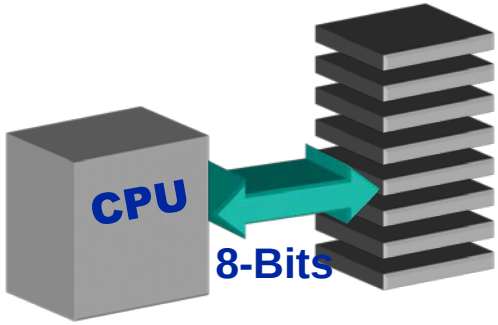
MICROCHIP

Regional Training Centers

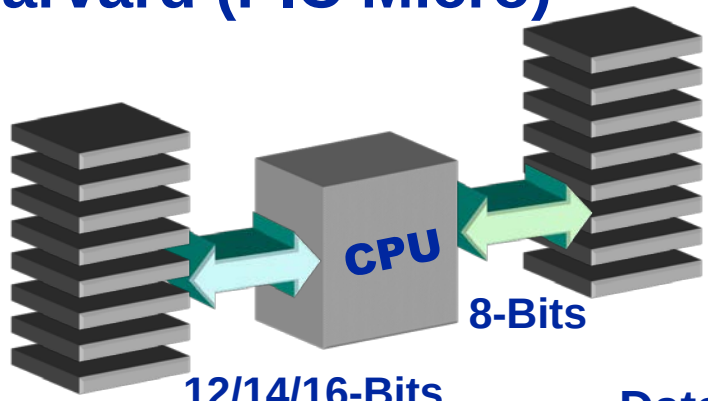
Microchip PIC18 MCU 架構
MPLAB C18 介紹與使用
MPLAB IDE 操作
PIC18F4520 基本 I/O

PICmicro® 的架構

Von Neumann (一般MCU)



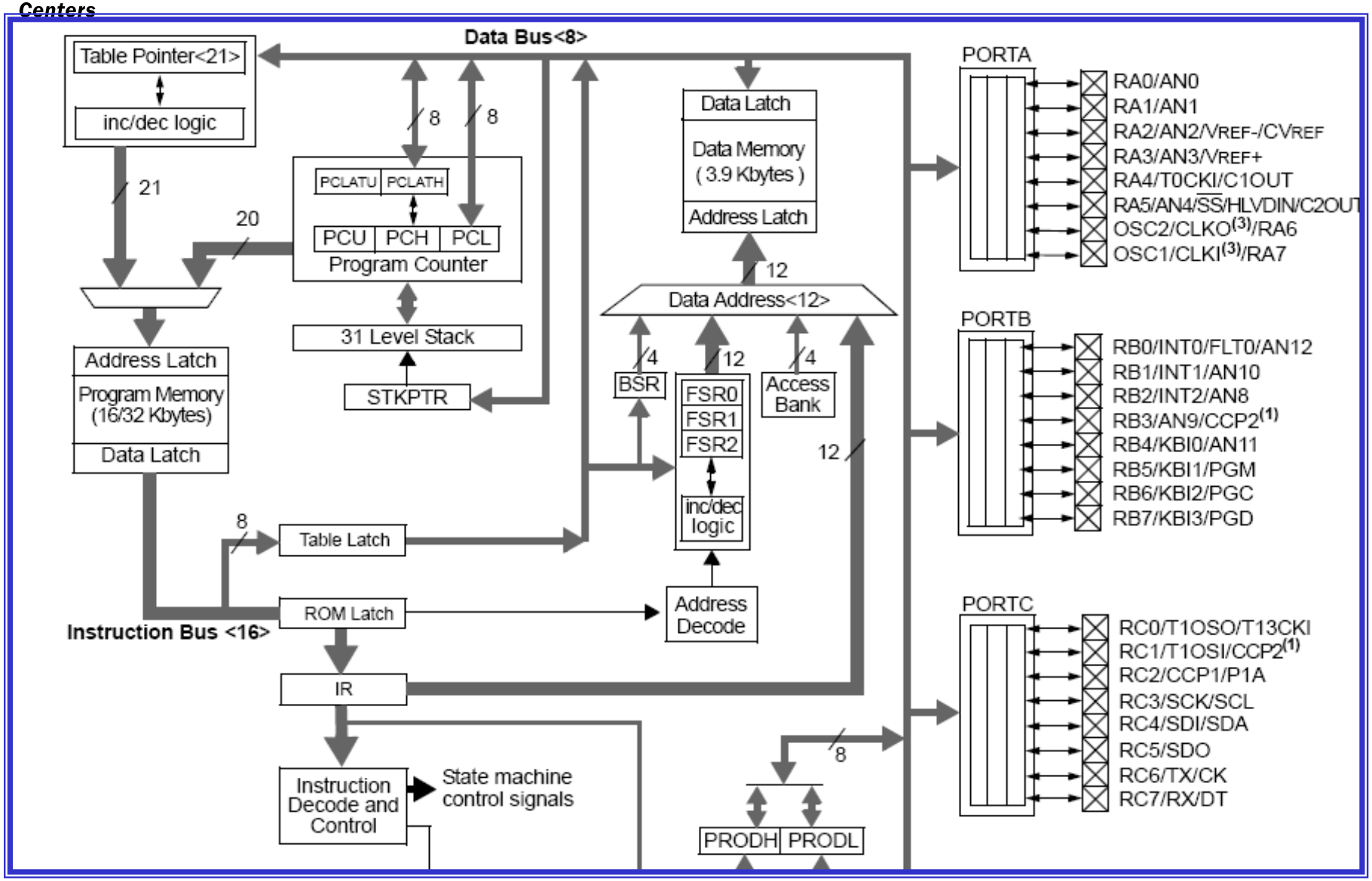
Harvard (PIC Micro)



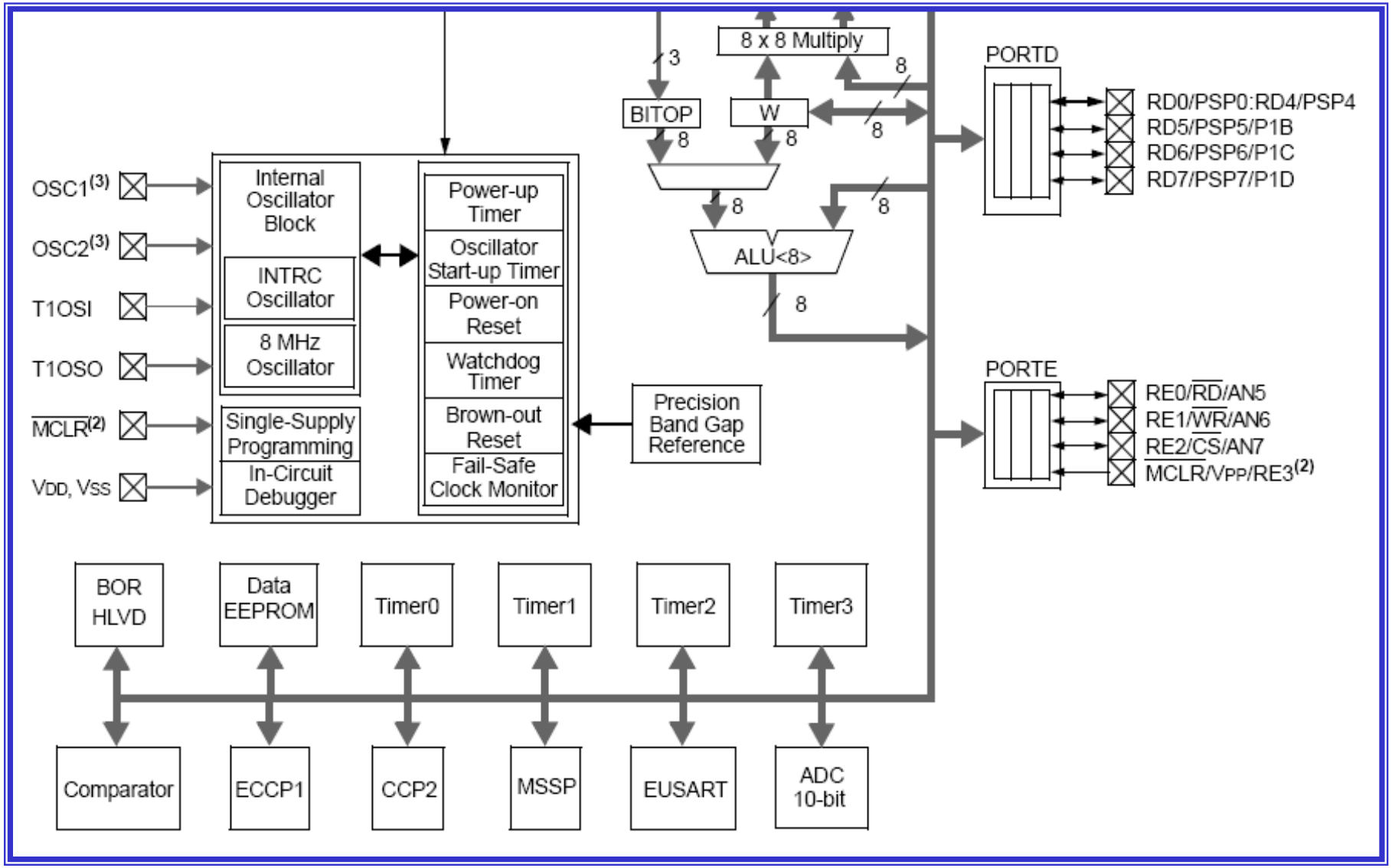
- 經由相同的記憶體及匯流排來提取程式及存取資料
  - 指令與資料無法有效率的同時被處理
  - 運作效率受到此結構影響而變慢
- 使用兩個不同的記憶空間與匯流排來存取程式及存取資料
  - 增加處理資料的效能與執行效率
  - 使得MCU可以具有不同寬度的程式記憶體與資料記憶體

MICROCHIP
Regional Training Centers

PIC18F4520 架構圖(一)

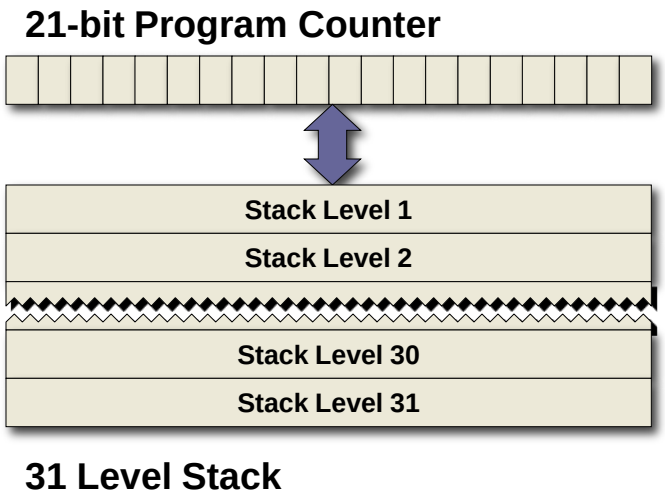


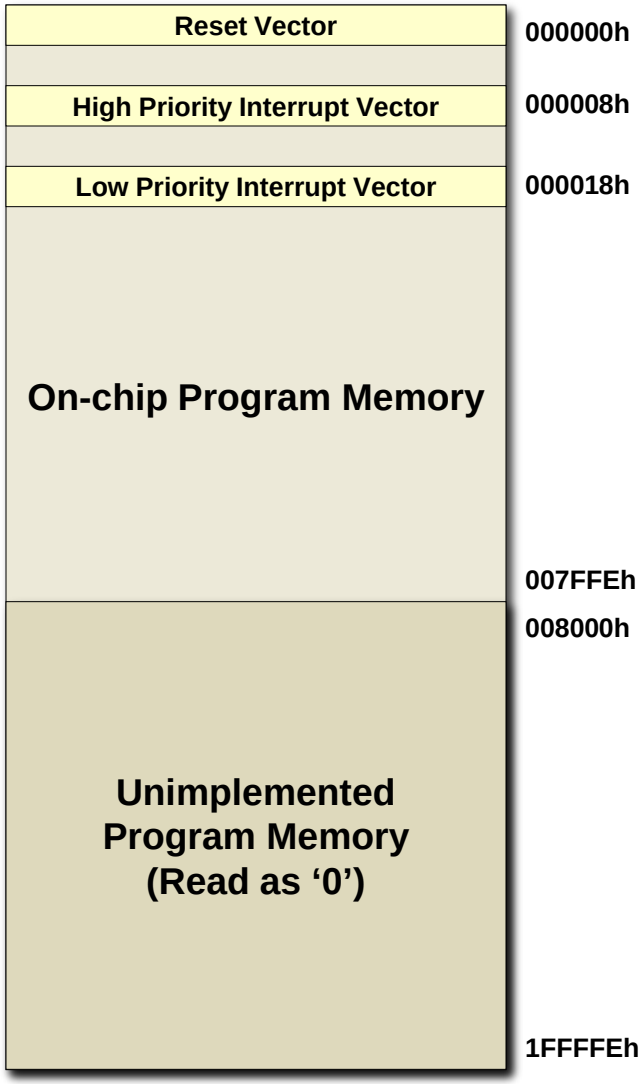
PIC18F4520 架構圖(二)



PIC18 Family 程式記憶體架構

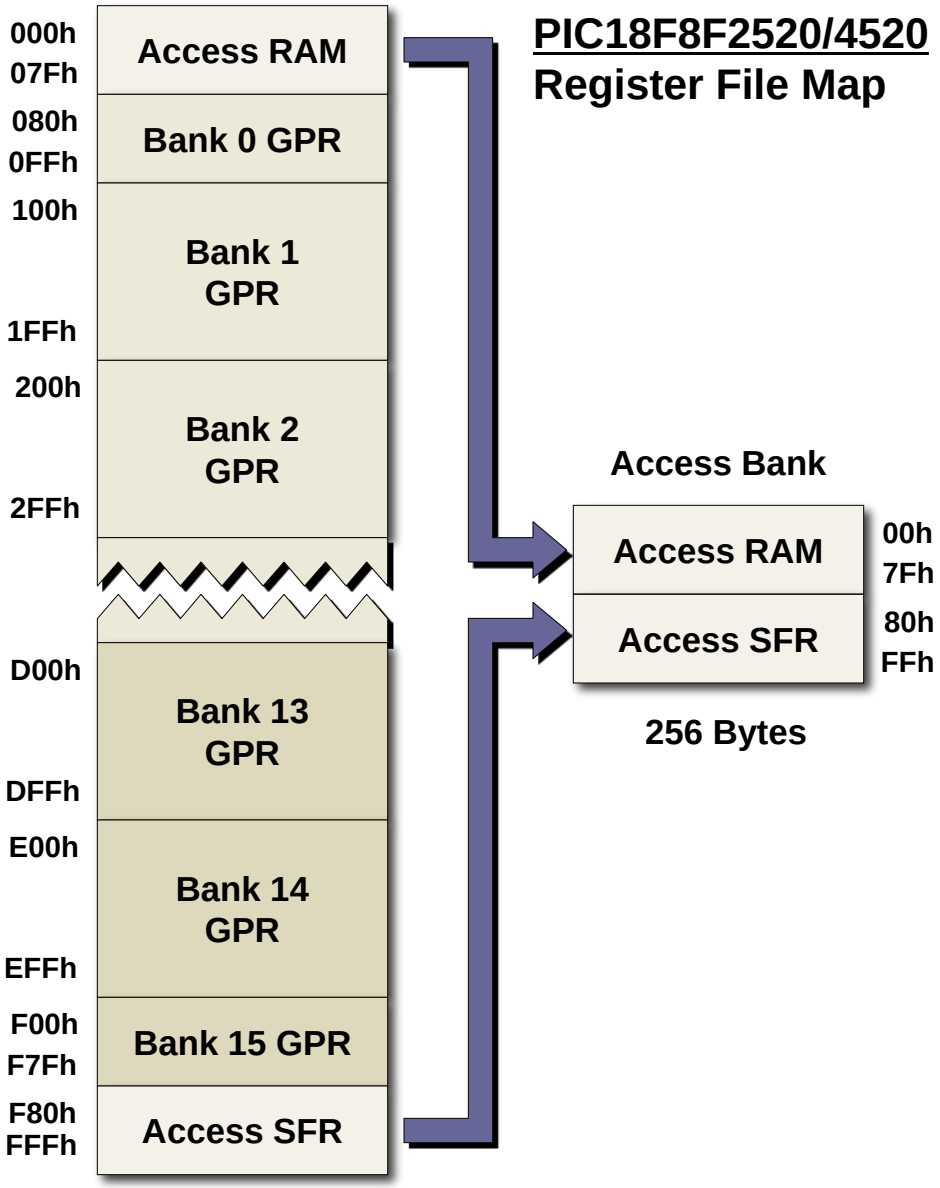
- 最高達 2MB (1 MW) , 連續而線性的單一程式記憶體空間





Data Memory Organization

- Data Memory 最大定址空間達 4k bytes
- 可以使用特定指令做線性的存取(需要兩個 Instruction words)
- 也可以 256 byte 為單位,使用 bank 的方式來定址 Data Memory(只需一個 Instruction word 但要配合 BSR 暫存器)
- bank 0 的前段部份與 bank 15的後段部份組合成一個特殊的 bank(Access Bank). 在特定指令中指定使用 Access Bank 時,不論 BSR 指向何處都會 Access 此 bank !



