank2
```

RAM 的指標
Midrange PIC16

- **RAM 的指標範圍爲 8-bits**
  - 可以存取兩個 **RAM banks (bank0+bank1 or bank2+bank3)**
    - 內定及使用 **bank1保留字**
      - 直接存取 **bank 0** 和 **bank 1**
    - **bank2 保留字可以存取 banks 2 and 3**
    - 由 **IRP 位元 (STATUS)** 決定

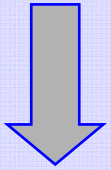
RAM 的指標
Midrange PIC16F

- **RAM 指標範例**
  - **char *ramptr;**
    - 在 **bank0** 宣告一個 **8-bit** 指標
  - **bank2 char * ramptr;**
    - 宣告一個 **8-bit** 指標，指向位址爲 **bank2**
  - **char * bank2 ramptr;**
    - 在 **bank2** 宣告一個 **8-bit** 指標，指向位址爲 **bank1**
  - **bank2 char * bank1 ramptr;**
    - 此種宣告有何意義**?**

RAM 的指標
Midrange PIC16F

- **Use Bank2 or Bank3 Pointers**
  - **bank2 char *ramptr;**
  - **bank3 char *ramptr;**

- **bank3 char * bank2 ramptr;**
  - 這種宣告會有何結果?



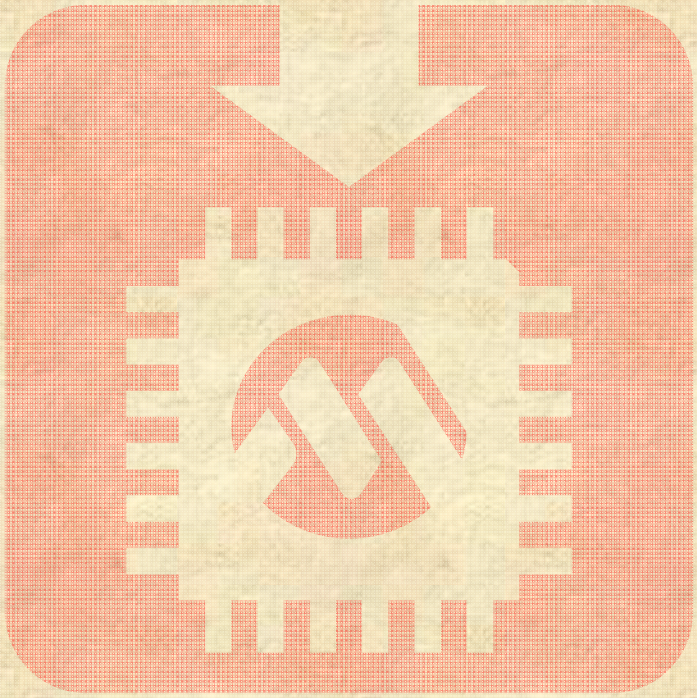
ramptr 是存放在 **BANK2** 的指標變數，其所指到的位址爲 **BANK3** 的某個變數

RAM 用指標方式存取

```
#include <pic.h>
char RAMarray1[ ]= “Hi-Tech PICC”;

void write_LCD ( char data )
{
     PORTD = data;
}

void Update_LCD ( void )
{
  char *ramptr;                          //define auto type pointer
  ramptr = RAMarray1;                   // init. pointer
  while( *ramptr )   {
                          write_LCD( *ramptr );
                          ramptr++;
                        }
 }
void main( void )
{
    Update_LCD();
    while(1);
}
```

RAM 用陣列方式存取

```
#include <pic.h>
char RAMarray[ ] = “Welcome to Master’s 2000”;

void write_LCD( char data )
{
        PORTD = data;
}

void Update_LCD( void )
{
  char ch;                      // auto variable
  char i = 0;                   // auto variable
  while (ch = RAMarray[i])      {
                                    write_LCD(ch);
                                    i++;
                                }
}

void main( void )
{
   Update_LCD();
   while(1);
}
```

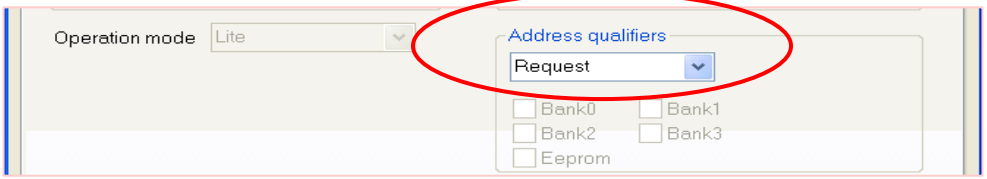


Lab Exercise 4

變數及指標的特定位址的設定

LAB4 – RAM Pointer.c

- 專案開啓：
  - **..\Hi-Tech PICC v2.0 LAB\lab4\LAB4 – RAM Pointer.mcp**
- 注意：**Compiler** 的 **Address Qualifier** 預設值爲 **Ignore**，需此功能要設爲 **Request**



- 程式執行使用**MPLAB SIM** ，先清除所有的 **RAM**
  - Debugger → Clear Memory → File Register
- 執行程式後，檢查指標變數存放的內容爲
  - `aptr = IRP=0，0x20 , 內容值爲 0x61 ('a')`
  - `bptr = IRP=0，0xA0 , 內容值爲 0x62 ('b')`
  - `cptr = IRP=1，0x10 , 內容值爲 0x63 ('c')`
  - `dptr = IRP=1，0x90 , 內容值爲 0x64 ('d')`

陣列的大小
Midrange PIC16F

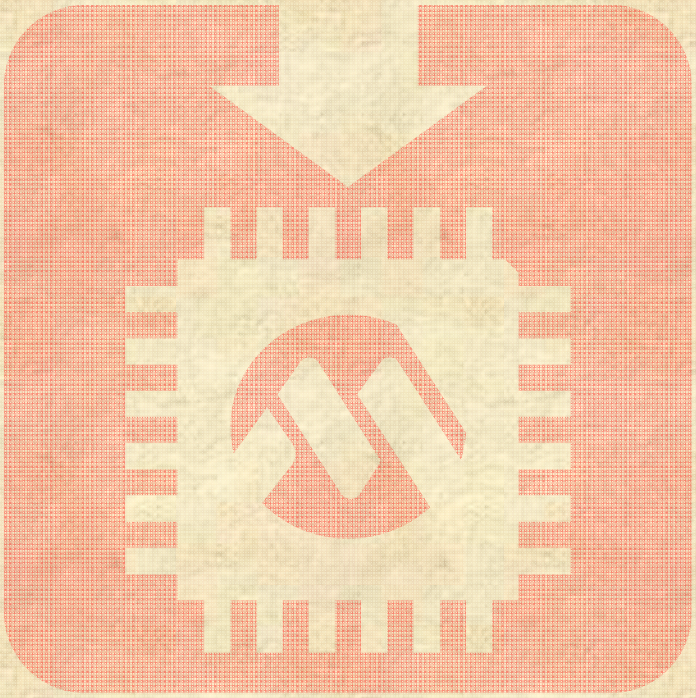
- **PIC16F887 RAM 的架構爲不連續 (Non Linear)**，所以在陣列宣告時有大小的限制
  - **BANK2 & BANK3 最大可以有 96 Bytes 的容量**
    - **bank2 char RAMarray2[96]; // 最大值，超過就會有錯！**
- 所以超過此限制就只能將陣列拆開使用
- 如果要使用較大陣列的話
  - 使用 **Enhanced PIC16F1xxxx** 的元件 **(線性定址)**
  - 使用 **PIC18Fxxxx** 的元件 **(合併Bank方式)**

ROM 指標的查表法