MICROCHIP
Regional Training Centers

PIC18 Family 指令集總覽

Byte Oriented Operations		
addwf	f,d,a	Add WREG and f
addwfc	f,d,a	Add WREG and Carry bit to f
andwf	f,d,a	AND WREG with f
clrf	f,a	Clear f
comf	f,d,a	Complement f
cpfseq	f,a	Compare f with WREG, skip =
cpfsgt	f,a	Compare f with WREG, skip >
cpfslt	f,a	Compare f with WREG, skip <
decf	f,d,a	Decrement f
decfsz	f,d,a	Decrement f, Skip if 0
dcfsnz	f,d,a	Decrement f, Skip if Not 0
incf	f,d,a	Increment f
incfsz	f,d,a	Increment f, Skip if 0
infsnz	f,d,a	Increment f, Skip if Not 0
iorwf	f,d,a	Inclusive OR WREG with f
movf	f,d,a	Move f
movff	f_s,f_d	Move f_s (src) to f_d (dst)
movwf	f,a	Move WREG to f
mulwf	f,a	Multiply WREG with f
negf	f,a	Negate f
rlcf	f,d,a	Rotate Left f through Carry
rlncf	f,d,a	Rotate Left f (No Carry)
rrcf	f,d,a	Rotate Right f through Carry
rrncf	f,d,a	Rotate Right f (No Carry)
setf	f,a	Set f
subfwb	f,d,a	Subtract f from WREG with borrow
subwf	f,d,a	Subtract WREG from f
subwfb	f,d,a	Subtract WREG from f with borrow
swapf	f,d,a	Swap nibbles in f
tstfsz	f,a	Test f, skip if 0
xorwf	f,d,a	Exclusive OR WREG with f

Bit Oriented Operations		
bcf	f,b,a	Bit Clear f
bsf	f,b,a	Bit Set f
btfsc	f,b,a	Bit Test f, Skip if Clear
btfss	f,b,a	Bit Test f, Skip if Set
btg	f,b,a	Bit Toggle f

MICROCHIP
Regional Training Centers

PIC18 Family 指令集總覽

Control Operations		
bc	n	Branch if Carry
bn	n	Branch if Negative
bnc	n	Branch if Not Carry
bnn	n	Branch if Not Negative
bnov	n	Branch if Not Overflow
bnz	n	Branch if Not Zero
bov	n	Branch if Overflow
bra	n	Branch Always
bz	n	Branch if Zero
call	n,s	Call subroutine
clrwdt		Clear Watchdog Timer
daw		Decimal Adjust WREG
goto	n	Go to address
nop		No Operation
pop		Pop top of return stack (TOS)
push		Push top of return stack (TOS)
rcall	n	Relative Call
reset		Software device RESET
retfie	s	Return from interrupt
return	s	Return from subroutine
sleep		Go into standby mode

Literal Operations		
addlw	k	Add literal and WREG
andlw	k	AND literal with WREG
iorlw	k	Inclusive OR literal with WREG
lfsr	f,k	Move 12-bit literal to FSR
movlb	k	Move literal to BSR<3:0>
movlw	k	Move literal to WREG
mullw	k	Multiply literal with WREG
retlw	k	Return with literal in WREG
sublw	k	Subtract WREG from literal
xorlw	k	Exclusive OR literal with WREG

Data Memory ⇔ Program Memory Operations	
tblrd*	Table Read
tblrd*+	Table Read with post-increment
tblrd*-	Table Read with post-decrement
tblrd+*	Table Read with pre-increment
tblwt*	Table Write
tblwt*+	Table Write with post-increment
tblwt*-	Table Write with post-decrement
tblwt+*	Table Write with pre-increment

Banked 直接定址模式

Regional Training Centers

‘a’ Bit from Instruction

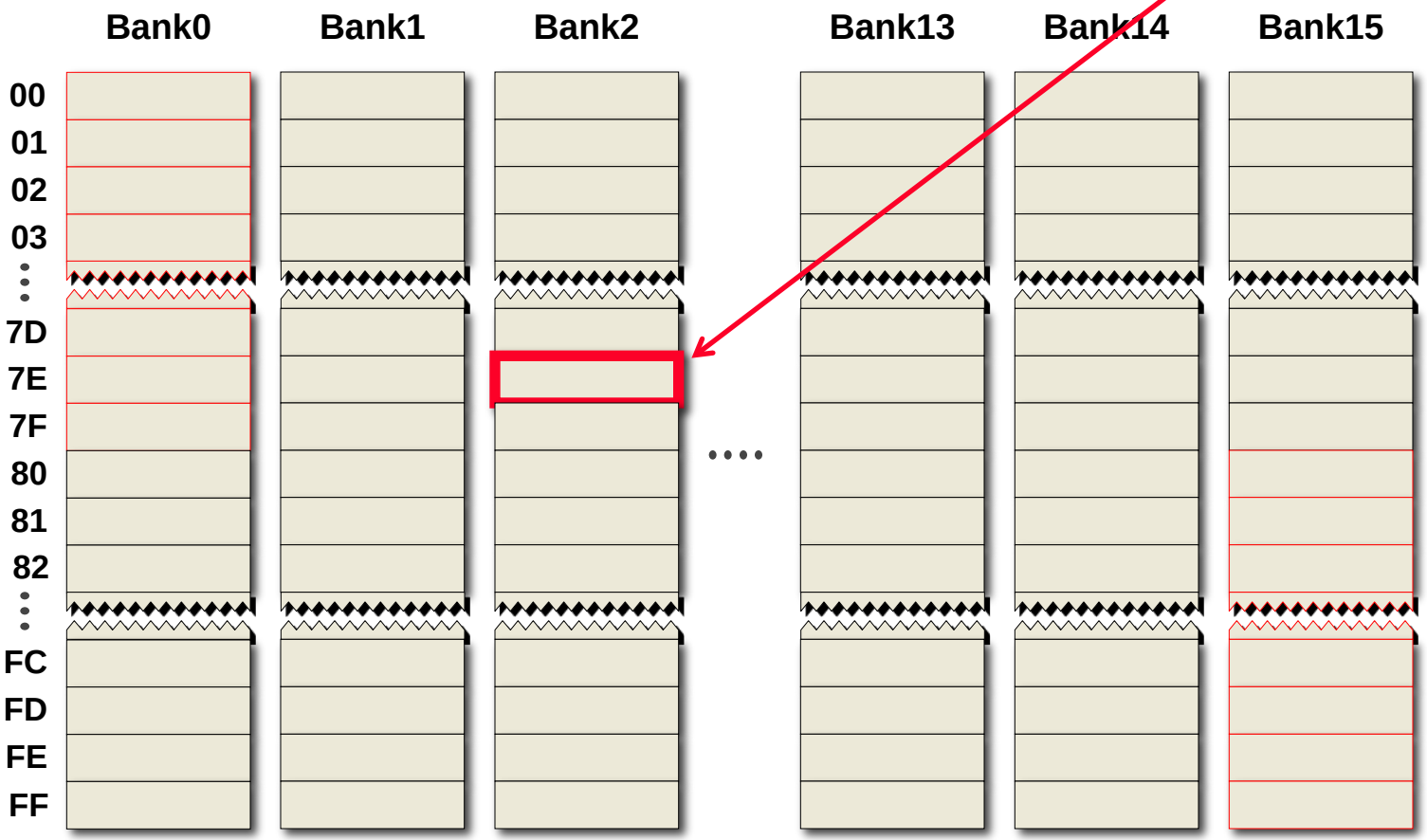
BSR

4-bits from BSR Register

‘f’ Operand

8-bits Encoded in Instruction

12-bit Effective Address (Use this when coding)



Banked 直接定址模式

‘a’ Bit from Instruction

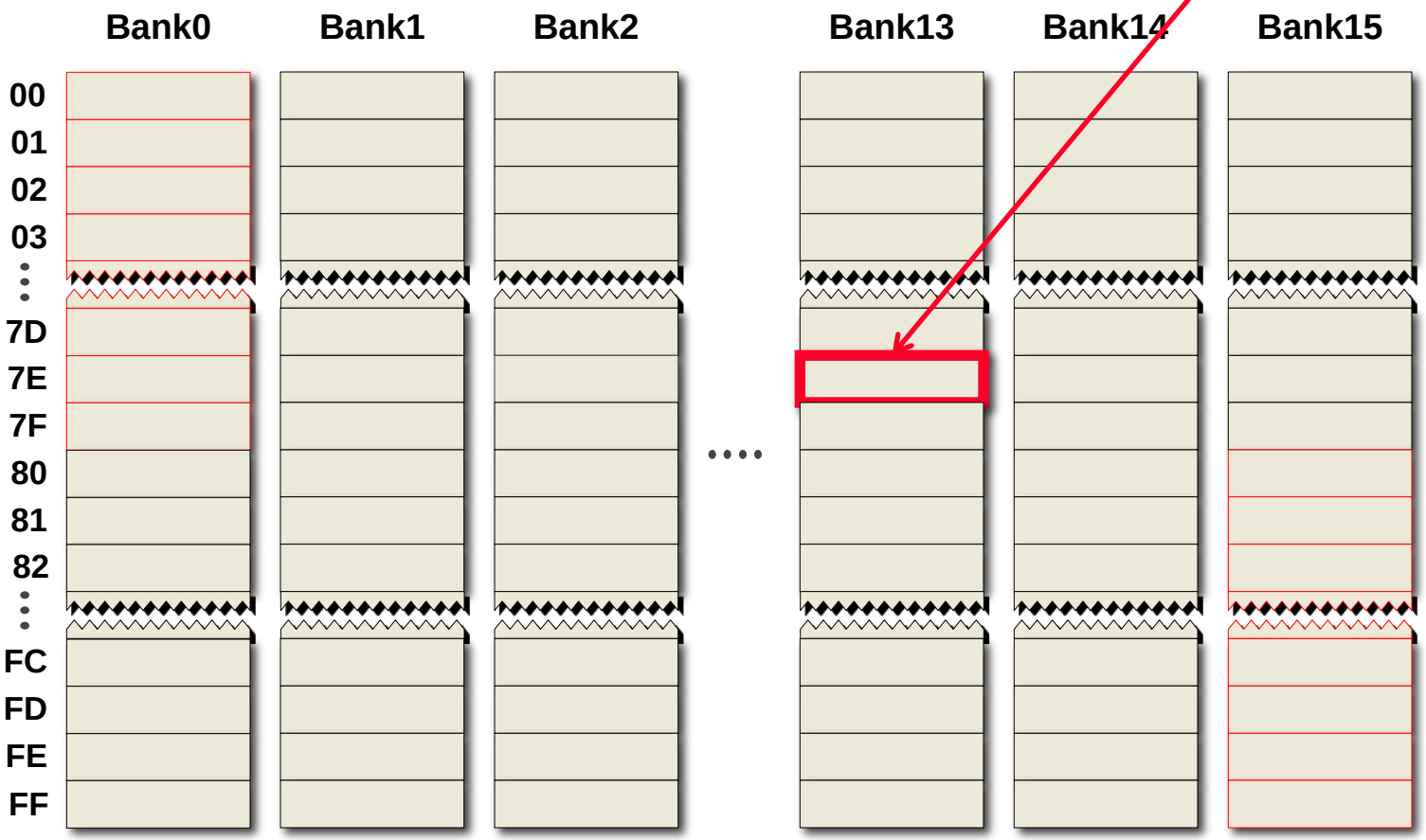
BSR

4-bits from BSR Register

‘f’ Operand

8-bits Encoded in Instruction

12-bit Effective Address (Use this when coding)



Access Bank 直接定址

‘a’ Bit from Instruction

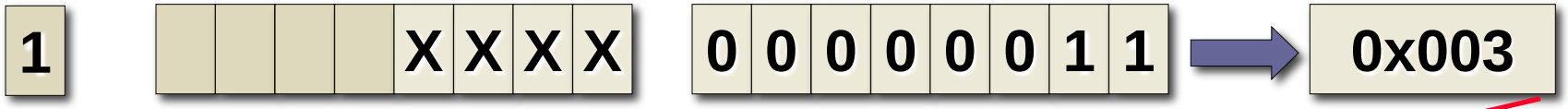
BSR

4-bits from BSR Register

‘f’ Operand

8-bits Encoded in Instruction

12-bit Effective Address (Use this when coding)



Access Bank 直接定址

‘a’ Bit from Instruction

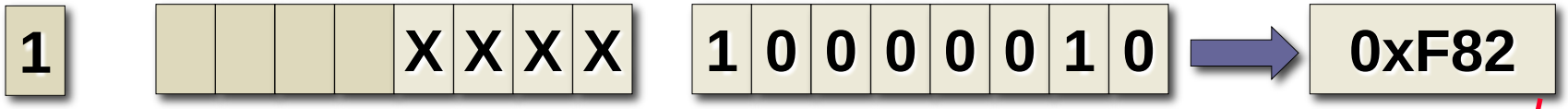
BSR

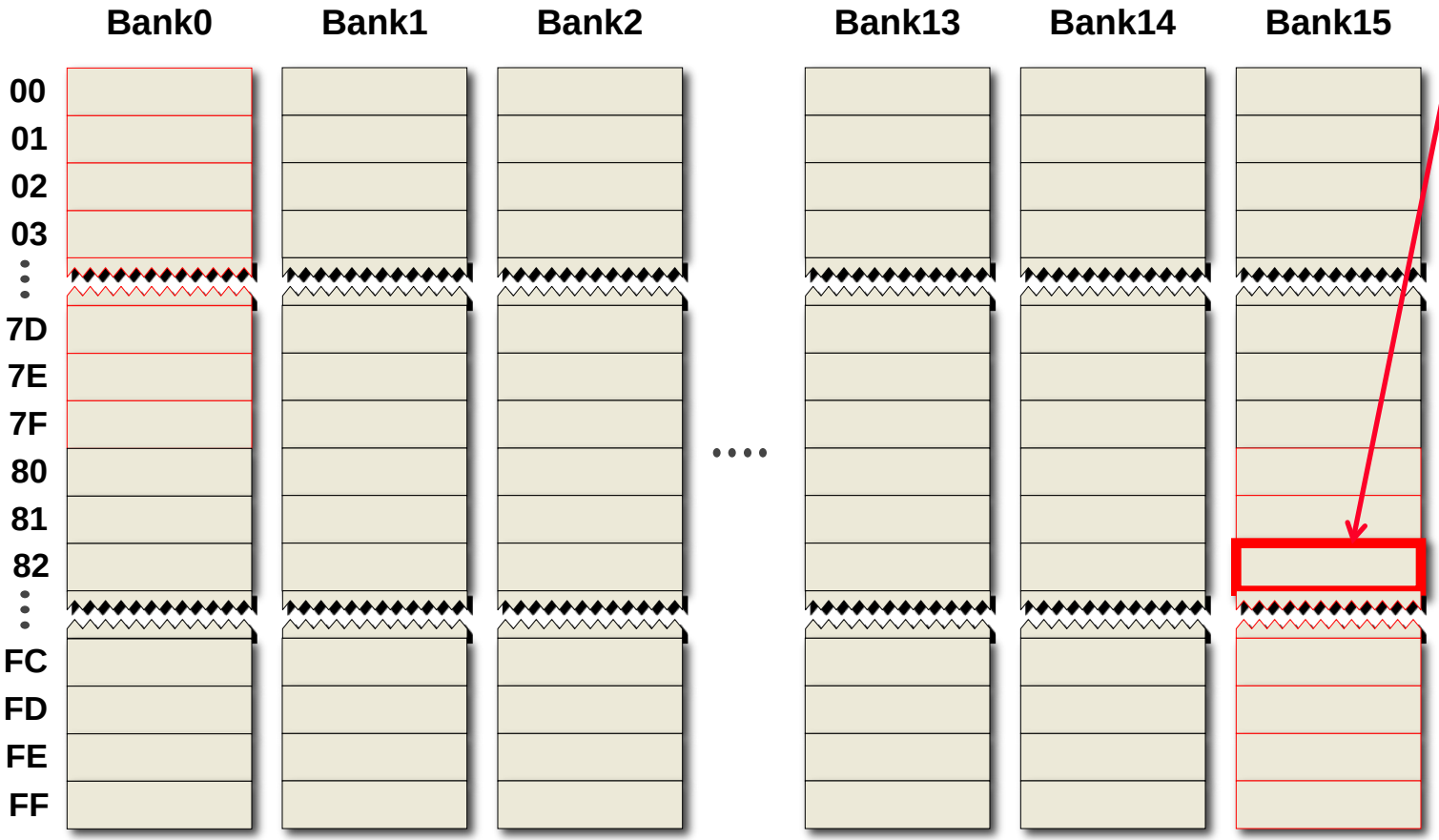
4-bits from BSR Register

‘f’ Operand

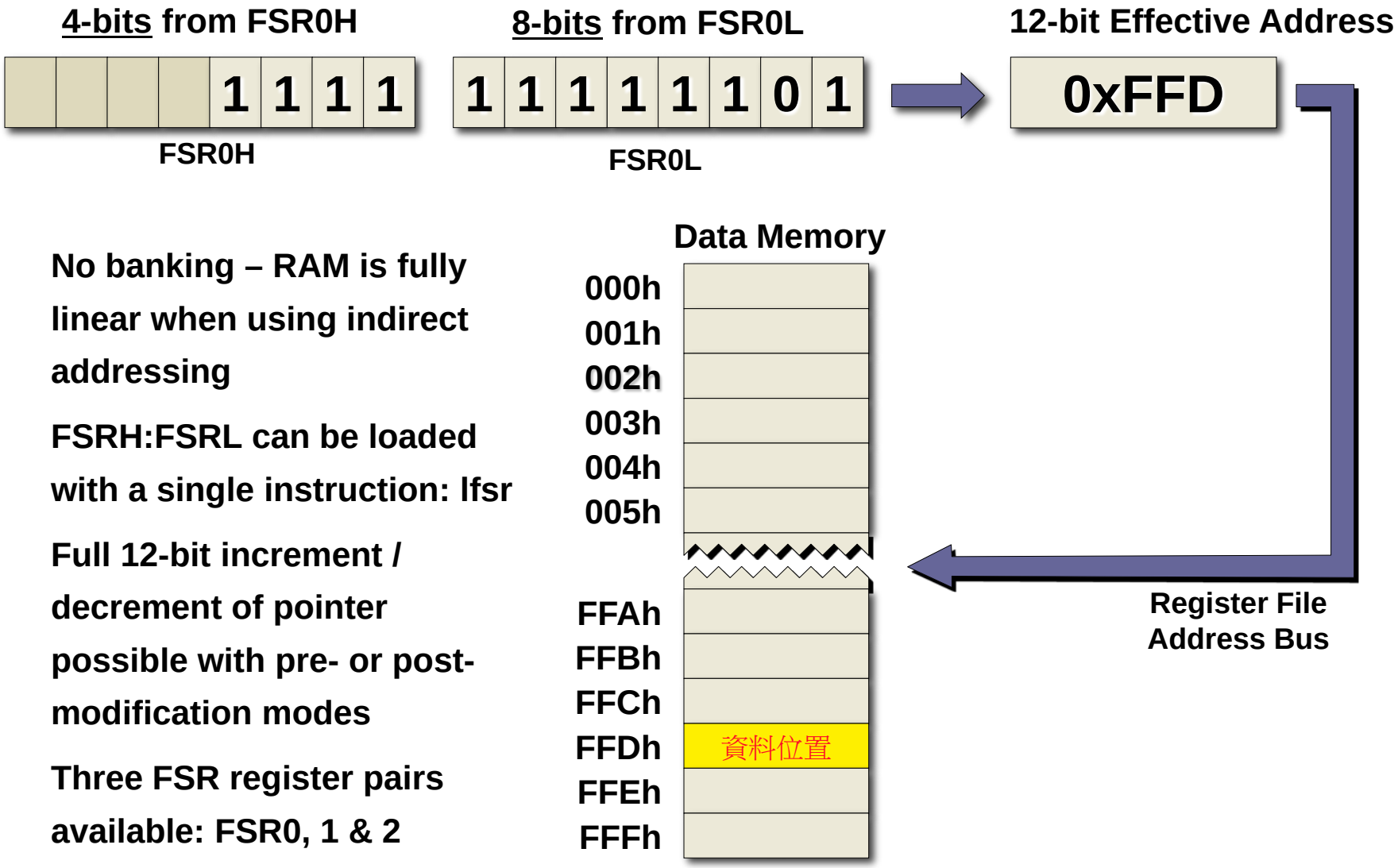
8-bits Encoded in Instruction

12-bit Effective Address (Use this when coding)



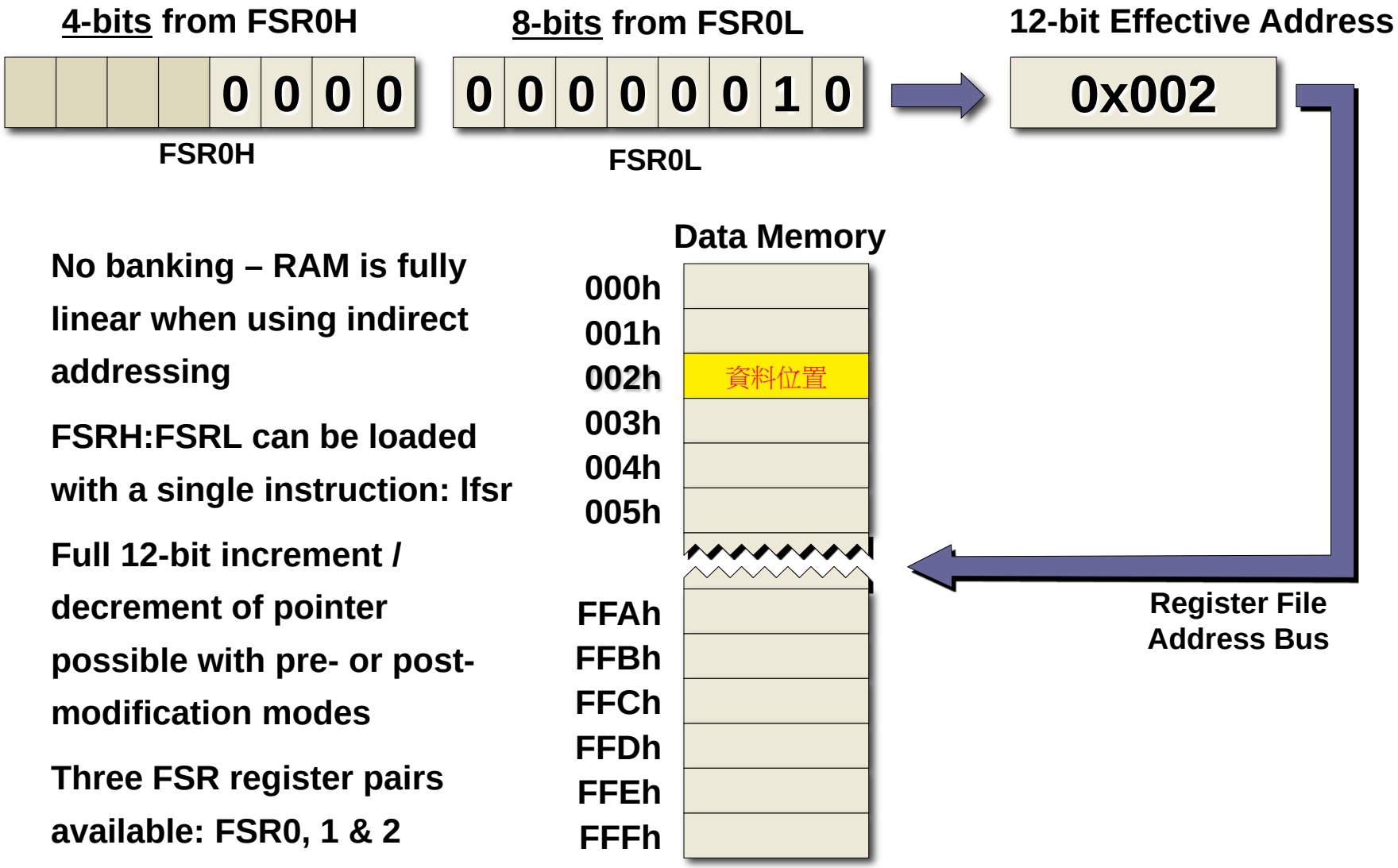


暫存器間接定址模式



- No banking – RAM is fully linear when using indirect addressing
- FSRH:FSRL can be loaded with a single instruction: lfsr
- Full 12-bit increment / decrement of pointer possible with pre- or post-modification modes
- Three FSR register pairs available: FSR0, 1 & 2