```
#include <pic.h>
const char ROMarray1[ ]="C Compiler for PICmicro";
const char *romptr;     // pointer defined

void write_LCD( char data )
 {
  PORTD = data;
}

void Update_LCD( void )
{
  romptr = ROMarray1;
  while(*romptr)     {          // test for null char
                                write_LCD( *romptr );
                                romptr++;
                     }
}

void main( void )
{
  Update_LCD();
  while(1);
}
```

ROM 陣列的查表法

```
#include <pic.h>
const char ROMarray[ ] = “Welcome to Master’s 2000”;

void write_LCD( char data )
{
  PORTD = data;
}

void Update_LCD( void )
{
   char ch;                                  // auto variable
   char i = 0;                               // auto variable
   while(ch = ROMarray[i])    {              // test for null char
                                             write_LCD(ch);
                                             i++;
                              }
}

void main( void )
{
  Update_LCD();
  while(1);
}
```



Lab Exercise 5

ROM & RAM 陣列的存取
及特定位址的設定

LAB5 – Access Array.c

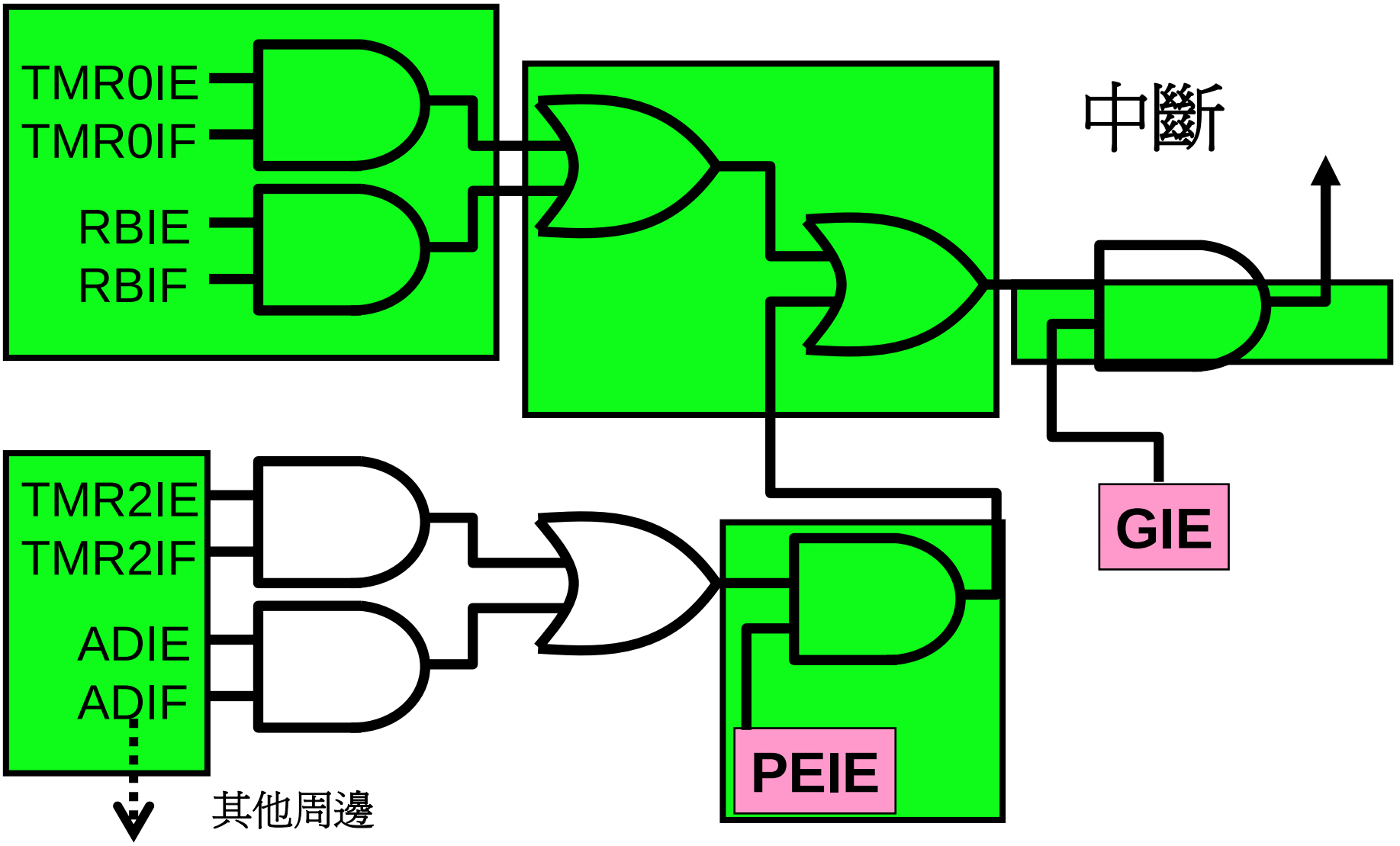
取得 **ROM** 陣列資料填入 **RAM** 陣列

- 專案開啓：
  - **..\Hi-Tech PICC v2.0 LAB\lab5 \LAB5 - Assign Array.mcp**
- 注意：**Compiler** 的 **Address Qualifier** 預設值爲 **Ignore**，需此功能要設爲 **Request**
- 程式執行使用**MPLAB SIM** ，先清除所有的 **RAM**
  - Debugger → Clear Memory → File Register
- 執行程式後，檢查兩個 **RAM** 陣列的位址及內容值是否正確
  - **RAMarray1** 位址 **@0x20**
  - **RAMarray2** 位址 **@0x110**


HANDS-ON
Training
MICROCHIP TECHNOLOGY
EXPLORE
DESIGN
APPLY
REGIONAL TRAINING CENTERS

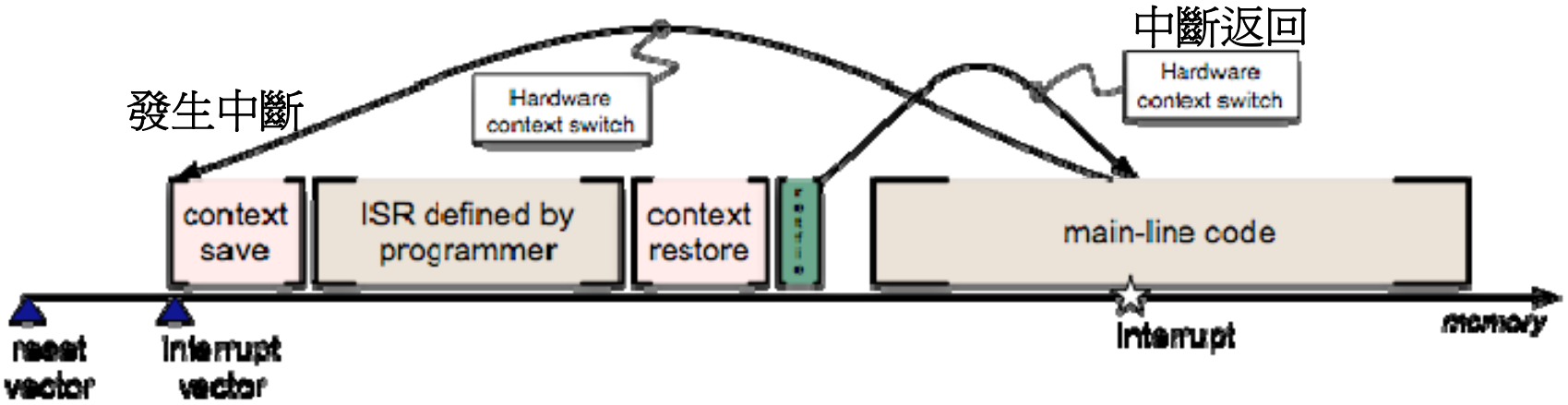
中斷管理

中斷邏輯



中斷

- **PIC16F** 系列只有一個中斷進入點
  - 進入位址 **0x0004**
  - 中斷服務函式 **(ISR)** 會自動安排此中斷位址的跳躍
- 如果超過一個週邊中斷來源
  - 須透過判斷各個中斷旗號來決定來源



撰寫中斷函式

- **ISR 可以完全用 C 來完成**
  - 也可以使用組合語言方式
  - 使用 **“interrupt”** 的定義字
  - 中斷函式不支援參數 / 引數的傳入，也不支援參數的回傳
  - 中斷函式所使用的變數宣告需加上 **“voltale”**，最好也使用 **“near”** 來擺放在 **Common Bank (0x70 ~ 0x7F)**
  - 中斷函式不可以當作一般的函式來呼叫
  - 會自動安排中斷進入位址
  - 使用 **“retfie”** 的指令返回

中斷函式範例

- C 函數可以被連結到中斷向量的位置，只要在函數原型宣告使用保留字 “***interrupt***” 即可

- 建議檢查中斷函式
  - 該週邊的中斷旗號與周邊的中斷致能位元

```
void interrupt isr(void)
{
  if(RCIF && RCIE)
    byte = RCREG;
}
```

中斷函數名稱

中斷背景暫存
Context Switch

	Mid-Range PIC16Fxxx	Enhanced Mid-Range PIC16F1xxx
GIE disable	= 0 (中斷抑制)	= 0 (中斷抑制)
Hardware save	PC	PC, PCLATH, WREG, STATUS, FSRs, BSR
Software save	PCLATH, WREG, STATUS, FSR	
User's ISR	✔	✔
Software restore	PCLATH, WREG, STATUS, FSR	
Hardware restore	PC	PC, PCLATH, WREG, STATUS, FSRs, BSR
Return	`retfie`	`retfie`
GIE enable	= 1 (中斷致能)	= 1 (中斷致能)

中斷函式勿使用一般函式

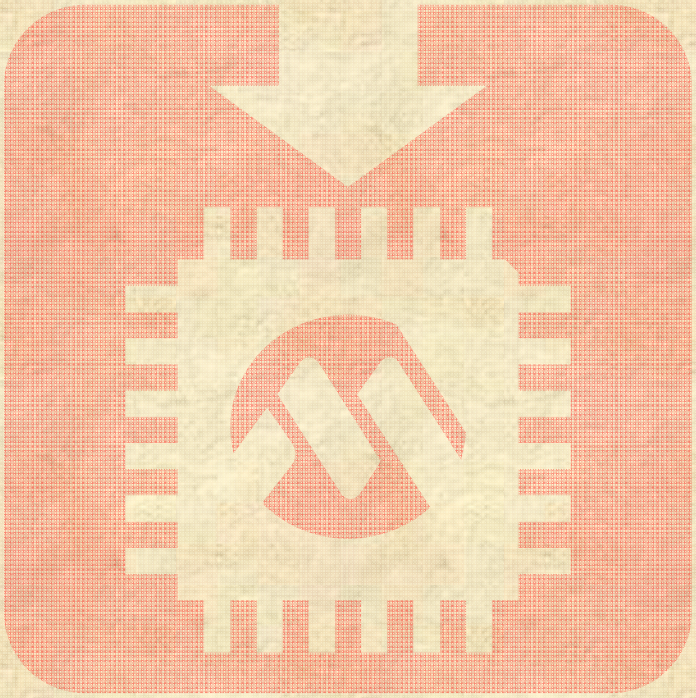
- 在中斷函式內部可呼叫一般的函式來使用
  - 有可能該函式正使用時被中斷打斷，若中斷裡有使用此函式將造成變數及暫存資料的損壞

```
void
main(void) {
  while( ! ready)
    wait(D_10uS);
    //要是此時發生中斷
  // ...
```

```
void interrupt
isr(void) {
  while( ! ready)
    wait(D_10uS));
 //被破壞了，返回後將發生錯誤
  // ...
```

多個中斷處理