暫存器間接定址模式



- No banking – RAM is fully linear when using indirect addressing
- FSRH:FSRL can be loaded with a single instruction: lfsr
- Full 12-bit increment / decrement of pointer possible with pre- or post-modification modes
- Three FSR register pairs available: FSR0, 1 & 2

MICROCHIP
Regional Training Centers

PIC18 Family 的定址模式
Program Memory Access

Mode	Example Syntax
Absolute	`goto <addr>`
Relative	`bra <addr>`
Table Read / Write	`tblrd*  tblwt*`
Table Read / Write Post Increment	`tblrd*+  tblwt*+`
Table Read / Write Post Decrement	`tblrd*-  tblwt*-`
Table Read / Write Pre Increment	`tblrd+*  tblwt+*`

Table Reads

- **PIC18F Program Memory is Divided Into:**
  - User Memory
    - **Up to 128kB Internal**
    - **Up to 2MB External**
  - User IDs
    - **8 Modifiable Bytes**
  - Configuration Memory
    - **Device Settings, Code Protects, etc.**
  - Device IDs
    - **Part and Revision Signature**
- **Program Counter can only access User Memory (PC is 21-bits wide)**
- **Table Pointer can access all memory (TBLPTR is 22-bits wide)**

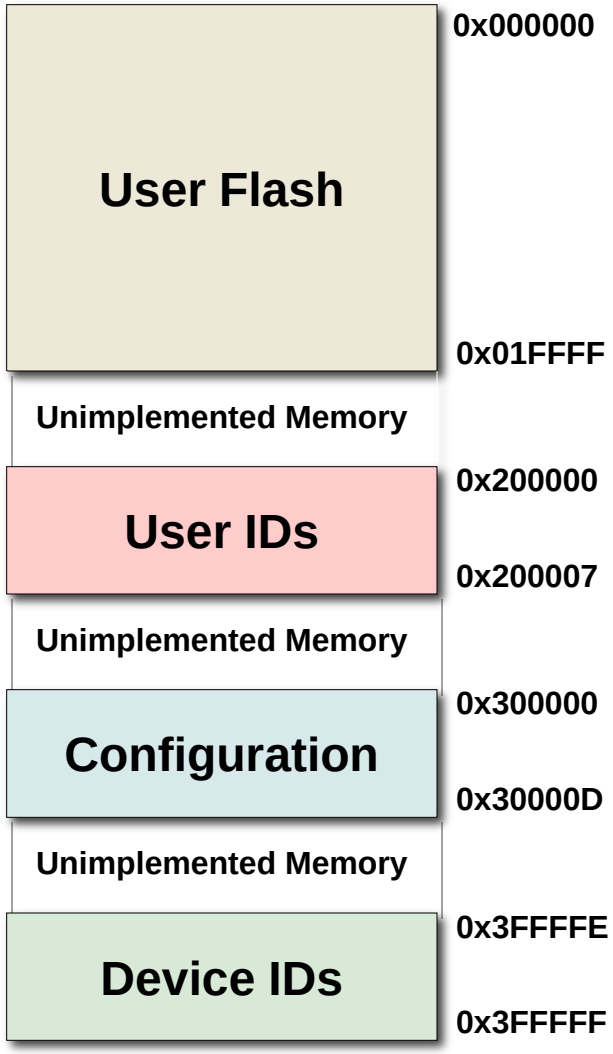
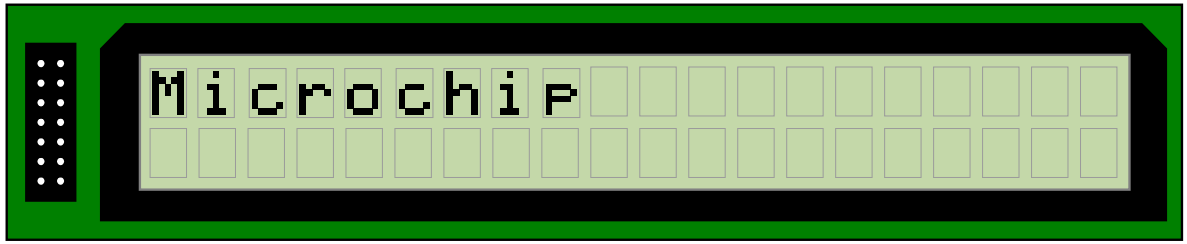


Table Read Example

- **Sending Characters to LCD**

How ASCII string tables are stored in program memory:



Program Memory		
Address	Contents	
6FFC	FF	FF
6FFE	FF	FF
7000	69	4D
7002	72	63
7004	63	6F
7006	69	68
7008	00	70
700A	FF	FF
700C	FF	FF

‘i’ @ 0x7001

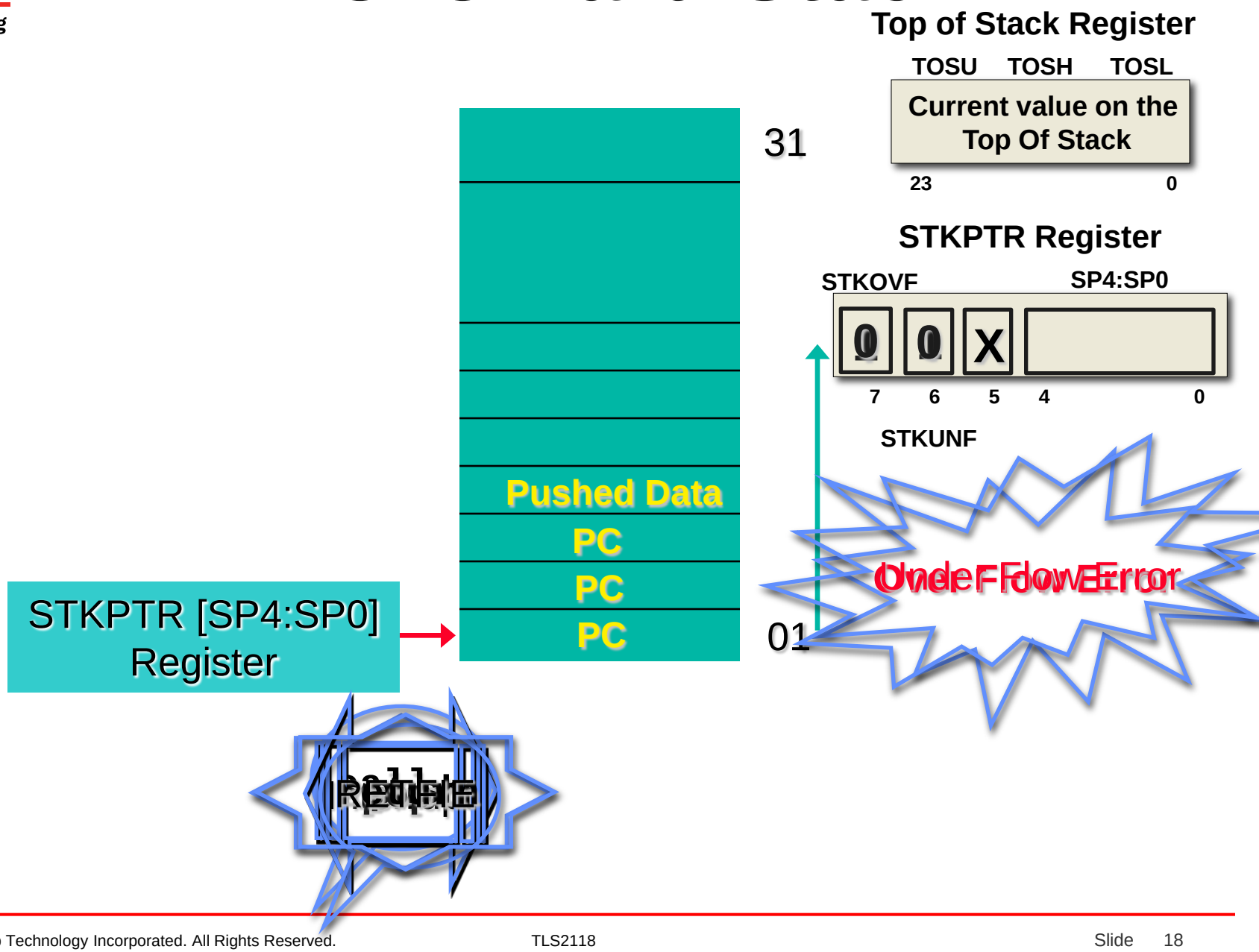
‘M’ @ 0x7000

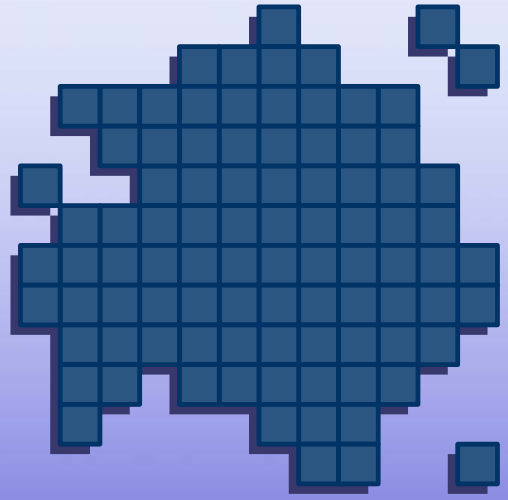
‘\0’ (NUL) @ 0x7009

Program Source Code:

```
        org  0x7000
MyStringTable
        DW   “Microchip\0”
```

PIC18 Hard Stack





MPLAB® C18

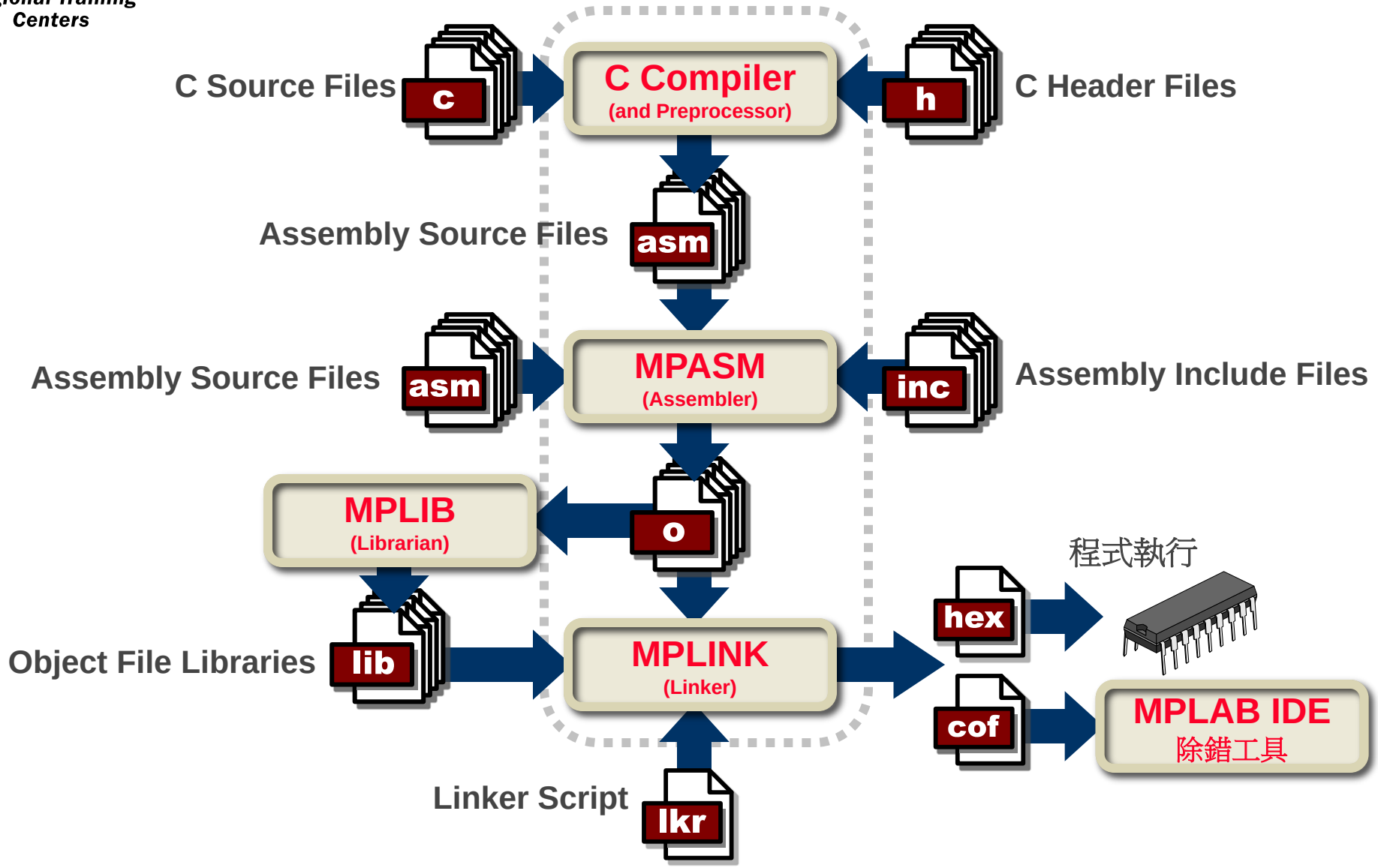
語言工具簡介

MPLAB® C18
概要說明

- 相容於 **ANSI x3.159-1989** 的 **C** 編譯器
- 具有最佳化處理功能
- 加入一些為 **PIC18** 特定周邊功能
- 跟 **MPLAB® IDE** 結合為一體
- 相容於 **MPASM** 及 **MPLINK**

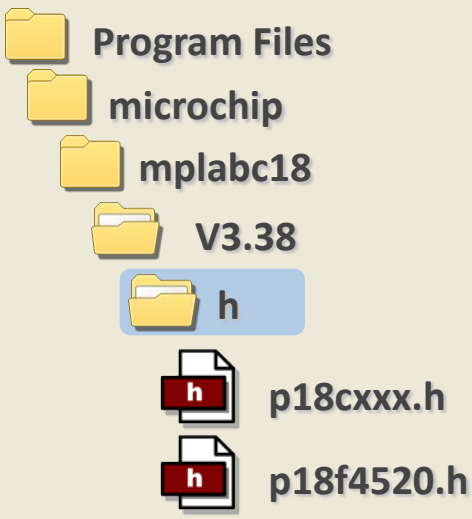
MPLAB® C for PIC18 Lite /Evaluation 的版本可以免費自 **Microchip** 的官網下載安裝。

C18 編譯流程



MPLAB® C for PIC18

標頭檔 (Header Files)



Header files 包含安裝 MPLAB C18 後所支援的所有的元件的名稱定義：

p18cxxx.h

p18cxxx.h 是一個通用型的標頭檔，他會依據 MPLAB IDE 所傳進來的使用元件料號加以判斷後，加入該元件的名稱定義標頭檔 (p18f4520.h)。

- 各元件的標頭檔提供了：
  - 提供所有暫存器的定義名稱
  - 定義周邊暫存器的位元操作名稱
  - 定義巨集指令以方便 C 來操作一些特殊指令

p18f4520.h 定義檔

- 定義18F4520 特殊功能暫存器(SFR) 的名稱及相關位元名稱
  - 位置 :C:\Program Files \Microchip\mplabc18\v3.38\h
- 定義一些常用的巨集
  - Nop( )
  - ClrWdt( )
  - Sleep( ) …
- 使用時用 #include <p18fxxx.h>
  - P18fxxx.h 為通用性標頭檔，他會一 MPLAB IDE 傳入的元件名稱自行判斷加入該元件的 h 檔。

為何要用 extern 的宣告?

有關ADCON0暫存器之定義

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

C18 的位元定義 (p18f4520.h)

- 所有的周邊暫存器及相關位元均在各標頭檔 **(Header file)** 中定義
  - 例: **PIC18F4520** 的各種周邊是定義在 “ **p18f4520.h** ”
- 撰寫程式時，只要按照 **Data Book** 所標示的名稱使用。對應的位元結構為暫存器名稱加上 **“bits”**

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

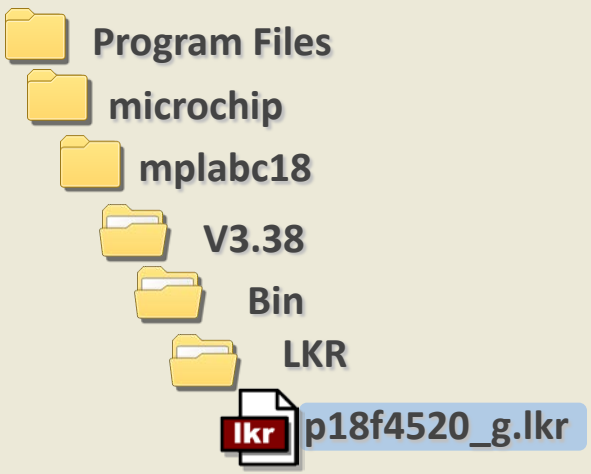
```
TRISD = 0 ;
PORTD = 0x55 ;
```

2. 將 PORTD 的 bit3 & bit5 設定為 1：

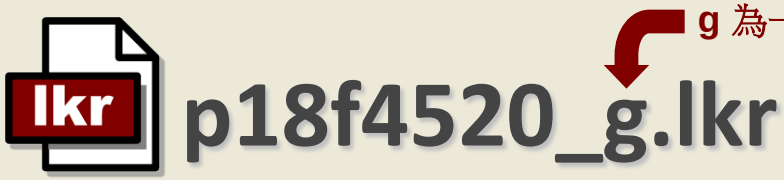
```
PORTDbits.RD3 = 1 ;
PORTDbits.RD5 = 1 ;
```

MPLAB® C for PIC18

連結描述檔 (Linker Scripts)



Linker Script files are included as part of the MPLAB installation :



- 每一個元件都有相對應連結描述檔:
  - 定義該元件的記憶大小、節區及邊界位址
  - 定義 **debug** 和 **release** 的工作模式
- **MPLAB IDE** 將自動選用連結描述檔，除非你有特別的指定使用

18f4520_g.lkr 描述檔

```
LIBPATH  .

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

將所要用到函數庫加入

程式空間

```
#IFDEF _DEBUGCODESTART
 CODEPAGE  NAME=page     START=0x0          END=_CODEEND
 CODEPAGE  NAME=debug    START=_DEBUGCODESTART  END=_CEND      PROTECTED
#ELSE
 CODEPAGE  NAME=page     START=0x0          END=0x7FFF
#FI

CODEPAGE  NAME=idlocs   START=0x200000     END=0x200007     PROTECTED
CODEPAGE  NAME=config   START=0x300000     END=0x30000D     PROTECTED
CODEPAGE  NAME=devid    START=0x3FFFFE     END=0x3FFFFF     PROTECTED
CODEPAGE  NAME=eedata   START=0xF00000     END=0xF000FF     PROTECTED
```

資料空間

```
 ACCESSBANK NAME=accessram  START=0x0          END=0x7F
#FI

DATABANK  NAME=gpr0     START=0x80         END=0xFF
DATABANK  NAME=gpr1     START=0x100        END=0x1FF
DATABANK  NAME=gpr2     START=0x200        END=0x2FF
DATABANK  NAME=gpr3     START=0x300        END=0x3FF
DATABANK  NAME=gpr4     START=0x400        END=0x4FF

#IFDEF _DEBUGDATASTART
 DATABANK  NAME=gpr5     START=0x500        END=_DATAEND
 DATABANK  NAME=dbgspr   START=_DEBUGDATASTART  END=_DEND       PROTECTED
#ELSE //no debug
 DATABANK  NAME=gpr5     START=0x500        END=0x5FF
#FI

ACCESSBANK NAME=accesssfr  START=0xF80        END=0xFFF       PROTECTED
```

軟體堆疊空間

```
#IFDEF _DEBUGDATASTART
  STACK SIZE=0x100 RAM=gpr4
#ELSE
  STACK SIZE=0x100 RAM=gpr5
```

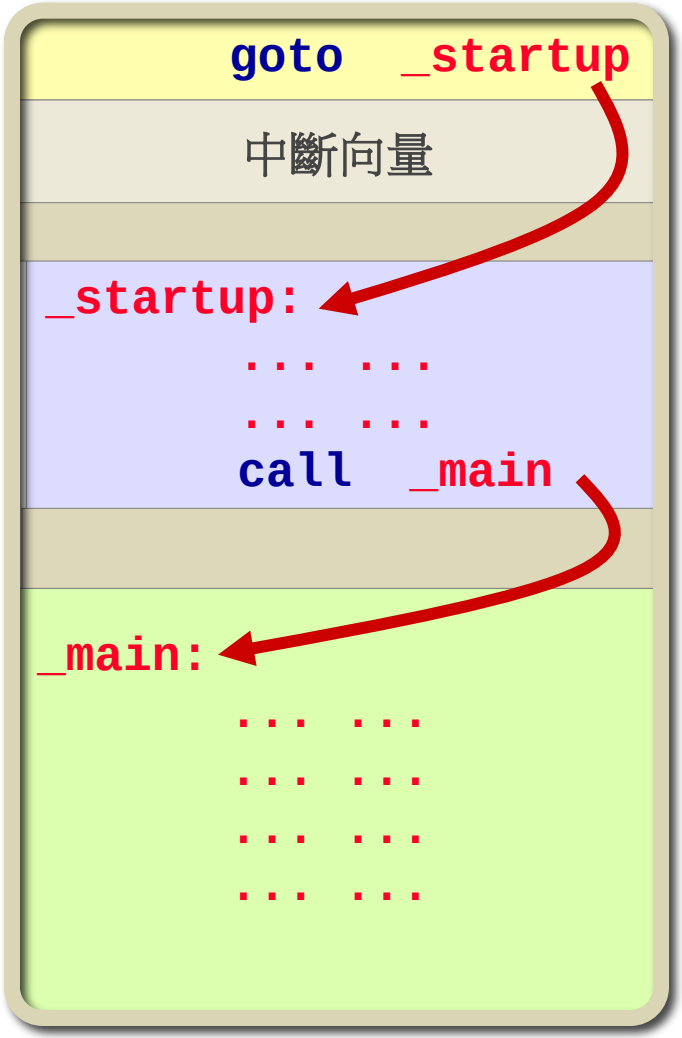
C 的 Linker 節區

Section Type	Default Name	Purpose
code	.code_*filename*	程式執行碼
romdata	.romdata_*filename*	程式記憶體的常數資料
udata	.udata_*filename*	無初始值的變數
idata	.idata_*filename*	有初始值的變數
stack (udata)	.stack	C 程式執行的軟體堆疊

- Linker 會對每一個輸入檔案 (*.o) 建立節區
- 所以你可以建立你自己的節區，並將相關的節區整合在一起放在記憶體裡

MPLAB® C for PIC18

啟動與初始設定



Reset Vector

MPLINK 會自動加入適當的啟動模組，一般是呼叫 c018i.o (_startup label)

c018i.o (內定使用)

C 啟動模組主要是做初始化的設定、軟體堆疊、指標設定等工作，設定完畢後再將控制權交給 main( ) (原始程式碼 : c018*.c)

main.c

你的 main( ) 函數，Linker 會自動安排執行位址的。

C 的啟動及初始設定

如何使用正確的啟動模組

如有使用到其它的啟動模組可以修改 lkr 檔裡的啟動模組的名稱

```
// File: 18f4520i.lkr
// Sample ICD2 linker script

LIBPATH .

FILES c018i.o
FILES TLS2118.lib
FILES clib.lib
FILES p18f4520.lib

CODEPAGE   NAME=page       START=0x0
CODEPAGE   NAME=debug      START=0x7D
CODEPAGE   NAME=idlocs     START=0x20
CODEPAGE   NAME=config     START=0x30
CODEPAGE   NAME=devid      START=0x3F
CODEPAGE   NAME=eedata     START=0xF0
```

修改啟動模組，共有六項可選用：

使用 PIC18F 傳統 指令集

- **c018i.o** idata & udata
- **c018iz.o** idata & zeroed udata
- **c018.o** udata only

使用 PIC18F 擴展型指令集

- **c018i_e.o** idata & udata
- **c018iz_e.o** idata & zeroed udata
- **c018_e.o** udata only

c018i.c 啟動模組程式

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

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

  _do_cinit ( );

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

RESET 位址: 0x000000

給予 STACK 及 TBLPTRU 初使值

設定初始變數值

將控制權交至 main( )

p18f4520.lib 周邊函數庫檔

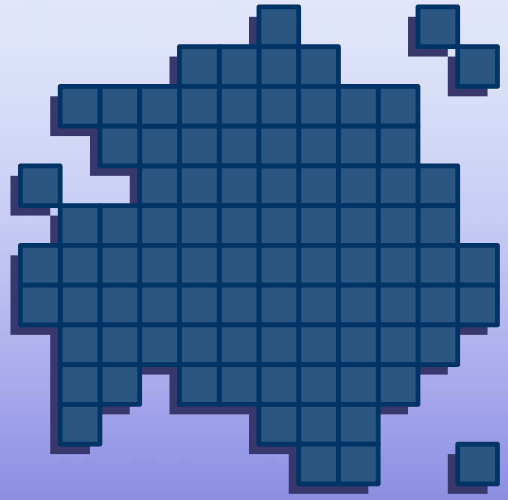
- 定義 **18F4520** 所有的特殊周邊暫存器 **(SFR)** 的位址
  - p18F4520.asm 組譯後以 obj 的型態存在於 p18f4520.lib
  - 原始程式為組合語言型態放在
    - **C:\Program Files\Microchip\mplabc18\v3.38\src\traditional\proc**
- **p18f4520.lib** 也提供各周邊控制函數
  - 相關的周邊控制函數，各自分門別類的以 obj 型態存在於 p18f4520.lib
  - 周邊控制函數請參閱 MPLAB C18 Reference Manual
    - **A/D , USART , Timer x , EEPROM …** 等
    - 原始程式在 **C:\Program Files\Microchip\mplabc18 \v3.38\src\pmc_common** 的目錄下

PIC18F4520 三個元件支援檔

- **p18f4520.o (asm)**
  - 定義周邊暫存器的位址給 C18 使用
- **p18f4520.h**
  - 定義周邊暫存器及相關位元名稱
- **p18f4520.inc**
  - 是給組合語言使用的周邊暫存器定義含入檔，不可混淆

有關特殊暫存器之位址定義

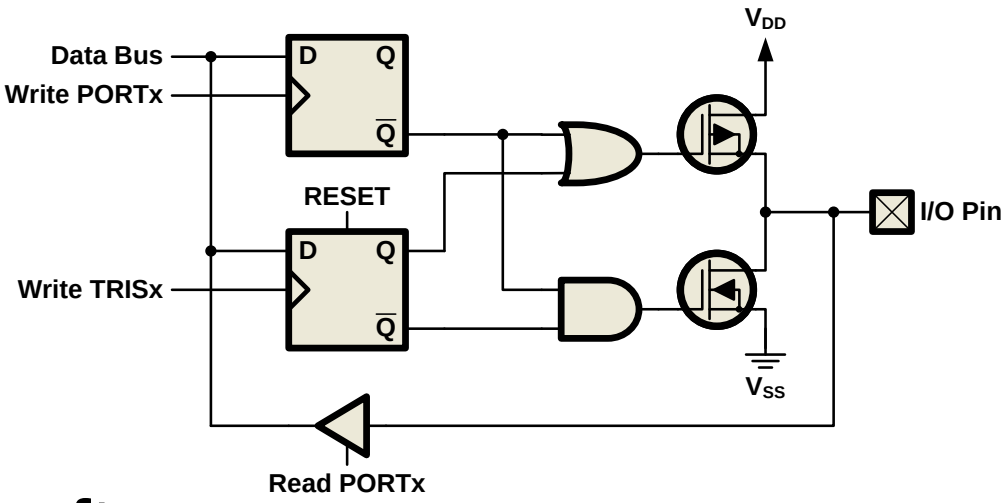
```
  LIST P=18F4520
       NOLIST
;-----------------------------------------------
; MPLAB-Cxx  PIC18F4520 processor definition module
;
; (c) Copyright 1999-2007 Microchip Technology,
;-----------------------------------------------

SFR_UNBANKED0       UDATA_ACS H'F80'