```
Static void interrupt isr (void)
{
        if (T0IF && T0IE) {                 // Tomer0 interrupt
                        TMR0 =250 ;         // Reload the Timer value
                        T0IF = 0 ;          // Clear Timer0 INT flag
        }

        if (INTF && INTE) {
                        Relay = 1 ;         // Turn the relay on
                        INTF = 0 ;          // Clear the interrupt
        }
}
```



Lab Exercise 6

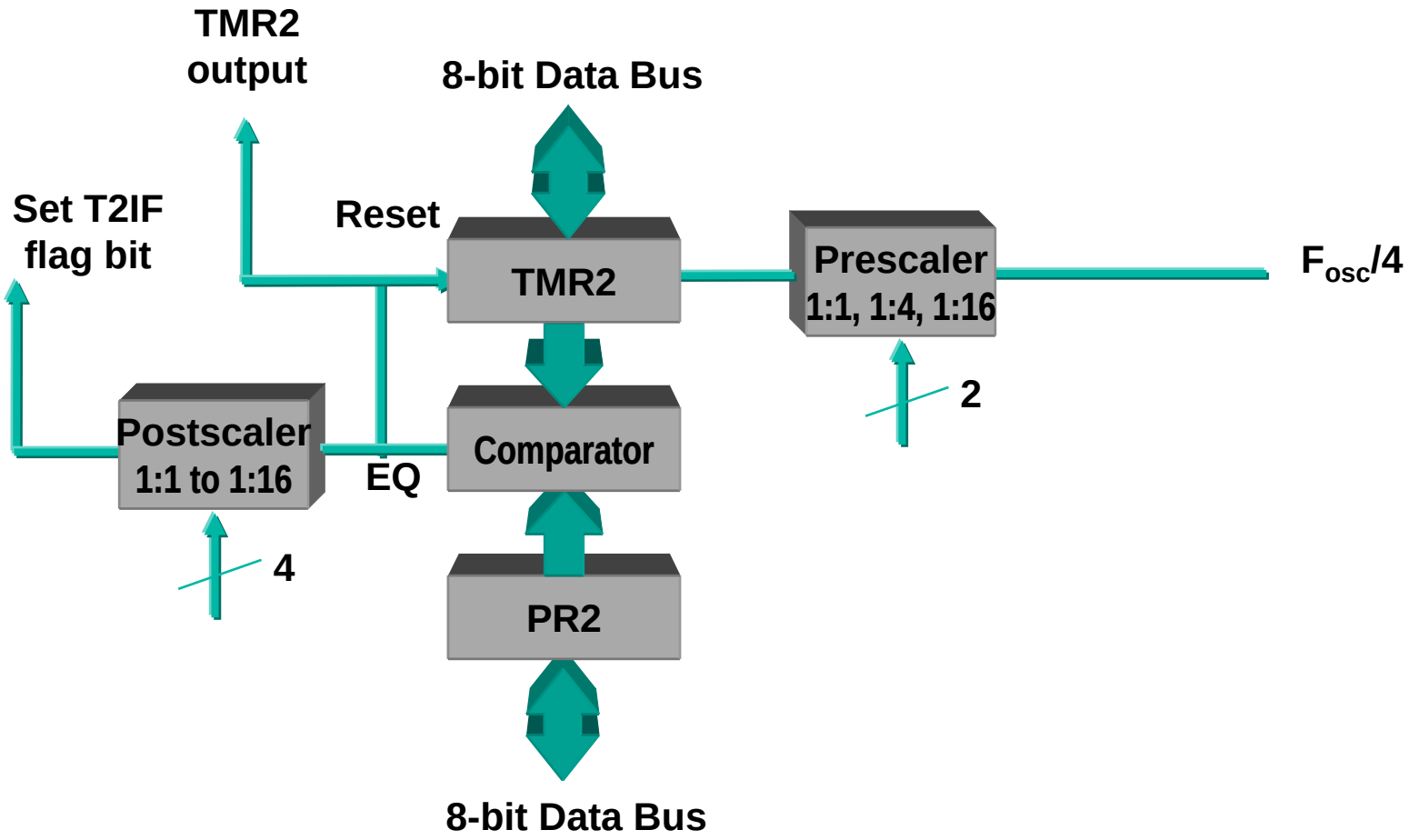
使用 Timer 2 的中斷

做一個左、右移動的霹靂燈

準備練習六
處理 Timer2 中斷

- **Timer2** 說明
  - **8-bit** 比較式計時器，配有前、後級除頻器
  - **PWM** 的基本計時計數器 **(Duty & Period)**
  - **TMR2** 是可讀、寫的計時暫存器
  - **TMR2** 每次增加一，直到與 **PR2** 暫存器值相等後，重新歸零
  - **TMR2** 與 **PR2** 暫存器值相等會輸出訊號到 **postscaler** 到達設定次數時會產生中斷
  - **SSP (SPI™)** 同步傳輸速率產生器

準備練習六
Timer2 方塊說名



準備練習六
Timer2 暫存器說明

TABLE 7-1: REGISTERS ASSOCIATED WITH TIMER2 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
92h	PR2	Timer2 Period Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer2 module.

Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

- 中斷設定：**GIE , PEIE , TMR2IE**
- 中斷旗號：**TMR2IF**