PORTA
PORTAbits           RES 1     ; 0xF80
PORTB
PORTBbits           RES 1     ; 0xF81
PORTC
PORTCbits           RES 1     ; 0xF82
PORTD
PORTDbits           RES 1     ; 0xF83
PORTE
PORTEbits           RES 1     ; 0xF84
                    RES 4
LATA
LATAbits            RES 1     ; 0xF89
LATB
LATBbits            RES 1     ; 0xF8A
LATC
LATCbits            RES 1     ; 0xF8B
LATD
LATDbits            RES 1     ; 0xF8C
LATE
LATEbits            RES 1     ; 0xF8D
```



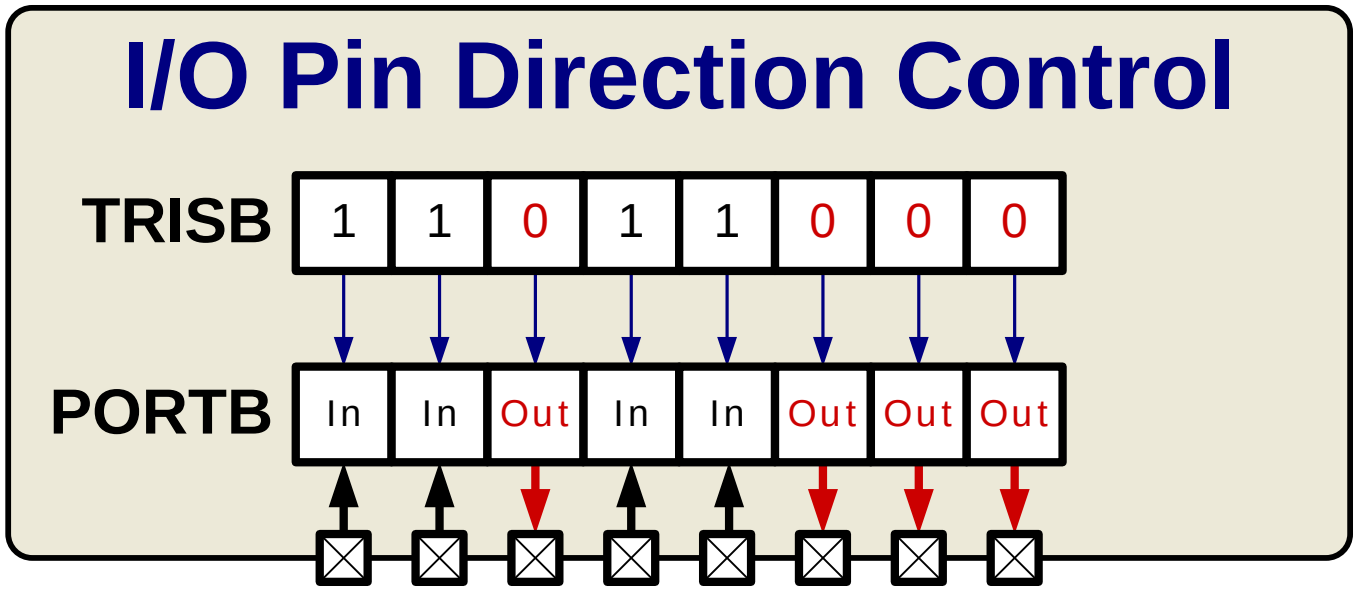
PIC18 Family 內建的基礎周邊說明 - I/O

I/O Ports

- High Drive Capability
- Can directly drive LEDs
- Direct, single cycle bit manipulation
- Each pin has individual direction control under software
- All pins have ESD protection diodes
- Pin RA4 is usually open drain
- 開機時，所有的 I/O 腳內定均設為輸入 (高輸入阻抗)，所以要注意沒有被使用到的 I/O 腳浮接狀態
- 注意：如果該 I/O 腳位有類比功能 (如 AD 輸入)，開機時其內定功能為類比的輸入功能

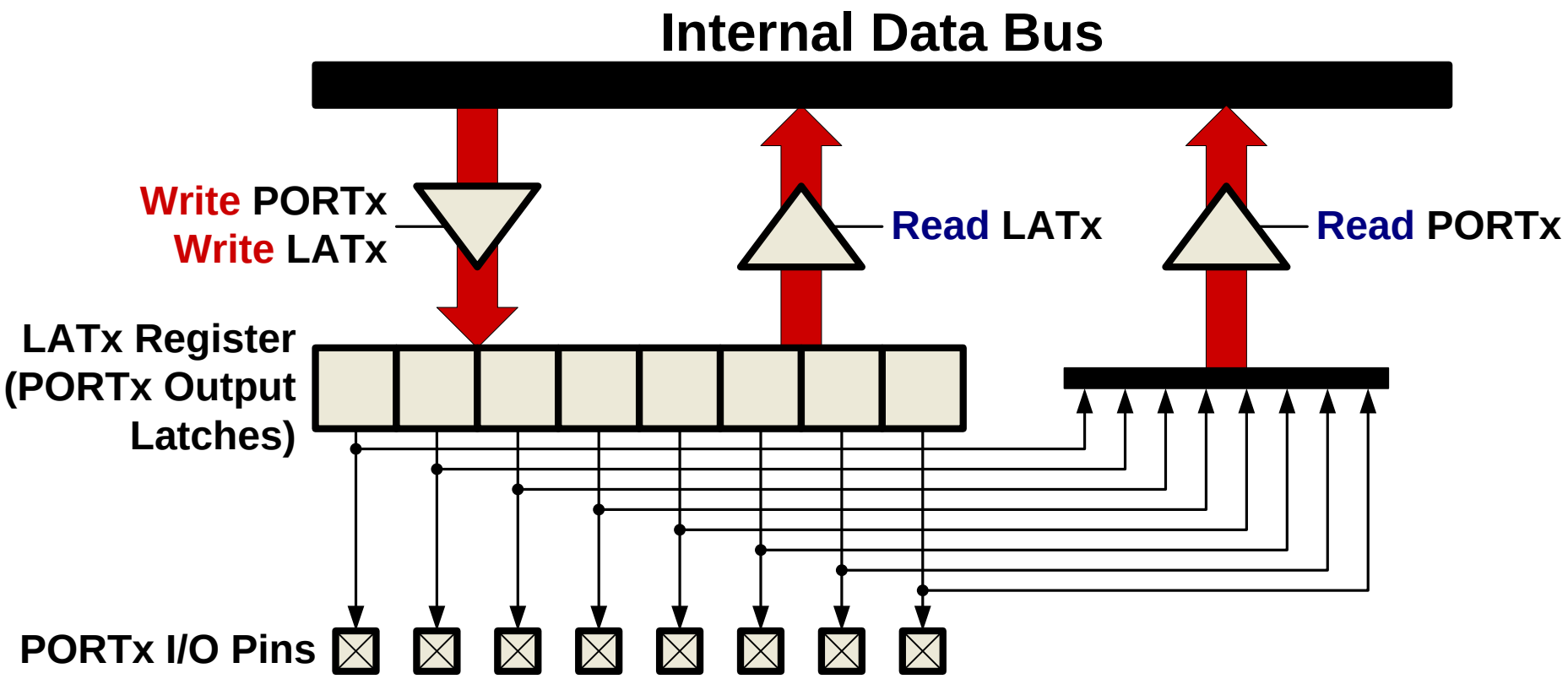


I/O Ports



- 位於 **TRISx** 暫存器中的 **Bit n** 決定在 **PORTx** 中的第 **n** 個腳位的方向
- **1 = Input, 0 = Output**

Digital I/O Ports – PIC18

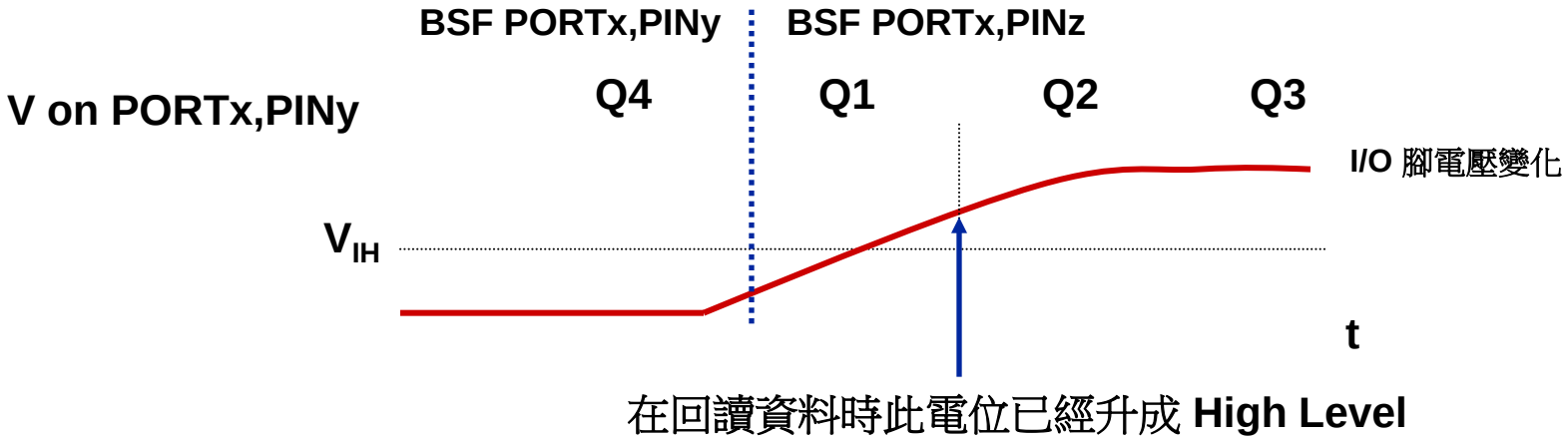


注意一下：**PORTx 與 LATx 在做輸出與輸入時，功能上有何不同？**

為何要加入 LATx 的暫存器

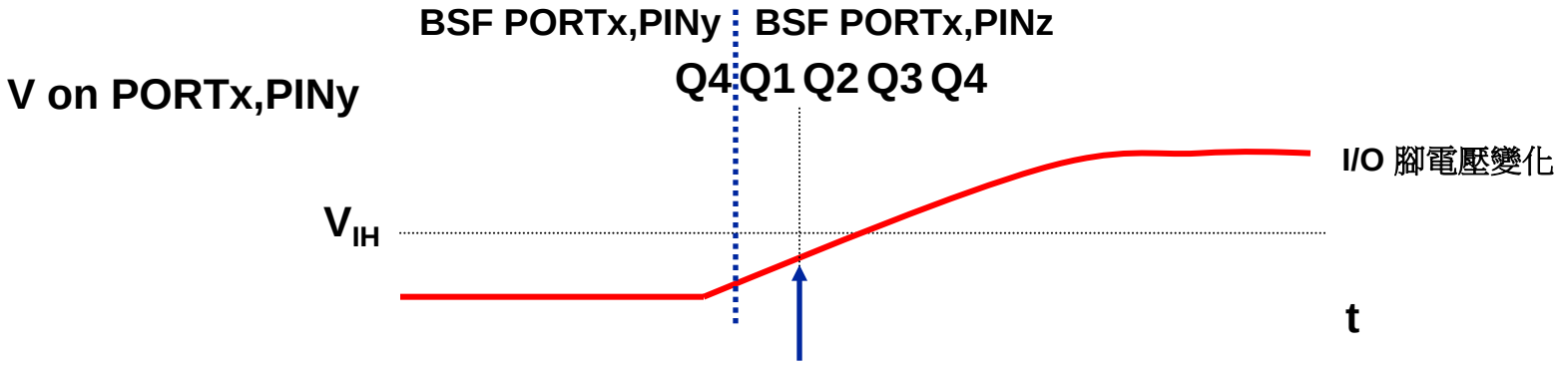
消除 Read-Modify-Write 動作的風險

在低工作頻率及較小的雜散電容



在回讀資料時此電位已經升成 **High Level**

在高工作頻率及較大的雜散電容



在回讀資料時此電位還沒有變成 **High Level**，會出錯!

Analog or Digital I/O?

- 有些 **I/O** 腳位的功能與類比輸入是多工使用的 **(Power On** 後的預設值為 **“analog mode”)**
  - 所以在初始化程式段落並須將欲使用為 Digital I/O 的腳位規劃為 Digital mode !
- 在 **PIC18F4520** 中，**ADCON1** 被用來設定這些共用接腳的操作模式為何.

Bit 7							Bit 0
-	-	X	X	1	1	1	1
		VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0

Set lower 4 bits to ‘1’ to make all multiplexed pins digital

I/O Ports 總結

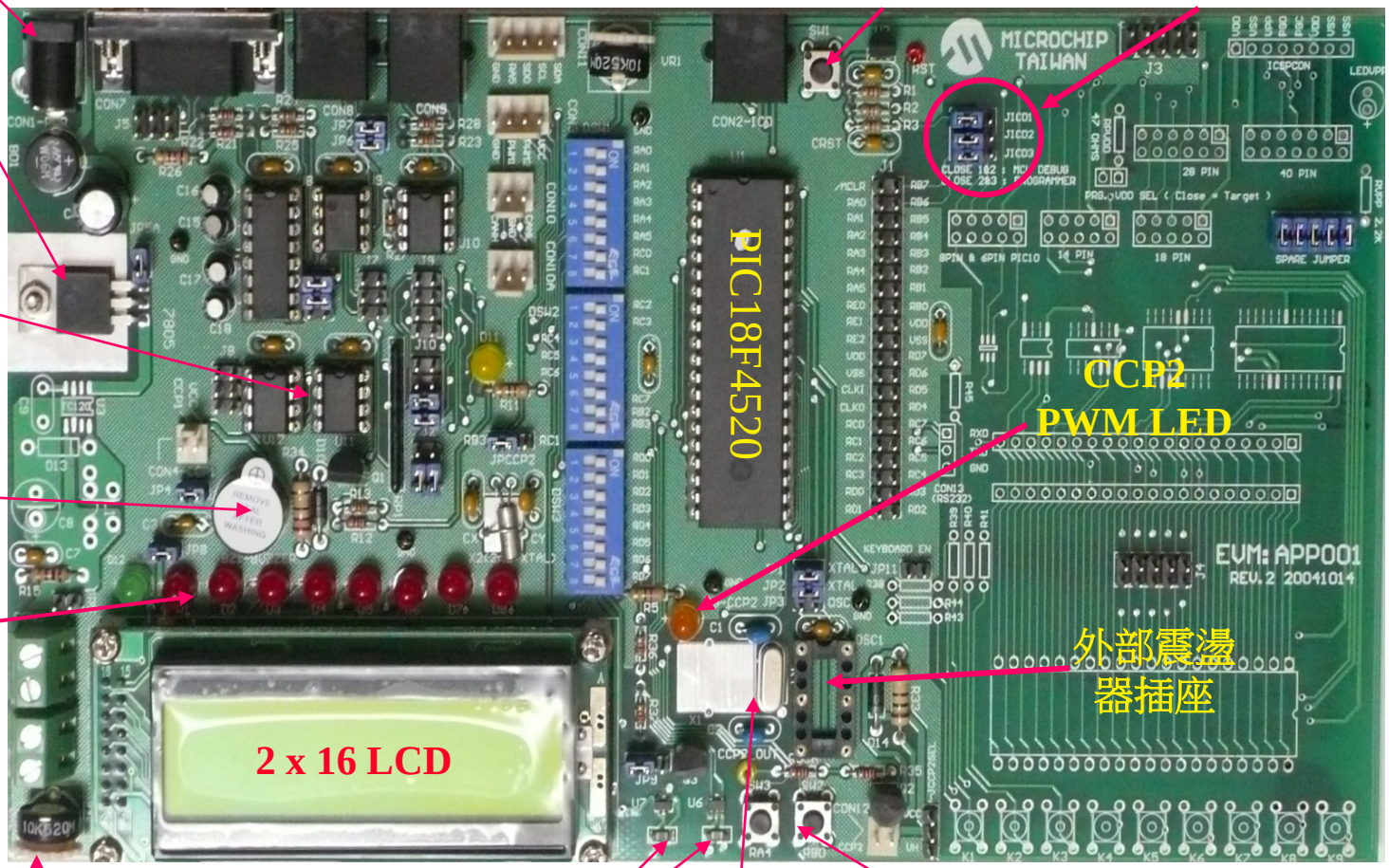
Summary

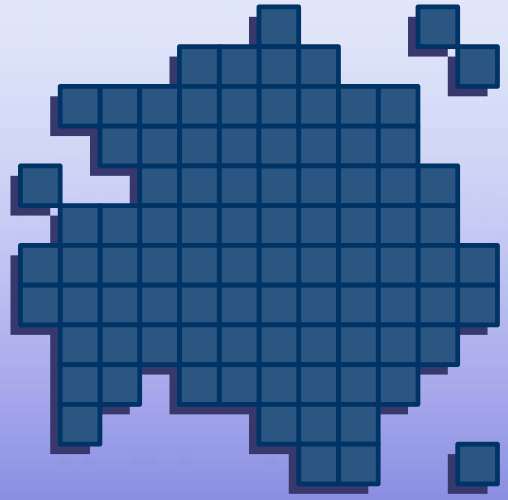
- 在 PIC18 Family 中使用於 I/O 控制的暫存器有
  - TRISx
    - 控制 I/O port 的方向
  - PORTx
    - 對 I/O port 讀取/寫入用的暫存器
  - LATx
    - 對 I/O port 寫入時的栓鎖暫存器 – 建議使用此暫存器進行位元運算的操作

APP001 實驗板使用手冊

- **APP001 實驗板的使用手冊、電路圖及出廠測試程式可以在 Microchip 台灣的網站下載:**
  - 教育訓練光碟的連結
    - **APP001 v3.0 實驗板電路圖**
    - **APP001 v3.0 中文使用手冊**
    - **APP001 V3.0 出廠測試程式**

認識 APP001 實驗板





練習一

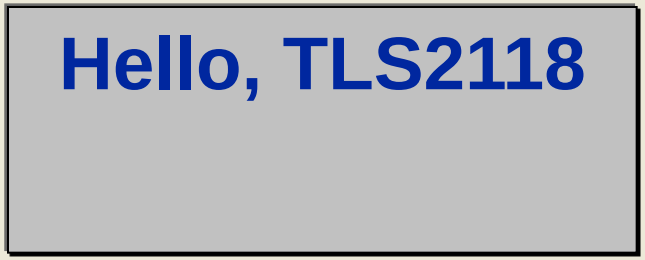
請照著後面的投影片的順序
建立新的專案、編譯、燒錄

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

練習一目標

建立第一個 **MPLAB C18** 的專案，使用 **C18** 來編譯此專案並確定成功後執行程式可以正確的看到 **“Hello, TLS2118!”** 的字串顯示在 **APP001 Demo Board** 的 **LCD.**



這是一步一腳印的練習，使用實際範例按步就班的建立一個 **MPLAB C18** 的基本專案。

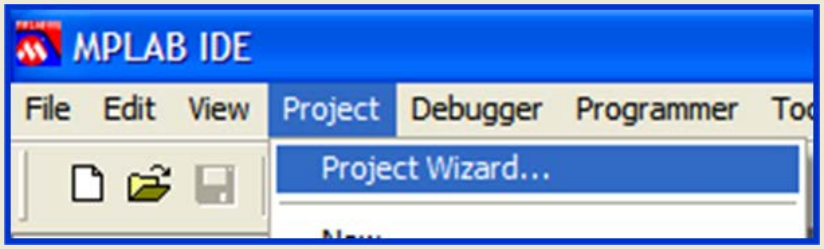
Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

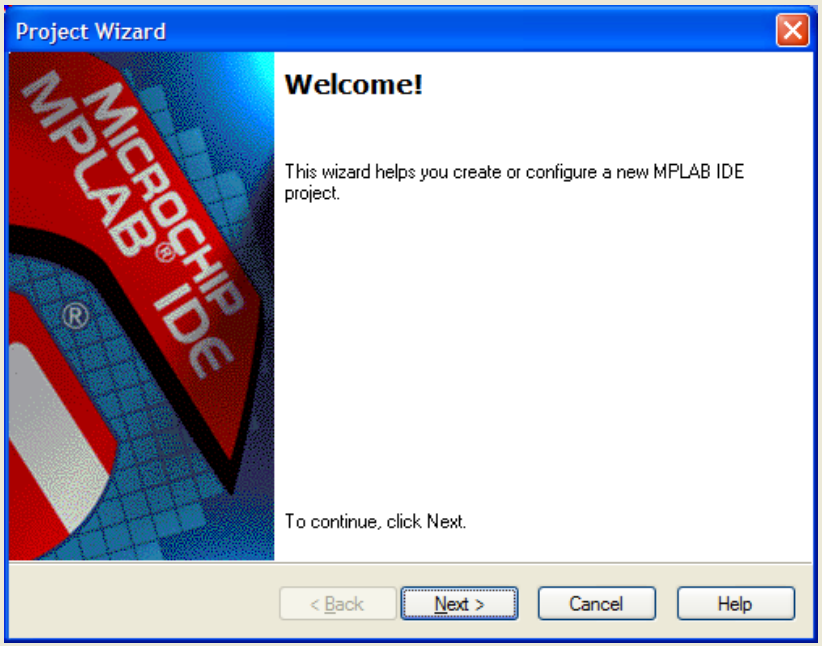
1. 在 **MPLAB IDE** 下的 **Project** 選項下，開啟專案精靈：

   Project ▸ Project Wizard…



After the Project Wizard opens,

Click Next > **to continue…**



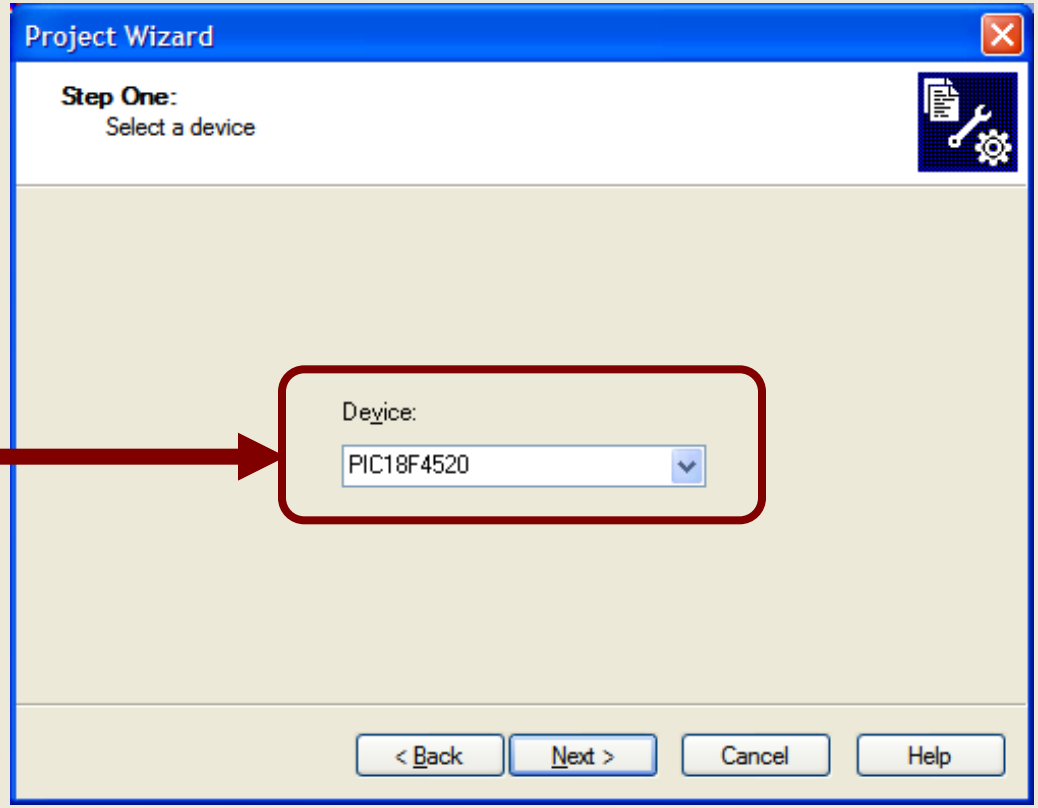
Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

2 選擇正確的使用元件名稱：

PIC18F4520



Click **Next >** to continue…

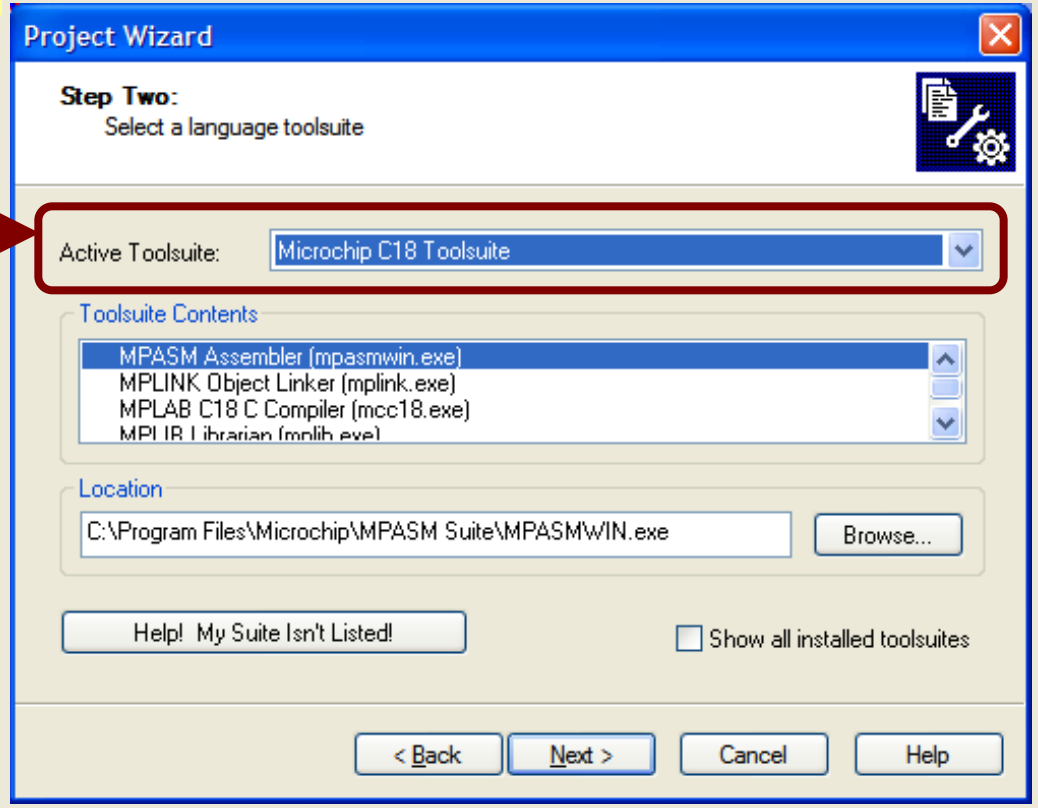
Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

3 在已安裝編譯器工具箱 (Toolsuite) 選擇：

Microchip C18 Toolsuite



Click **Next >** to continue…

Lab Exercise 1

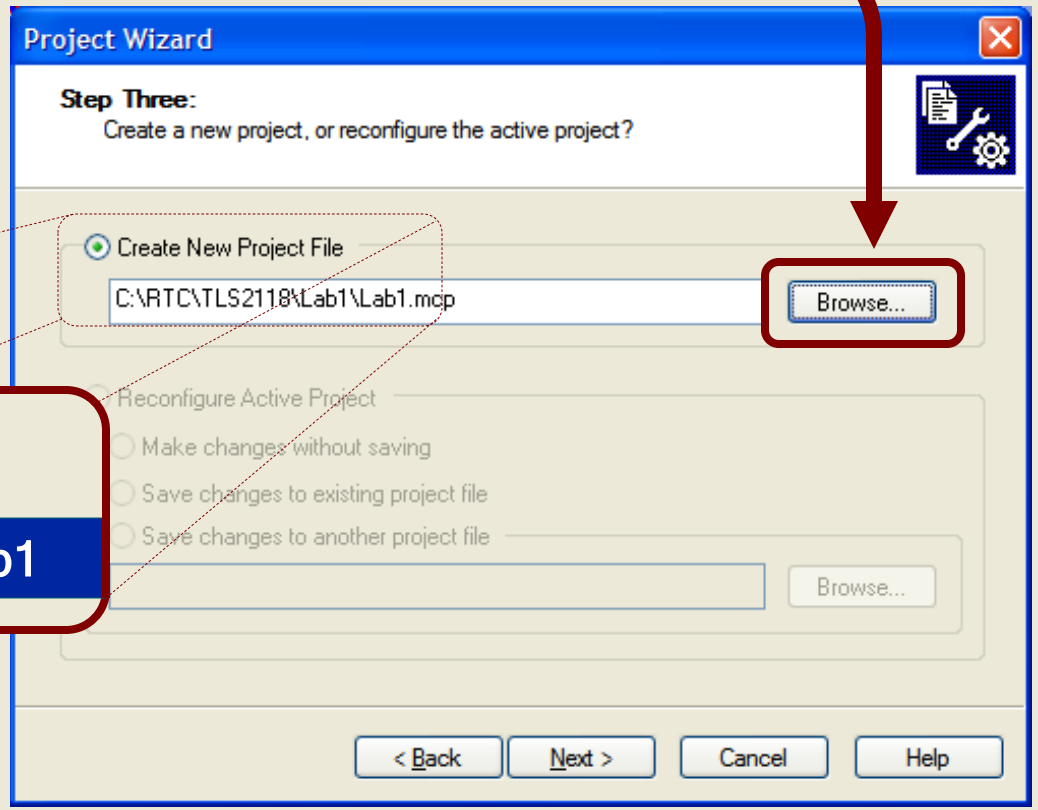
建立一個 MPLAB® C18 程式專案

步驟

4 按下 **Browse...** 給予專案名稱 及路徑：

C:\RTC\TLS2118\Lab1

專案的名稱為 **Lab1.mcp**



Create New Project File

C:\RTC\TLS2118\Lab1\Lab1

Click **Next >** to continue...

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

5 加入原始程式到專案裡，在左邊表列裡可以選擇：

C:\RTC\TLS2118\Lab1

選擇 **Lab1.c**

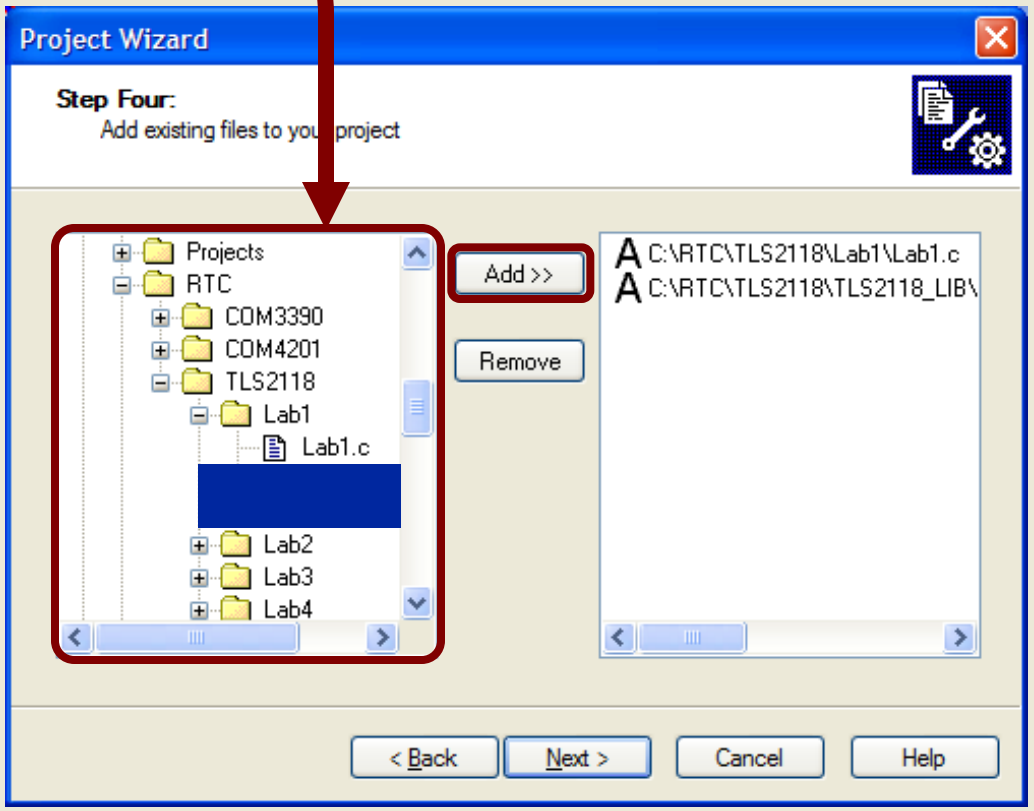
按下 **Add >>**

6 接著到底下的路徑加入所要使用的 **Library**：

C:\RTC\TLS2118\TLS2118_LIB

選擇此檔案 **TLS2118.lib**

Click **Add >>**



Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

- **MPLAB** 在較新的版本會自行為使用者選擇適當的連結器描述檔 **(LKR** 檔毋需指定**)**
- 使用者若自行加入連結器描述檔 ，將會取代原先 **MPLAB** 的預設選擇。
- 使用者可以為自己建立的專案加入一個連結器描述檔 **– Linker Script (**例如：**p18f4520_g.lkr)**
- 一般而言，除非使用者要修改程式區間的分配作業才須自行指定連結器描述檔

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

連結器描述檔的內容範例（主要做為可用記憶區塊的宣告）

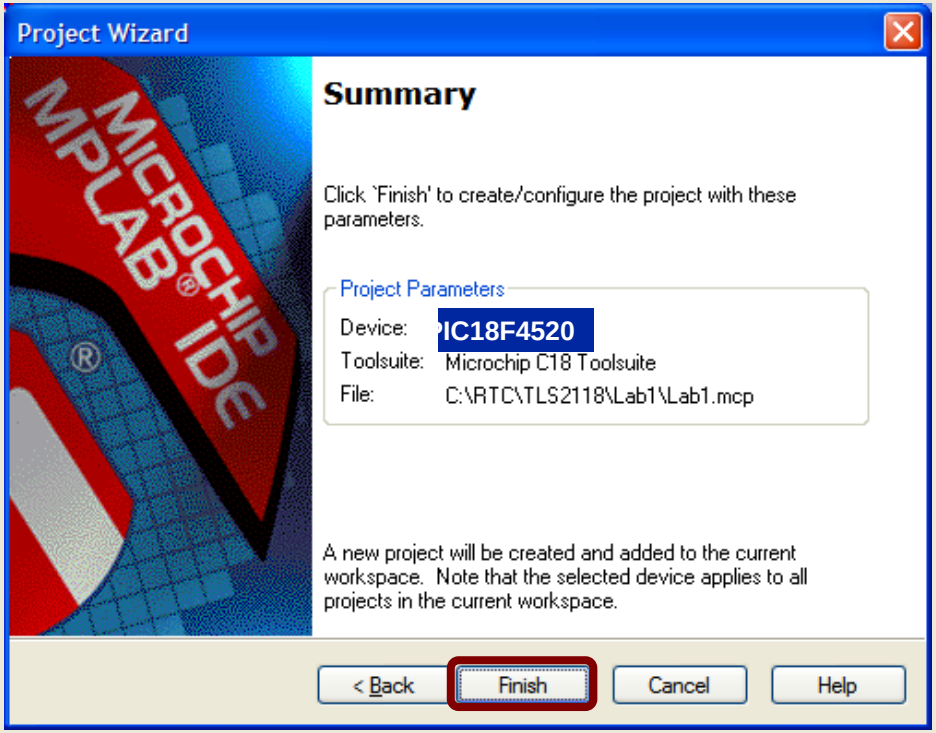
```
#IFDEF _DEBUGCODESTART
  CODEPAGE   NAME=page       START=0x0                 END=_CODEEND
  CODEPAGE   NAME=debug      START=_DEBUGCODESTART     END=_CEND       PROTECTED
#ELSE
  CODEPAGE   NAME=page       START=0x0                 END=0x7FFF
#FI
        CODEPAGE   NAME=idlocs          START=0x200000    END=0x200007     PROTECTED
        CODEPAGE   NAME=config          START=0x300000    END=0x30000D     PROTECTED
        CODEPAGE   NAME=devid           START=0x3FFFFE    END=0x3FFFFF     PROTECTED
        CODEPAGE   NAME=eedata          START=0xF00000    END=0xF000FF     PROTECTED
#IFDEF _EXTENDEDMODE
  DATABANK   NAME=gpre              START=0x0         END=0x5F
  ACCESSBANK NAME=accessram         START=0x60        END=0x7F
#ELSE
  ACCESSBANK NAME=accessram         START=0x0         END=0x7F
#FI
        DATABANK   NAME=gpr0            START=0x80        END=0xFF
        DATABANK   NAME=gpr1            START=0x100       END=0x1FF
```

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

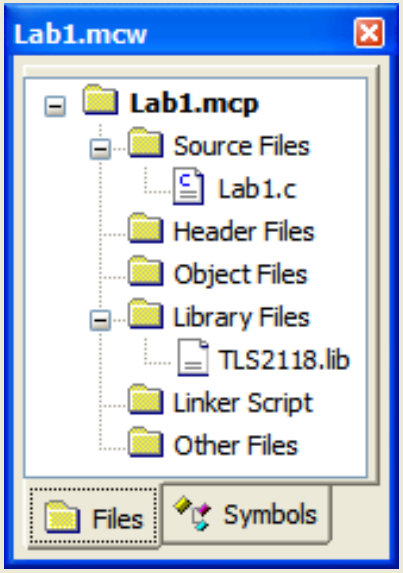
7 按下 **Finish**



這時將可以看到專案視窗
如無法看到專案視窗：

View ▸ Project

Lab 1 Project Tree

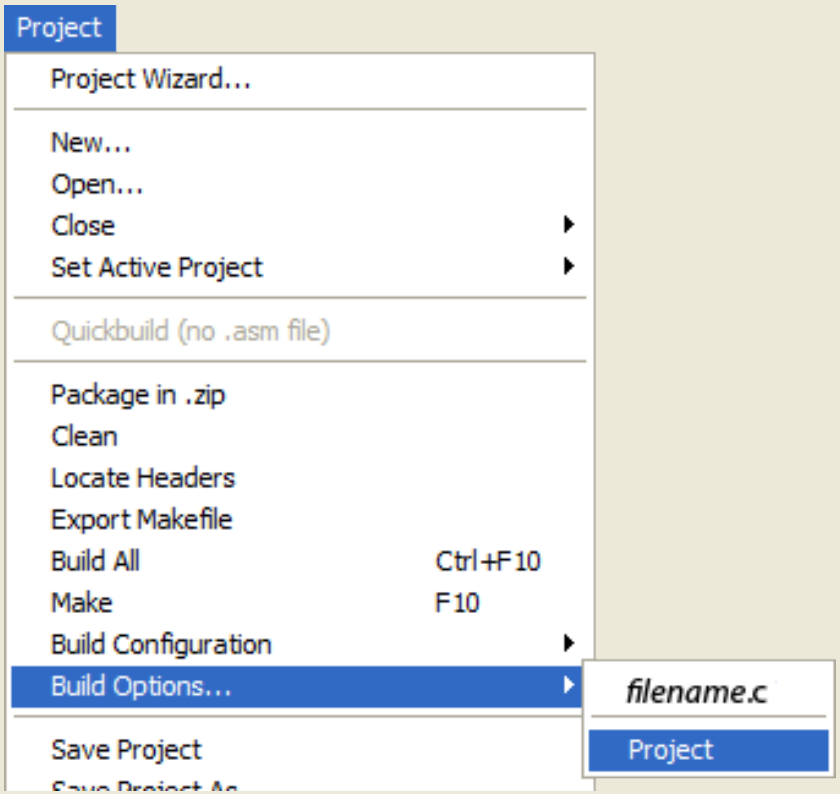


Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

8 設定專案內的編譯選項 **Project ▸ Build Options ▸ Project**



記住：
這個選項將會開啟該專案選項的對話視窗，你可以用它來變更編譯器的設定

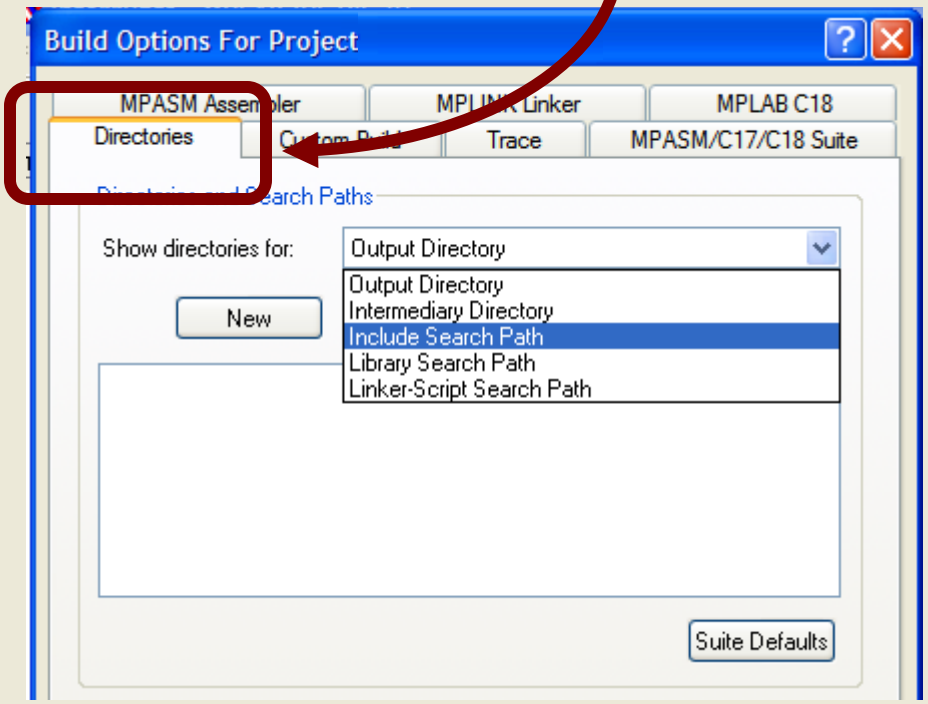
這個編譯選項在本課程中將常用到

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

9 選擇 Directories 對話項



現在要設定在專案裡所使用到的檔案路徑，這樣編譯才可以成功

所有的練習的預設路徑都放在:
C:\RTC\TLS2118 的目錄下

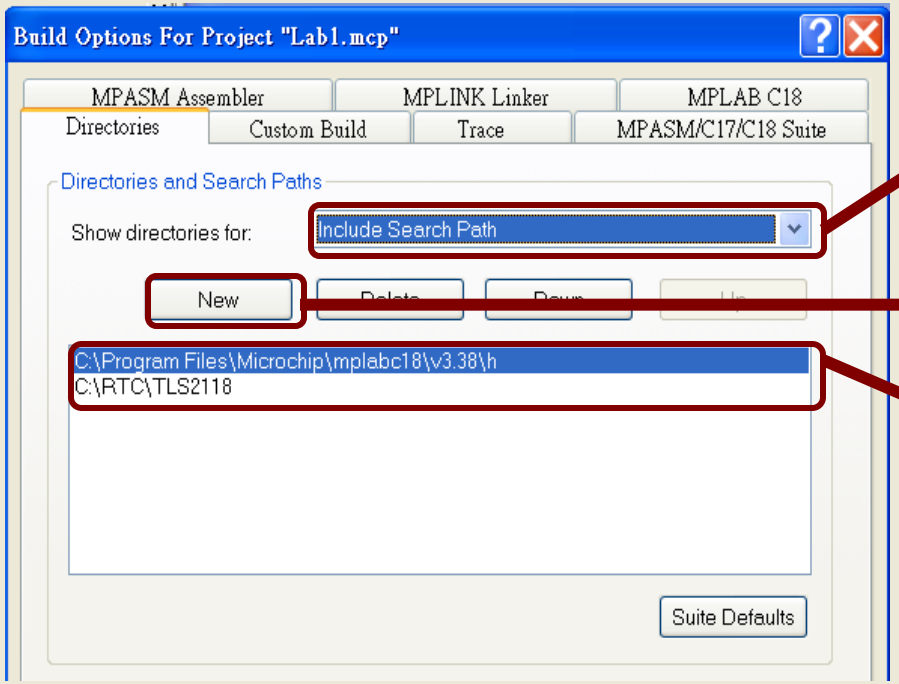
練習一的 **Project** 擺在:
C:\RTC\TLS2118\lab1\lab1.mcp

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

10 設定 Include 搜尋路徑



C18 v3.37 以後版本內定安裝路徑不再是 **c:\mcc18** 而是改到 **C:\Program Files** 下了

a. Select **Include Search Path** from the combo box

b. Click on **New**

c. 加入搜尋路徑： **..\Microchip\mplabc18\v3.38\h (Default 尋找路徑)**
C:\RTC\TLS2118 (加入新的搜尋)

TLS2118 的課程除了內定的標頭檔路徑外還必須指定自行加入的標頭檔路徑

這些 Header 路徑有何不同?

1. **#include <p18cxxx.h>**
2. **#include “TLS2118_LIB/P18F_LCD.h”**
3. **#include “P18F_LCD.H”**

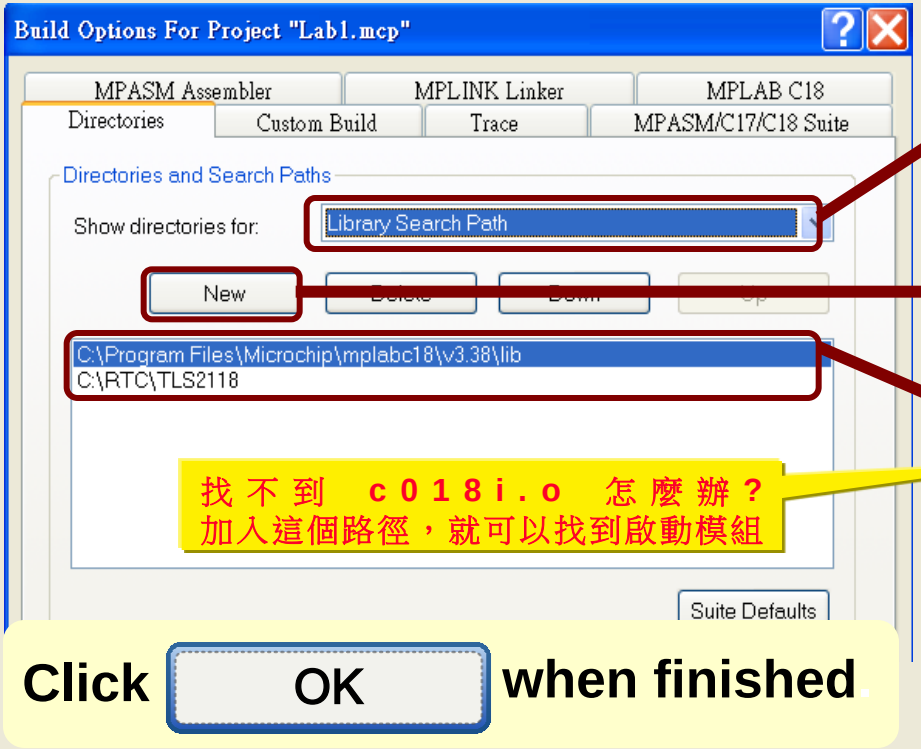
- 第一個使用 **< xxxx.h>** 的符號是表示使用：**..\Microchip\mplabc18\v3.38\h** 路徑下的 **Header File**
- 第二個使用 **“xxxxxx/xxxxx.h”** 引號是指目前的 **Project** 目錄，再下一層的指定路徑
- 第三是使用單純的**“xxxx.h”** 引號是指使用目前 **Project** 的目錄

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

11 設定 Library 搜尋路徑



Click OK when finished.

C18 v3.37 以後版本內定安裝路徑不再是 c:\mcc18\ 而是改到 C:\Program Files\ 下了

a Select Library Search Path from the combo box

b Click on New

c 加入路徑:
..\Microchip\mplabc18\v3.38\lib
C:\RTC\TLS2118\TLS2118_LIB

除了內定的資料庫路徑外，TLS2118_LIB 的路徑在本課程裡是需要特別指定的。

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

12 **針對Lab1.c 所加入最基礎所需的部分**
在 project 視窗中, Double click Lab1.c 可檢視下列的敘述

`#include` 所要用到的標頭檔是一你的應用程式而定.

- 元件定義標頭檔 (做為存取該元件的的定義名稱)
- 標準 C 及 Microchip 資料庫標頭檔 (if used)
- 自行定義的函數所要使用的標頭檔 (if used)

Lab1.c