- 中斷時間：**(8Mhz / 4) [16 x 16 x (77+1)] = 98.9Hz (10mS)**

LAB6 - Timer2 Interrupt.c

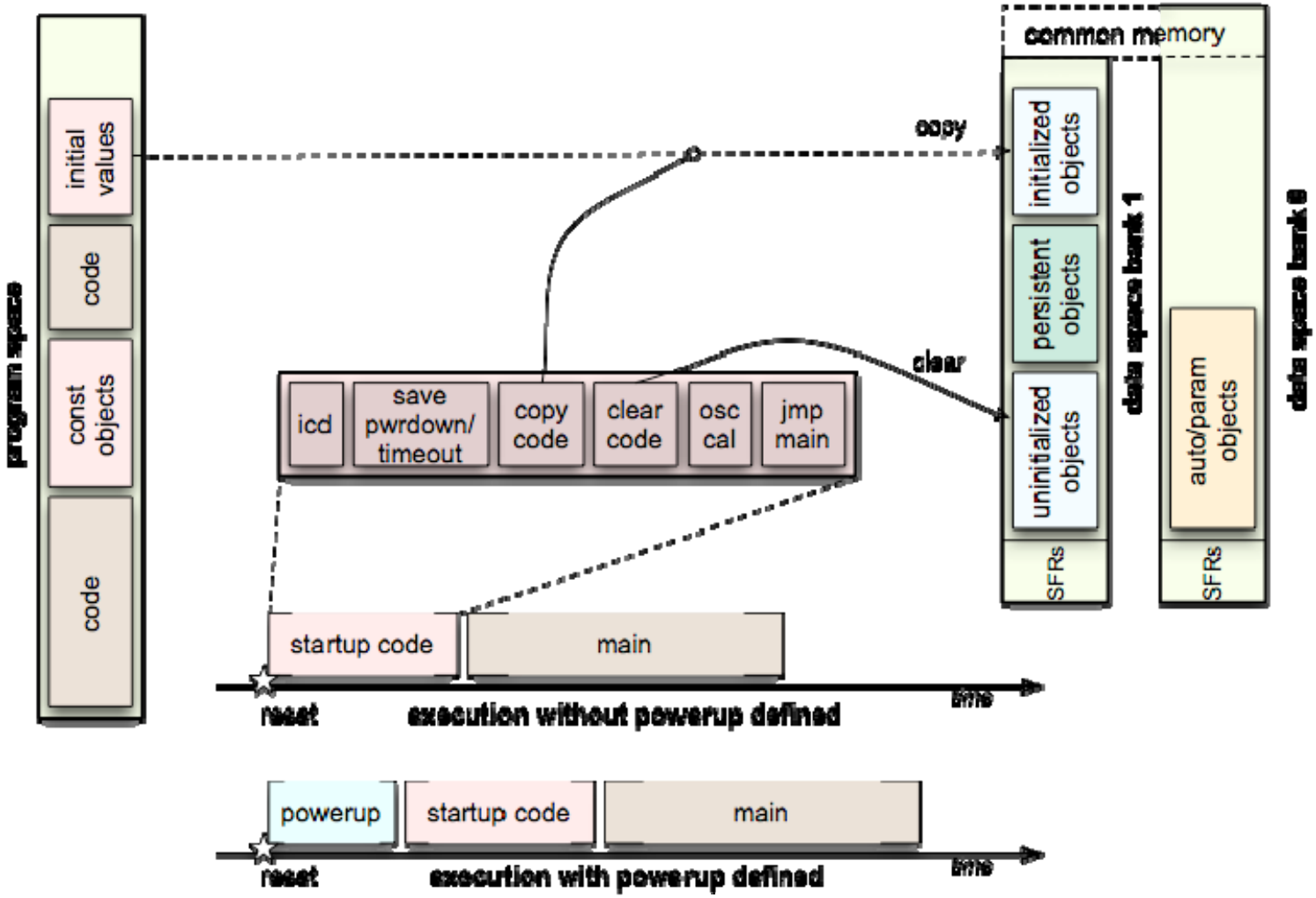
設定週邊中斷函數

- 專案開啓：
  - **..\Hi-Tech PICC v2.0 LAB\lab6\LAB6 – Timer2 Interrupt.mcp**
- 程式加入 **Timer2** 中斷處理功能，讓程式能夠每 **10mS** 中斷一次，每 **100mS** 執行跑馬燈的動作
- 使用 **Proteus** 觀察 **PORTD** 的 **LED** 變化

Hi-Tech PICC
重置後的初始設定

Rev 2.0 August 2011 Modified by Richard yang

開機後啓動的動作



啓動模組

- 重置位址在 **0x00**，函數名稱爲 ***startup***
  - **MCU Reset** 後立即執行此程式
  - 編譯成功後可以在 **Project** 下找到 **startup.as**
  - 可藉修改此程式，強制先執行 **I/O** 設定動作
  - **startup( )** 執行完後，控制權將交給啓動模組 **start( )**

- 啓動模組工作，函數名稱爲 ***start***
  - 設定 **data psects** 的初始資料
  - 清除 **bss psects (uninitialised data)** 區域
  - 載入**RC**振盪器的校正值(有內建**RC**振盪器者)
  - 將程式控制權轉移至 **main( )**

自行設定的啓動工作

- 如需一些在電源重置後需立即動作的事件可透過修改 **startup.as** 後重新編譯再連結即可達成自行設定的起始功能
  - 開機後，需要緊急設定 **I/O** 起始狀態
  - 使用著需修改 **startup.as** 的組合語言程式模組讓 **C** 程式可以執行自定的啓動工作
  - 採用先跳到 **mystartup( )** 先執行 **I/O** 設定後再跳到 **start( )** 執行啓動模組功能
  - 需考慮到 **PIC** 的架構

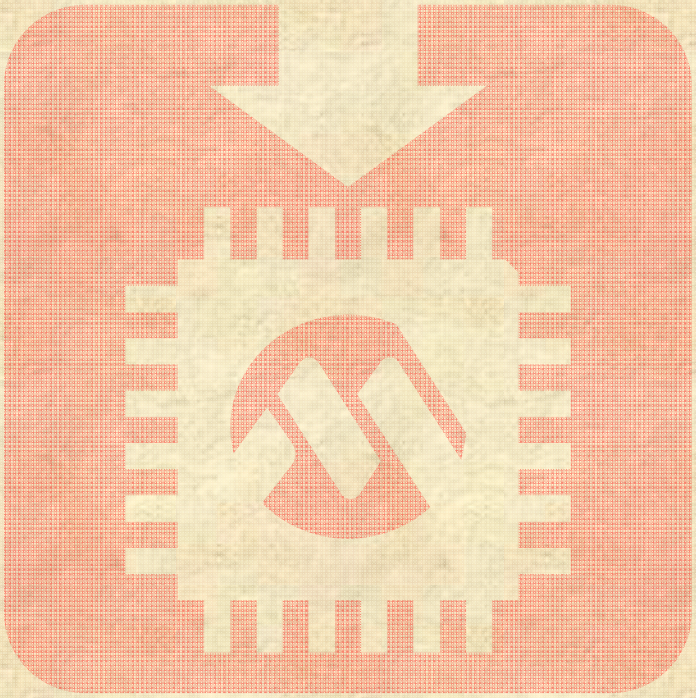
I/O Initialization on Power-Up
- PIC16FXXX

startup.as