```
13  #include <p18fcxxx.h>
14  #include "TLS2118_LIB/P18F_LCD.h"
15
```

所使用的相對路徑的設定是在 project options 對話視窗下設定

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

13 針對 Lab1.c 所加入最基礎所需的部分

在程式中已加入對configuration bits 設定的敘述

- 使用 `#pragma config`
- 參考各 help file 中對各 device 選項的列表，選項間可用逗號來區隔
- 為被指定的選項將會使用其預設值

hlpPIC18ConfigSet.chm

Lab1.c 如果使用 C18 Lite 版本建議將 extend instruction 功能關閉

```
17  #pragma config OSC=INTIO7, WDT=OFF, BOREN = ON, BORV = 1, LVP=OFF,
                PBADEN=OFF, XINST = OFF, MCLRE = ON
```

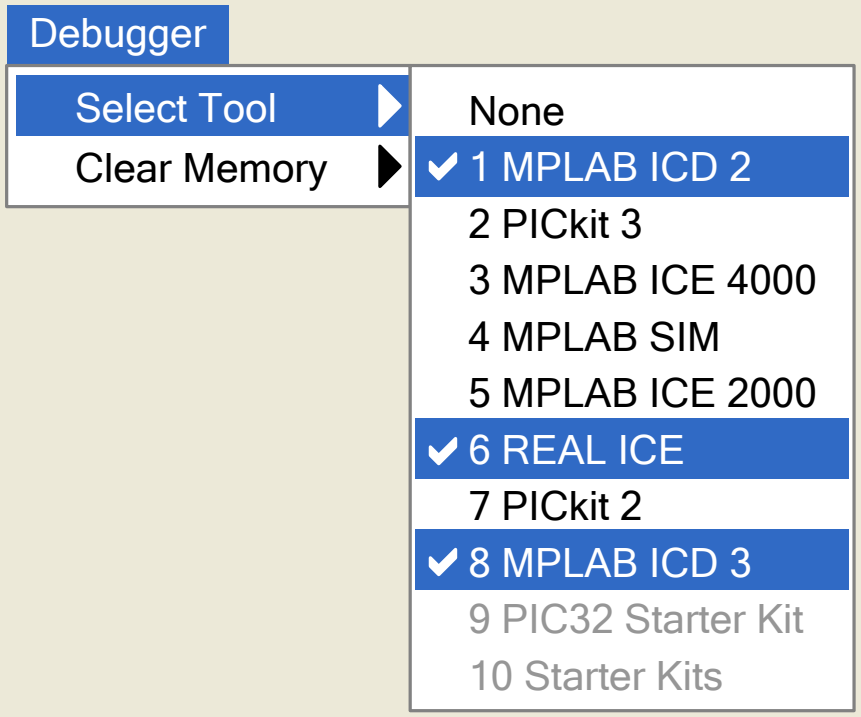
Lab Exercise 1

建立一個 MPLAB® C18 程式專案

步驟

14 開啟你的除錯工具選項：

Debugger ▸ Select Tool ▸



選則除錯工具：

- **MPLAB PICkit 3**
- **MPLAB ICD 3**
- **REAL ICE**
- **MPLAB SIM**

取決於何種工具連接到 **APP001** 的板子

DO NOT enable tool from the Programmer menu.

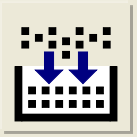
Lab Exercise 1

建立一個 MPLAB® C18 程式專案

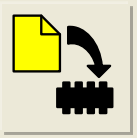
執行 Debug 的步驟

15 Select Debug mode.

16 Click Build All button.



17 If no errors reported, Click Program button.



18 When programming completes,click Reset button.

19 Click Run button.

20 Click Halt button.

Exercise 1 的結果

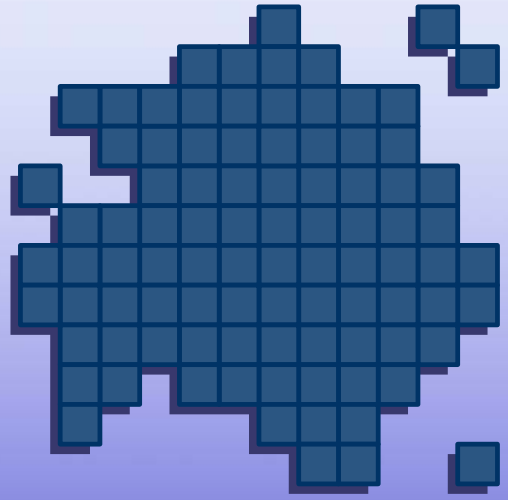
- 將可以看到 LCD 顯示幕出現
  “**Hello,TLS2118!**”

- **D1 LED 在閃爍著!**

MICROCHIP
Regional Training Centers

修改一下練習一

- 將 LCD 顯示的 “Hello,TLS2118!”
  - 改成您自己的英文名字
- D1 LED 在閃爍著!
  - 改成 D1~D8 以二進制方式加一顯示
- 目前的實驗室使用 HS Mode (16MHz)，請改成使用內部的 RC 震盪模式，並選擇 8MHz 的 Fosc.
  - LED 速度是否慢了一倍?



建立自己的 LIB 練習

建立一個 LCD 函數庫

(TLS2118.LIB)

Library Exercise

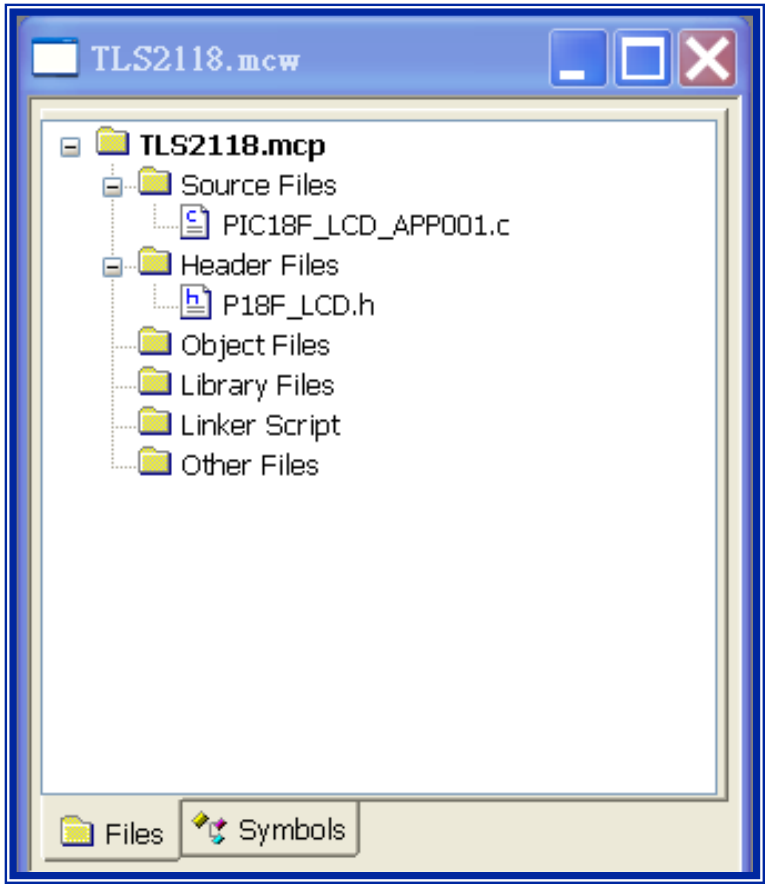
Lab1 功能 Review

使用自己的 **Library** 中的 **functions** 來操作 **LCD**

- 必須將所需要的 Library file 含入 project file 中
- 程式中必須含入欲使用該 Library 必須參考的 header file: P18F_LCD.h
- Lab1 使用的 function 有 OpenLCD( ) 、putrsLCD( )
- TLS2118.lib 中常用的 function 為
  - **void OpenLCD (void) ;**
  - **void putsLCD(char *) ;**
  - **void putrsLCD(const rom far char *) ;**
  - **void putcLCD(unsigned char) ;**
  - **void puthexLCD(unsigned char) ;**
  - **void LCD_Set_Cursor(unsigned char , unsigned char) ;**

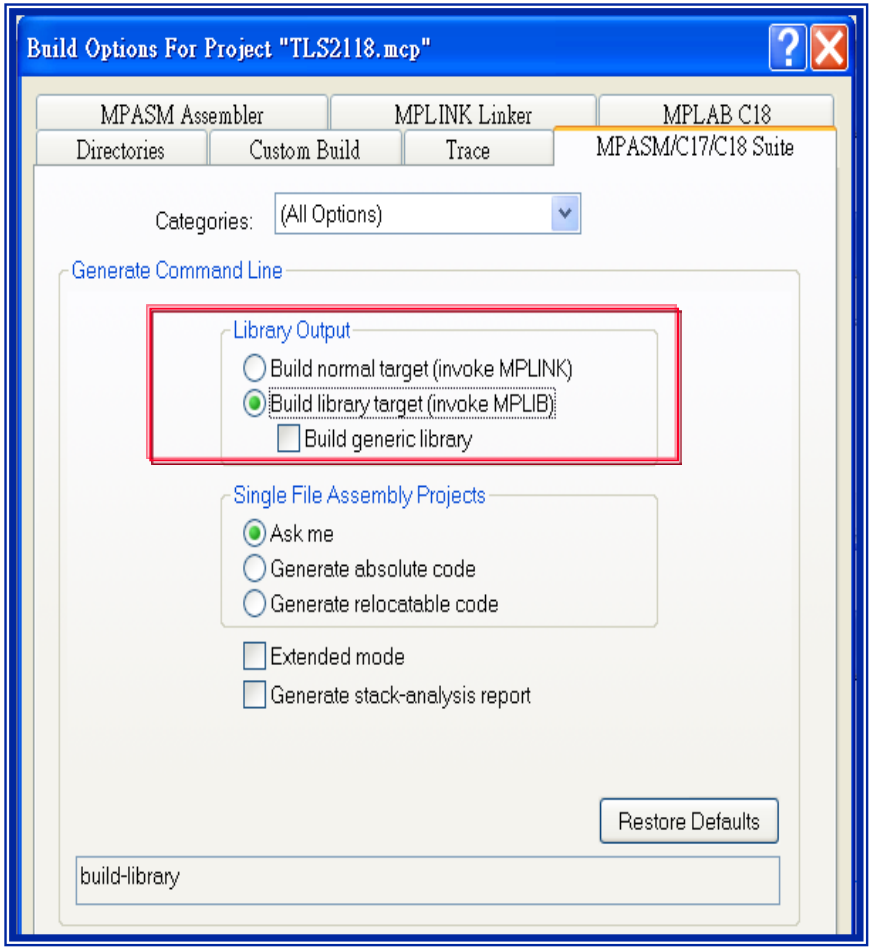
建立 LIB 步驟一

- 如同建立一般的 **Project** 一樣，將 **C** 程式加到 **Project** 裡。
- 每一個 **obj** 都可被 **Linker** 是為一個獨立模組，**Linker** 會將其抽出來使用，而不會將整個 **Library** 都拉進來佔空間。



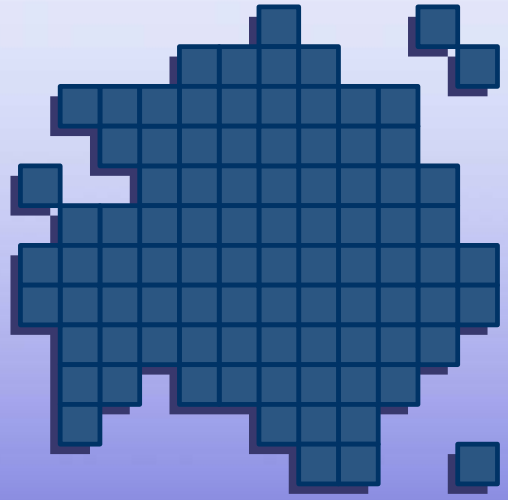
建立 LIB 步驟二

- 在 **Project → Build Options → Project** 下開啟左側的對話視窗
- 在 **Library Output** 選項下，選擇 **“ Build Library Target**
- 按下套用後離開
- 啟動 **Build All** 開始建立 **TLS2118.LIB**



Memory Model 討論

- 在 **LCD** 的函數裡:
  - putrsLCD(const rom far char *)
- **const rom** 有何意義?
- 選用何種 **Memory Model** 來編譯?
  - Large Mode 的修飾字 far 可擴展程式的視野到 2MB
  - Small Mode (Default) 64KB



PIC18 Family 內建的 Special Features

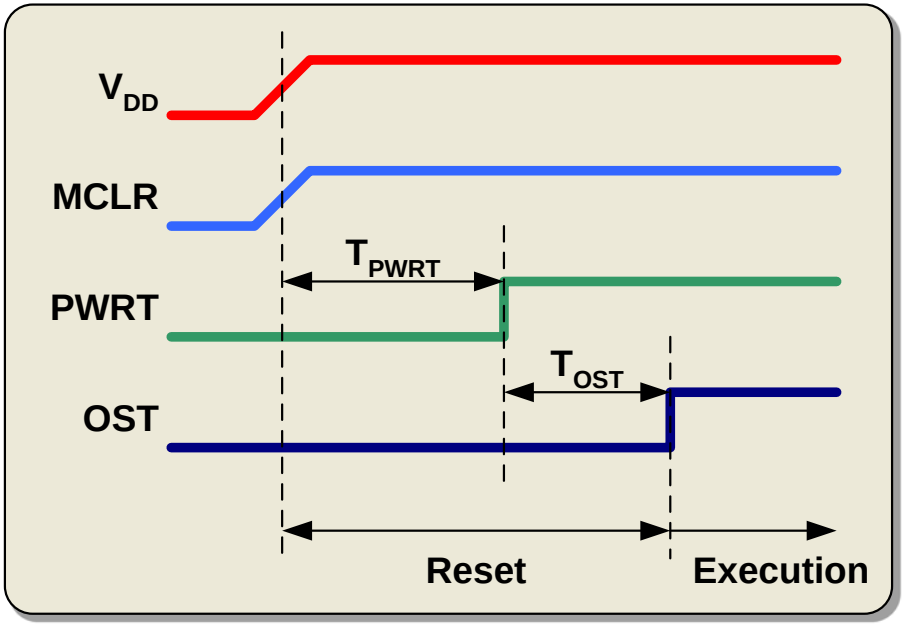
PIC18 震盪器的選擇

XT	Standard frequency crystal oscillator	100kHz - 4MHz
HS	High frequency crystal oscillator	DC - 40MHz
HS+PLL	High frequency crystal with 4x PLL	4MHz - 10MHz
LP	Low frequency crystal oscillator	5kHz - 200kHz
RC	External RC oscillator	DC - 4MHz
RCIO	External RC oscillator, OSC2=RA6	DC - 4MHz
INTRC	Internal RC oscillator	Various
EC	External Clock, OSC2=f_{osc}/4	DC - 40MHz
ECIO	External Clock, OSC2=RA6	DC - 40MHz

- **Selectable clock options provide greater flexibility for the designer:**
  - LP Oscillator designed to draw least amount of current
  - RC or INTRC provide ultra low cost oscillator solution
  - XT optimized for most commonly used oscillator frequencies
  - HS optimized to drive high frequency crystals or resonators
- **表上的工作頻率範圍僅做為設計參考準則**

POR, OST, PWRT

- **POR: Power On Reset**
  - With MCLR tied to V_{DD}, a reset pulse is generated when V_{DD} rise is detected
- **PWRT: Power Up Timer**
  - Device is held in reset for 72ms (nominal) to allow V_{DD} to rise to an acceptable level (after POR only)
- **OST: Oscillator Start-up Timer**
  - Holds device in reset for 1024 cycles to allow crystal or resonator to stabilize in frequency and amplitude; not active in RC modes; used only after POR or Wake Up from SLEEP



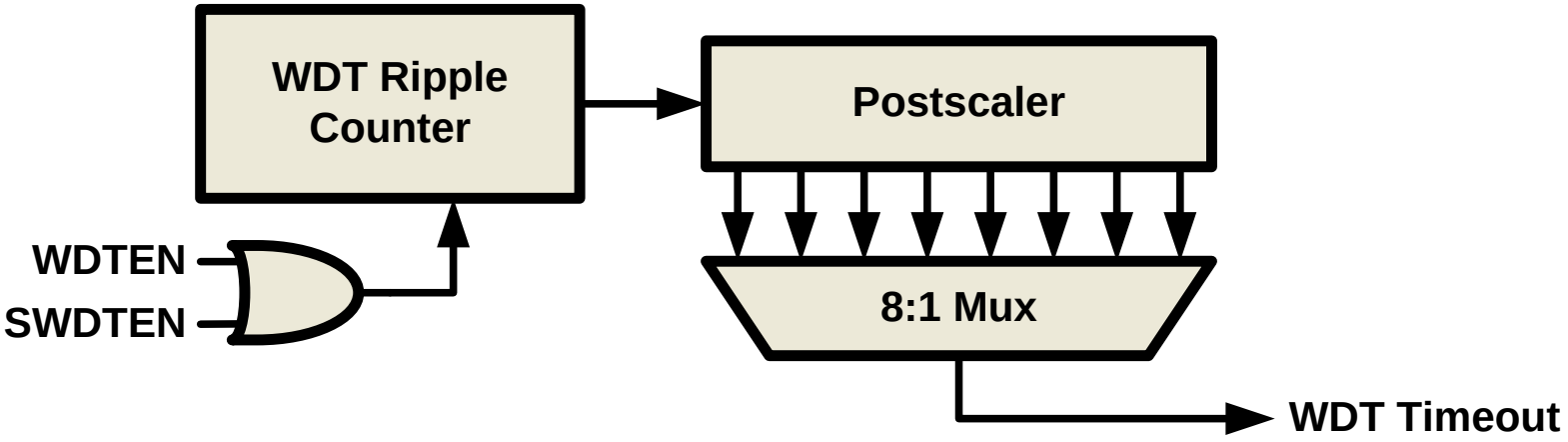
Sleep Mode

- **The processor can be put into a power-down mode by executing the SLEEP instruction**
  - System oscillator is stopped
  - Processor status is maintained (static design)
  - Watchdog timer continues to run, if enabled
  - Minimal supply current is drawn - mostly due to leakage (0.1 - 2.0μA typical) , XLP device is much lower than std. device

可以使用於在 Sleep Mode 喚醒 CPU 的周邊來源	
MCLR	Master Clear Pin Asserted (pulled low)
WDT	Watchdog Timer Timeout
INT	INT Pin Interrupt
TMR1	Timer 1 Interrupt (or also TMR3 on PIC18)
ADC	A/D Conversion Complete Interrupt
CMP	Comparator Output Change Interrupt
CCP	Input Capture Event
PORTB	PORTB Interrupt on Change
SSP	Synchronous Serial Port (I^2C Mode) Start / Stop Bit Detect Interrupt
PSP	Parallel Slave Port Read or Write

Watchdog Timer

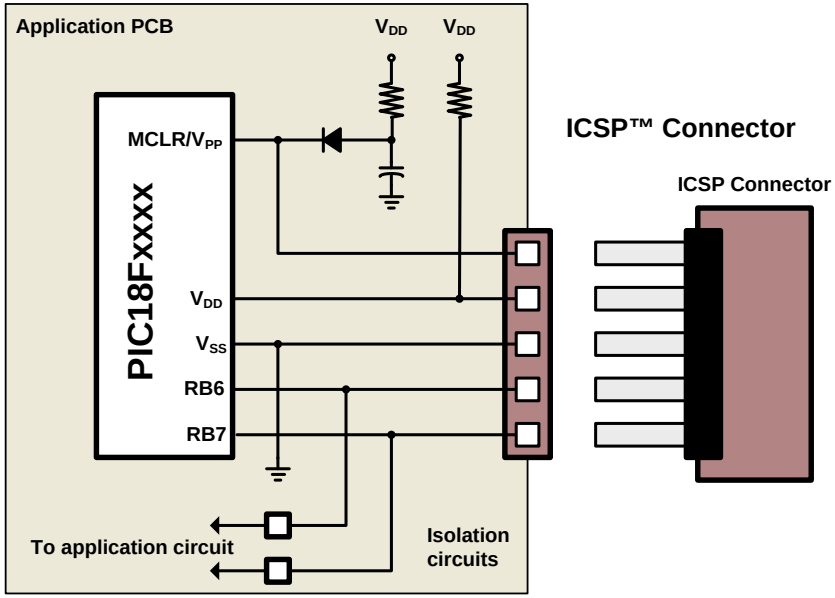
- Helps recover from software malfunction
- Uses its own free-running on-chip RC oscillator
- WDT is cleared by CLRWDT instruction
- Enabled WDT (WDTEN) cannot be disabled by software
- WDT overflow resets the chip
- Programmable timeout period: 18ms to 3.0s typical
- Operates in SLEEP; on time out, wakes up CPU



In-Circuit Serial Programming™

- **除了電源與 Vpp(MCLR) 接腳外，只需要另外兩個接腳即可進行對 PIC18 device 的燒錄與除錯**
  - ICSP Data
  - ICSP Clock
- **也可以很方便地進行對下列的記憶體做 In-System Programming**
  - Calibration Data
  - Serialization Data
- **MPLAB® ICD3，PICKit 3，Real ICE 等燒錄/除錯器都可以使用此種方式為 PIC18 device 燒錄程式並進行除錯**

Pin	Function
V_{PP}	Programming Voltage = 13V
V_{DD}	Supply Voltage
V_{SS}	Ground
RB6	Clock Input
RB7	Data I/O & Command Input



BOR – Brown Out Reset

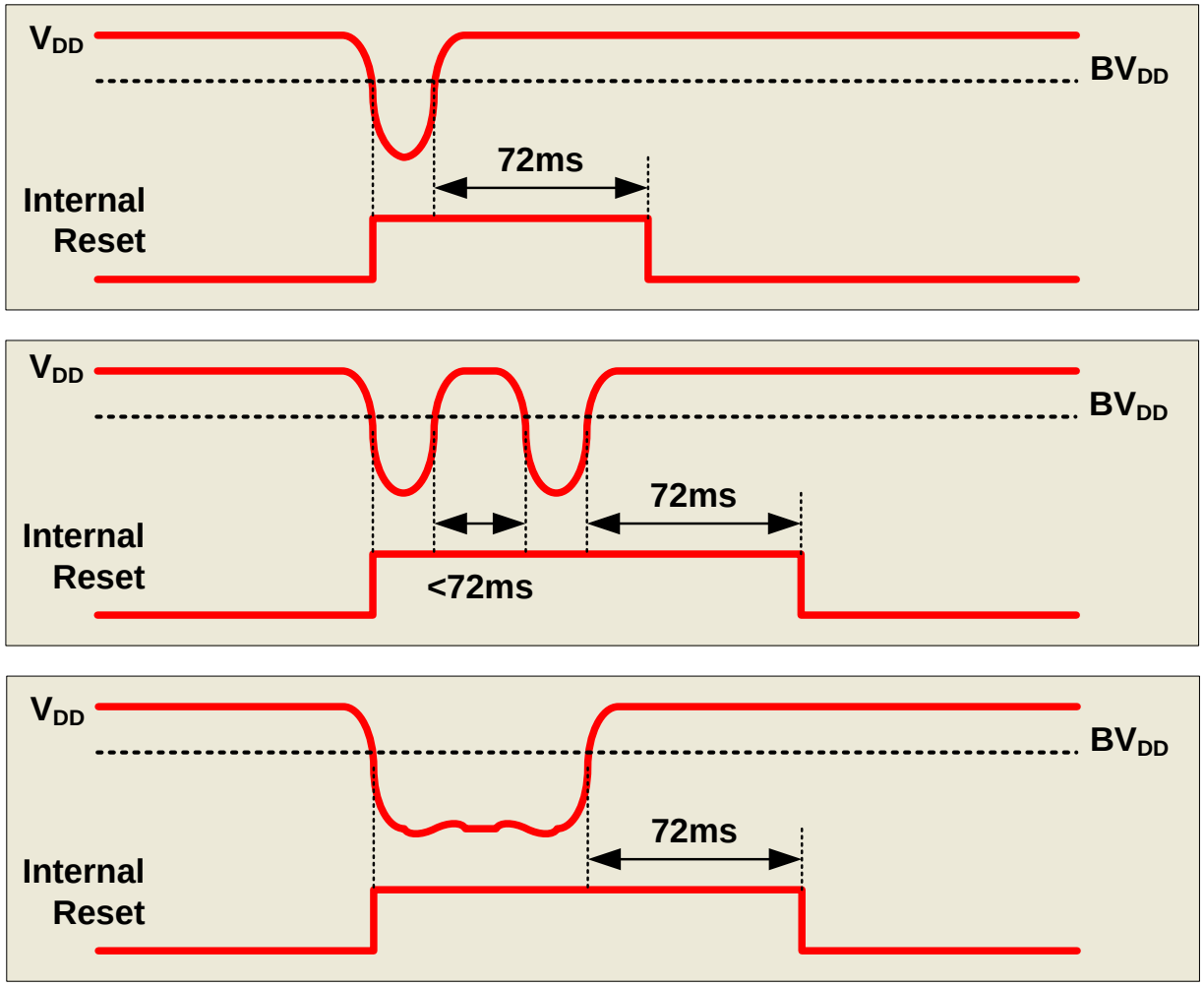
- When voltage drops below a particular threshold, the device is held in reset
- Prevents erratic or unexpected operation
- Eliminates need for external BOR circuitry
- 注意: BOR 的消耗電流在 Sleep Mode 下
- 一般使用建議啟用 BOR

PBOR
Programmable Brown Out Reset

- **Configuration 的選項 (在燒錄 IC 時一併被設定)**
  - 無法使用軟體來 enabled / disabled
- **有四個可選擇的 BV_{DD} 觸發位準選項 : (不同 device 間可能有部同的規格)**
  - 2.5V
  - 2.7V
  - 4.2V
  - 4.5V
- **如果 device 內設的臨界值未能符合需求, 可以使用外部的 supervisor (MCP1xx, MCP8xx/TCM8xx, or TC12xx)**

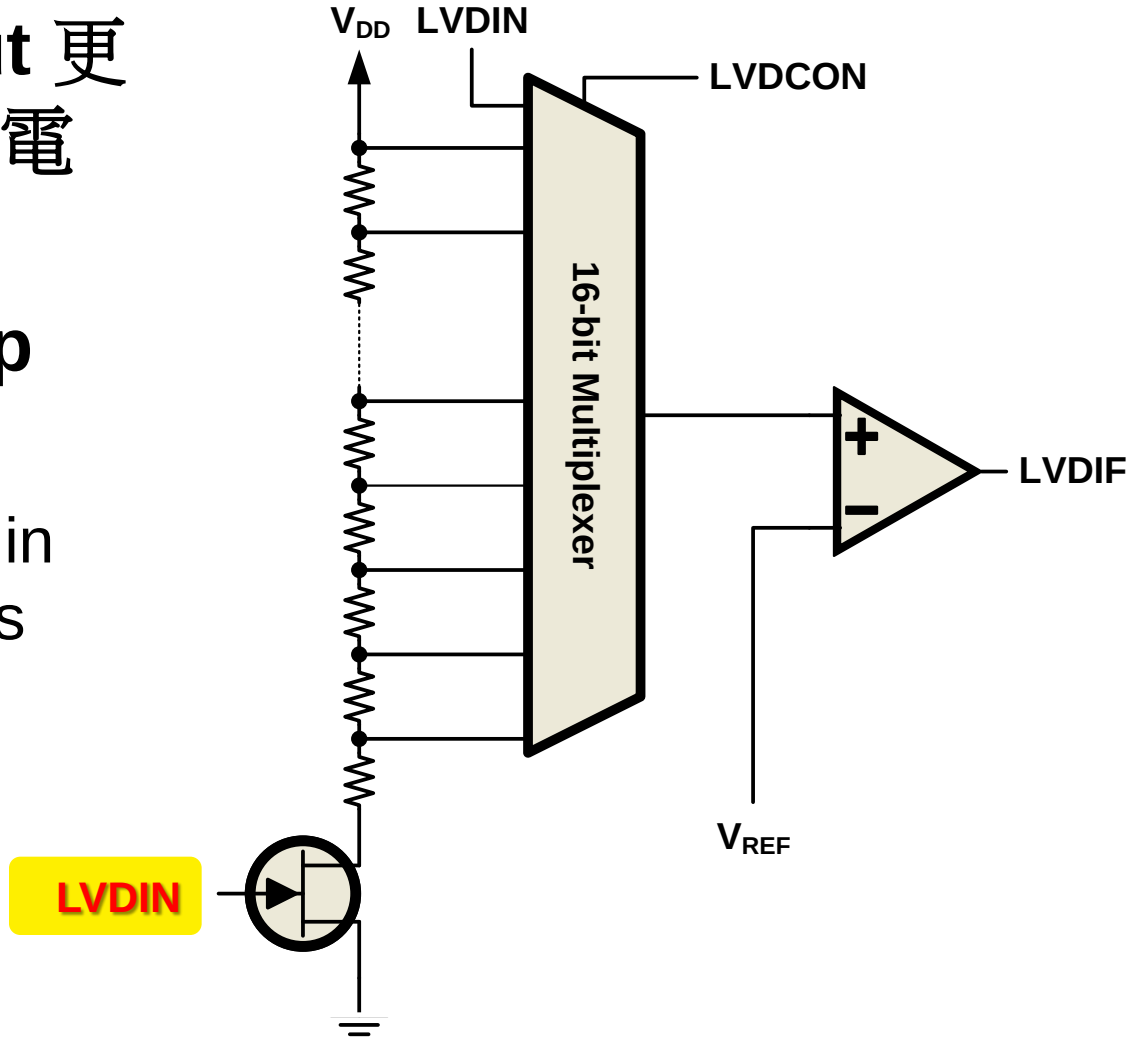
PBOR – Programmable Brown Out Reset

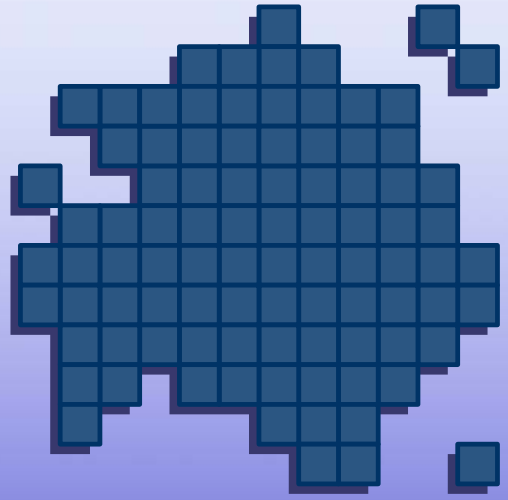
- Holds PICmicro® MCU in reset until ~72ms after V_{DD} rises back above threshold



PLVD – Programmable Low Voltage Detect

- 可以比 **brown out** 更早對 **device** 做出電源異常警示
- **16 selectable trip points:**
  - 1.8V up to 4.5V in 0.1 to 0.2V steps
  - External analog input
- **Internal V_{REF}**





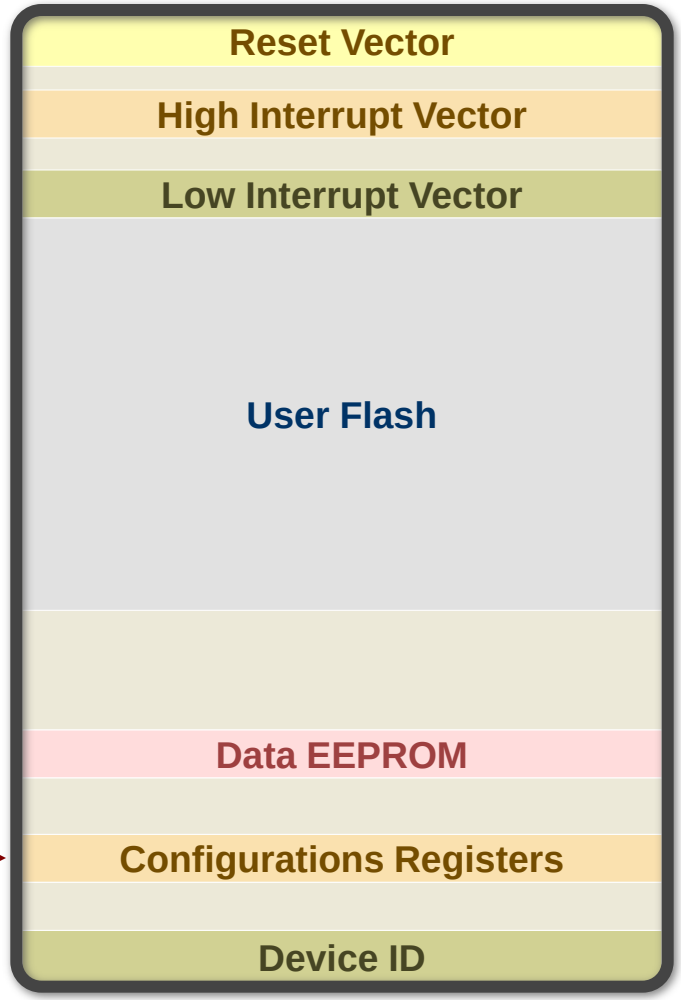
How to Set PIC® Configuration Bits

Using the `config` pragmas

What are Configuration Bits?

- **Used to setup device features:**
  - Code Protect
  - Watchdog Timer
  - Oscillator Options
  - Debug Options
  - More…

CONFIG registers located in program memory space, outside range of executable code space (starts @ 0x300001)



Configuration Bits

PIC18F4520 Registers (4 of 11)

CONFIG1H Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IESO	FCMEN	—	—	FOSC3	FOSC2	FOSC1	FOSC0
bit 7							bit 0

CONFIG2L Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	BORV1	BORV0	BOREN1	BOREN0	$\overline{\text{PWRTEN}}$
bit 7							bit 0

CONFIG2H Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN
bit 7							bit 0

CONFIG3H Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
MCLRE	—	—	—	—	LPT1OSC	PBADEN	CCP2MX
bit 7							bit 0

如何在程式中直接設定 Configuration Bits

Syntax