```
extern  _mystartup

        psect   reset_vec
reset_vec:
        ; No powerup routine global start

; jump to User's mystartup
        goto    (mystratup & 0x7FF)

        psect   init
start
_exit
        psect   end_init
        global start_initialization
        ljmp start_initialization
;jump to C runtime clear & initialization
```

mystartup( )

```
. . .
void mystartup( void )
{
    PORTC = 0x00;
    TRISC = 0x00;
 #asm
    clrf   _STATUS
    clrf   _PORTB
    bsf    _STATUS,5
    clrf   _TRISB

    movlw   high start
    movwf   _PCLATH
    goto    ( start & 0x7FF )
#endasm
}
. . .
```



Lab Exercise 7

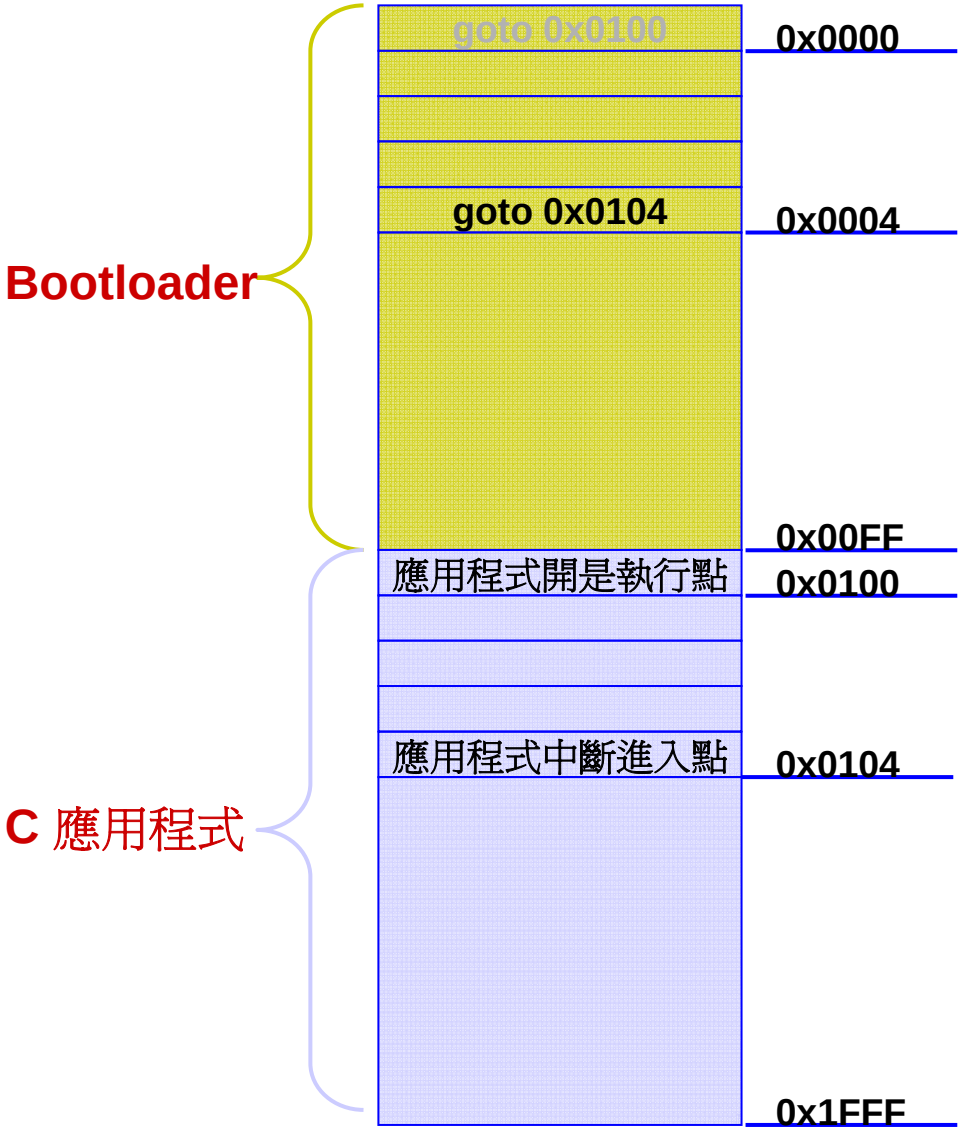
架構一個可以被 Bootloader 載入的
C 應用程式

Bootloader vs. 應用程式

- 以本實驗 **Lab 7** 爲例
  - **Bootloader** 要放在 **Program** 位址 **0x00 ~ 0xFF**
  - **C** 的應用程式要位移到 **0x100** 的位址
- 本實驗使用 **Lab 6** 的練習，讓使用者了解如何做程式位址的移動及原始中斷向量的規劃
- 實驗中將使用 **MPLAB SIM** 用來檢視程式的位址
  - 填入 **goto 0x104** 的指令碼到 **0x0004** 的中斷向量位址
  - 填入 **goto 0x0100** 的指令碼到 **0x0000 Reset** 位址，
  - 程式不使用 **Debug Mode** 直接使用 **Progeammer** 方式燒成 **Stand-Alone Mode** 運作。

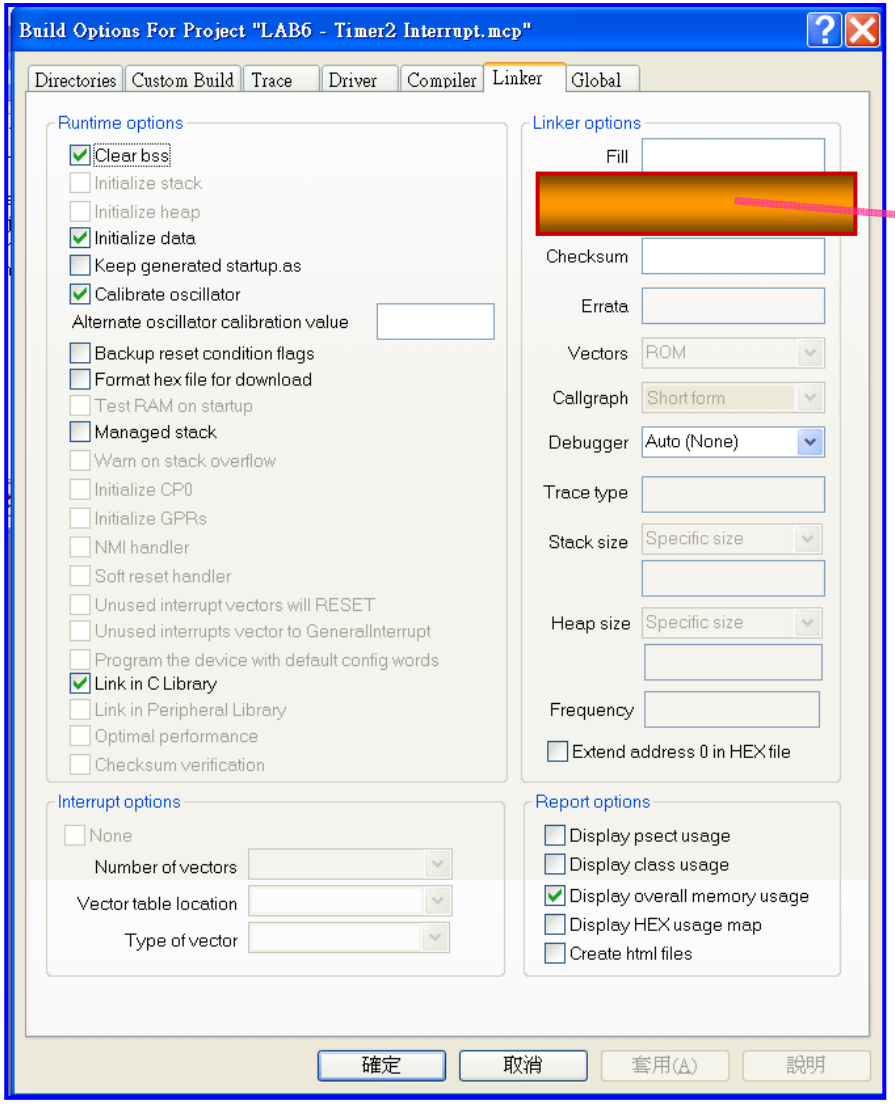
程式碼的分配

- **Bootloader** 爲一獨立程式，在 **PIC16F** 系列可以用 **UART**、**I2C**、**SPI** 等方式載入應用程式
- 應用程式是透過 **Bootloader** 載入的，載入完成後將控制權交給 **0x0100** 的應用程式進入點 **(**此點爲 **C** 應用程式的啓動模組**)**
- 中斷發生時，程式跳到 **0x0004** 在執行中斷轉移到 **0x0104** 的位址

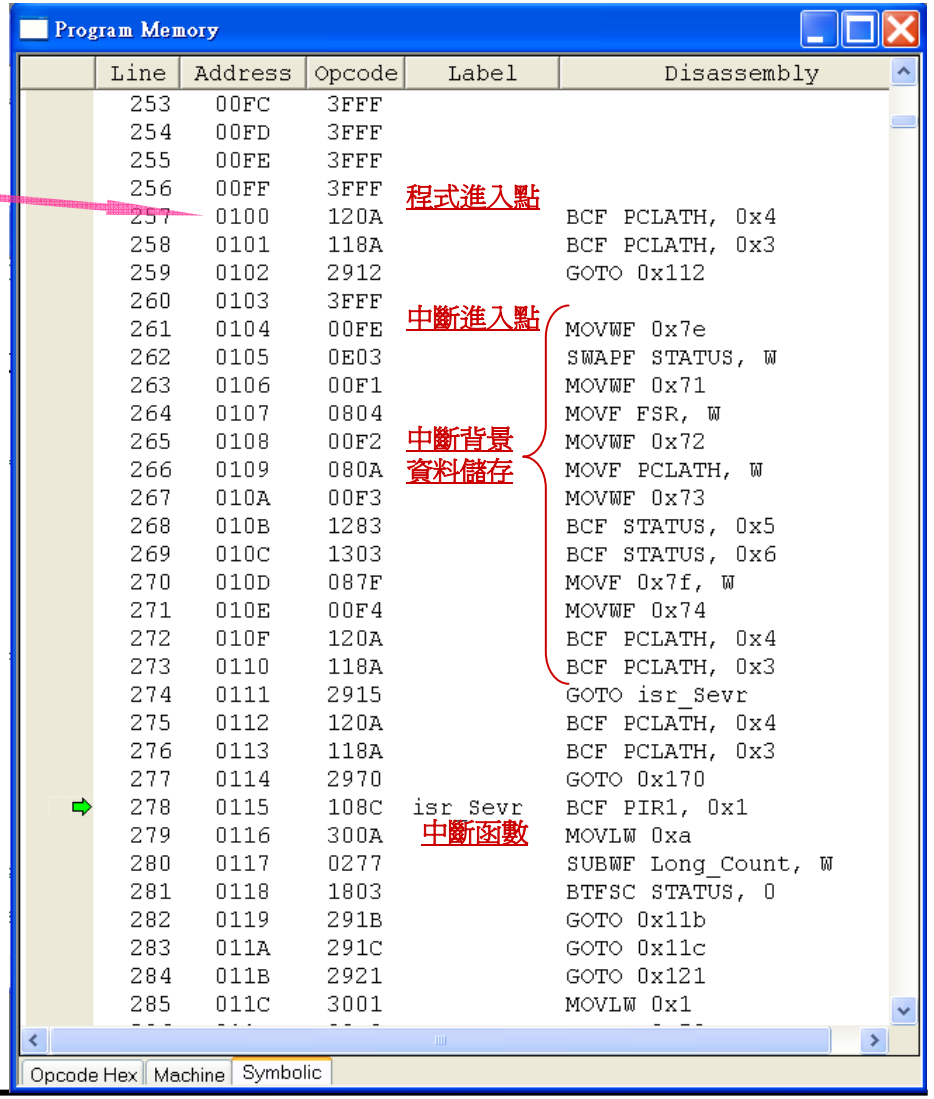


如何爲應用程式作位移

設定程式位移量



經編譯後的程式碼



LAB7 執行應用程式

- 專案位址
  - **..\Hi-Tech PICC v2.0 LAB\lab7\LAB7 - Bootloader Application.mcp**
- 將程式的位移設爲 **0x100** 編譯後(以下動作不可再編譯，因爲會刪除 **Bootloader**區域的程式碼)
  - 使用 **Proteus** 模擬，發現程式無法正確執行？
    - 因爲原始 **Reset** 及 **Interrupt** 向量沒有轉移出來
  - 將位址 **0x0004** 填入**OP code 0x2904 (goto 0x0104)** 將原始的中斷向量轉移至應用程式的中斷後再執行 **Proteus**？
    - 因爲程式還是無法將執行控制權完整的交給應用程式，因爲原本是 **Bootloader** 是會將控制權交給應用程式，但現在因爲沒有 **Bootloader** 所以控制權無法轉移
    - 所以將位址 **0x0000** 填入**OP code 0x2900 (goto 0x0100)**
- 應用程式可以正確執行了，使用 **Programmer Mode** 燒錄 **APP001** 驗證也是可以在 **Stand-Alone** 模式下執行

Power-Up 是否可以?

- 既然可以掌控 C 應用程式的一切執行動作，是否也可以將 “緊急要先做的開機動作” 先執行呢?
  - 可否將先前的 **mystartup( )** 放在 **Bootloader** 的區域，開機後先執行 **mystartup** 的開機初始 **I/O** 設定後，再將控制權交給 **C** 的應用程式裡的啓動模組 **?**

Bootloader 程式那裡有？

- **AN1310: High-Speed Bootloader for PIC16 and PIC18 Devices**
  - http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en546974
- **AN851: A FLASH Bootloader for PIC16 and PIC18 Devices**
  - http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en012031
- **Third-Party 網站**
  - http://www.microchipc.com/sourcecode/#bootloader

合併
組合語言與 C 語言

Rev 2.0 August 2011 Modified by Richard yang

合併組合語言與 C 語言

- 爲何要合併？
  - 增加程式執行得速度
  - 減少程式碼的空間
  - 減少 **RAM** 使用的空間
  - 舊有的資料庫或函數
  - 好玩，創新，故意讓別人無法改程式

合併組合語言與 C 語言

- 組合語言使用特殊助憶文字以擴展程式的視野
  - **ljmp –** 自動切換程式頁 **(Page)** 的跳躍指令 **(goto)**
  - **fcall –** 自動切換程式頁的副程式呼叫指令 **(call)**
- 組合語言指令使用 **“,w”** 或 **“,f ”** **(**內定的設定**)** 來指定運算後的儲存目地
- 組合語言不做最佳化處理

嵌入式的組合語言

- 使用 **#asm , #endasm** 的宣告
- 使用 **asm("……..");** 的語法

```
unsigned int var;
void main(void)
{
        var = 1;
#asm                            // like this...
        BCF 0,3
        BANKSEL         (_var)
        RLF     (_var)&07fh
        RLF     (_var+1)&07fh
#endasm
                                // do it again the other way...
        asm("BCF 0,3" );
        asm("BANKSEL _fvar");
        asm("RLF (_var)&07fh" );
        asm("RLF (_var+1)&07fh" );
}
```

獨立式組合語言

- 嵌入式的組合語言的程式碼會被檢查及作最佳化的縮減
  - 獨立式的組合語言將不會被做最佳化的處理
- 獨立式的組合語言使用的附加檔名爲 **“.as”**

獨立式組合語言

- 組合語言內的特殊助憶符號表示
  - **C** 的內涵物件**(Objects)** 和函數在轉換爲組合語言形式時，會在組合語言物件助憶名稱前自動加入“**_**”符號
  - 有關函數所使用的參數則會在物件助憶名稱前自動加入 “ **?_function name**”
  - 函數所使用的區域變數 **(auto variables)** 則會在物件助憶名稱前自動加入 “**?a_function_name**”

獨立式組合語言

- 組合語言與 C 語言可以同時使用函數或物件

在組合語言使用 C 的物件

```
int         response;              // 在 C 程式所宣告的公用變數
```

```
global    _response              ; directive used to give reference to external object
movwf  (_response & 07Fh)  ; accessing C object in assembly code
```

在 C 程式下使用組合語言的物件

```
global _answer                        ; directive to give object global scope
_answer   ds 2                        ; defining 2 byte variable
```

```
extern    unsigned int answer;    // 對外部的變數做宣告
answer = 10;                      // 用 C 來存取組合語言所需告的公用變數
```

獨立式組合語言

- 存取變數
  - 組合語言變數
    - 用 **DS** 虛指令宣告變數
    - 必須使用 **PSECT** 的定義
    - 標記或變數助憶符號需宣告爲 **GLOBAL** ，以允許被其它程式使用呼叫

獨立式組合語言

- 存取變數
  - **C** 語言變數
    - 在 **C** 程式哩，它必須宣告爲公用變數
    - 不要使用 **static** 方式宣告
    - 在組合語言使用 **C** 的變數時，需用 **(_)** 放在變數名稱前
  - **Multiple Byte** 資料型態
    - 低位元組**(LSB)**資料方在較低的位址
    - 高位元組**(MSB)**的存取可以使用位移增加量的方式存取該變數的每一**byte**

獨立式組合語言

- 變數的存取

Assembly:

```
#include <aspic.h>

global _Foo
signat _Foo,88
global _Var1, _Var2

psect text,global,class=CODE,delta=2

_Foo
      movf     _Var1,W
      addwf    _Var2,F
      banksel  (PORTB)
      movwf    BANKMASK (PORTB)
      return
      end
```

C:

```
#include <pic.h>

extern void Foo( void );

voltale near char Var1;
voltale near char Var2;

void main( void )
{
      Var1 = 0xAA;
      Var2 = 0x55;
      Foo( );
}
```

Hi-Tech PICC
所提供的資料庫

Rev 2.0 August 2011 Modified by Richard yang

函數庫

- 編譯器提供 **ANSI C** 標準函數庫
  - 標準字串處理函數庫, **e.g. strcpy( ), islower( )**
  - 標準數學函數庫, **e.g. sin( ), sqrt( )**
  - **Hi-Tech** 特殊函數庫, **e.g. persist_check(), get_cal_data( )**

函數庫

- 提供標準 **ANSI C** 函數庫
  - 所有支援的函數庫可以在 **Index** 的目錄下，查詢 **“L”** 下的函數庫名稱。或可以參考 **User’s Manual** 的第七章的函數使用說明
- 不支援 **Peripheral Libraries**
  - 字串及記憶體處理函數庫
    - **strcpy**
    - **strcmp**
    - **strlen**
    - **strcat**
    - **memset**
    - **memcpy**
    - **memcmp**
    - **and more . . .**

字串及記憶體處理函數庫

- 字串及記憶體處理函數庫
  - **strcpy**
  - **strcmp**
  - **strlen**
  - **strcat**
  - **memset**
  - **memcpy**
  - **memcmp**
  - **and more . . .**

數學運算函數庫

- 數學運算函數庫
  - **sin**
  - **cos**
  - **tan**
  - **log**
  - **pow**
  - **exp**
  - **floor**
  - **sqrt**
  - **and more . . .**

字元處理及測試函數庫

- 字元處理及測試函數庫
  - **isalnum**
  - **isalpha**
  - **isascii**
  - **isupper**
  - **islower**
  - **isspace**
  - **isdigit**
  - **and more . . .**

函數庫

- 其它功能函數庫
  - **atof**
  - **ftoa**
  - **atoi**
  - **itoa**
  - **atol**
  - **ltoa**
  - **ldiv**
  - **rand**
  - **and more . . .**

課程結束!

感謝您對 **Microchip**的 支持
如有任何問題請電

0800-717-718 台灣技術服務專線
www.microchip.com.tw 台灣技術論壇