```
#pragma config setting-list
```

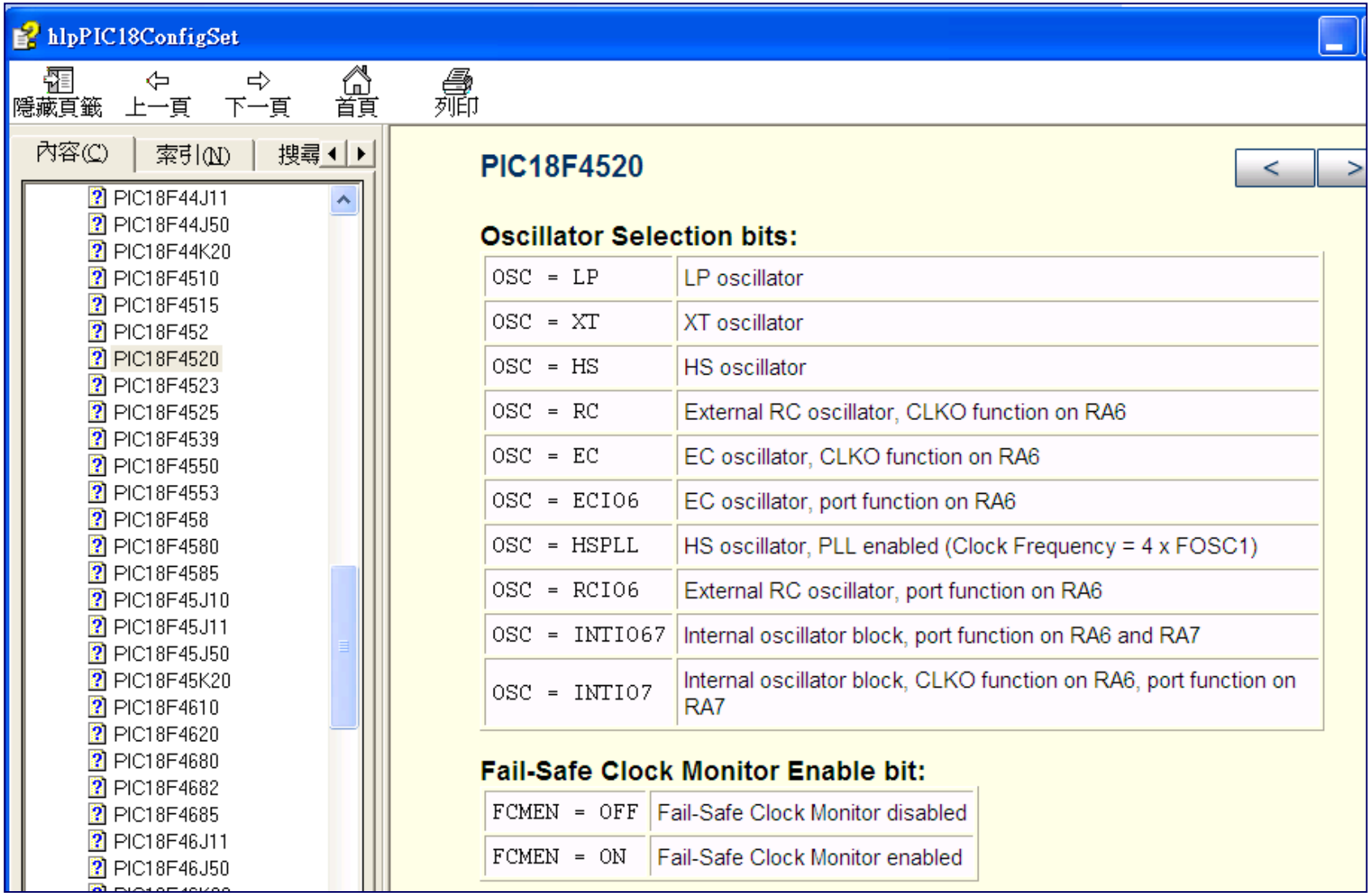
- setting-list is a comma separated list of settings as defined in hlpPIC18ConfigSet.chm

Values for the setting list are defined in the PIC18 Config Set help file.

Example

```
#pragma config OSC = INTIO67, WDT = OFF, BOREN = OFF
```

Configuration Bits 設定項



MICROCHIP
Regional Training Centers

How to Set Configuration Bits

PIC18F4520 擁有的選項 (Part 1)

Feature Symbol	Description	Setting Options
OSC	Oscillator Selection	LP,XT,HS,RC,EC,ECIO6, HSPLL,RCIO6,INTIO67,INTIO7
FCMEN	Fail-Safe Clock Monitor Enable	ON,OFF
IESO	Internal/External Oscillator Switchover	ON,OFF
PWRT	Power Up Timer Enable	ON,OFF
BOREN	Brown Out Reset Enable	ON,OFF,NOSLP,SBORDIS
BORV	Brown Out Reset Voltage	0,1,2,3
WDT	Watchdog Timer Enable	ON,OFF
WDTPS	Watchdog Timer Postscale Select	1,2,4,8,16,32,64,128,256,512, 1024,2048,4096,8192,16384,32768
MCLRE	MCLR Pin Enable	ON,OFF
LPT1OSC	Low Power TMR1 Oscillator Enable	ON,OFF
PBADEN	PORTB A/D Input Enable	ON,OFF
CCP2MX	CCP2 Mux	PORTBE,PORTC
STVREN	Stack Overflow/Underflow Reset Enable	ON,OFF
LVP	Single Supply ICSP Enable	ON,OFF
XINST	Extended Instruction Set Enable	ON,OFF
DEBUG	Background Debugger Enable	ON,OFF
CP*n*	Code Protection Enable Block *n* (*n* = 0-3)	ON,OFF
CPB	Boot Block Code Protection Enable	ON,OFF
WRT*n*	Write Protection Enable Block *n* (*n* = 0-3)	ON,OFF
WRTB	Boot Block Write Protection Enable	ON,OFF

How to Set Configuration Bits

PIC18F4520擁有的選項 (Part 2)

Feature Symbol	Description	Setting Options
WRTC	Configuration Register Write Protection	ON,OFF
WRTD	Data EEPROM Write Protection	ON,OFF
EBTR*n*	Table Read Protection Block *n* (*n* = 0-3)	ON,OFF
EBTRB	Boot Block Table Read Protection Enable	ON,OFF

- Available configuration options are device specific
- **hlpPIC18ConfigSet.chm** has an entry for every PIC18
- See device specific data sheet for details on configuration options and their settings

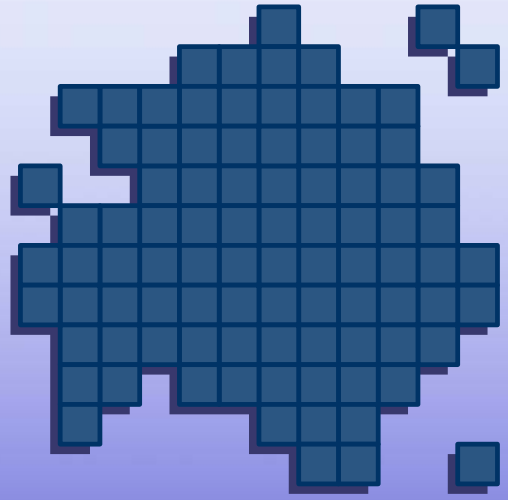
TLS2118 實驗
Configuration Bits 的設定

- 底下的程式使用在 TLS2118 所有的練習裡，用來設定 Configuration Bits

```
#pragma config OSC=INTIO7, WDT=OFF, BOREN = ON,
    BORV = 1, LVP=OFF, PBADEN=OFF, XINST = OFF,
    MCLRE = ON
```

注意:

- 在除錯模式下有些限制: WDT 必須是關閉的。LVP 也是要關閉。MCLRE 必須設 ON 以啟用 MCLR Pin 的外部 Reset 功能。
- 還有如果 C18 是使用 Lite 版本，Extended Instruction Sets 功能也要關閉。



How to Read and Write Registers

Word and Bit Access

How to Read and Write Registers

Full 8-bit Word

Syntax

REGNAME

- Use the name of the register as you would an ordinary `char` type variable
- Variables defined in device specific header file with register name as shown in datasheet

Example

```
PORTB = 0x31;                    // Write 0x31 to PORTB
AtoD_Result = ADRESH;            // Read A/D Result (High)
TXREG = 'a';                     // Send 'a' out UART
if (RXREG == 'x') { … }          // If received char is 'x'
while (RXREG) { … }              // While char is not '\0'
```

Register Variable Declaration

Example for PIC18F4520

PORTA Variable Declaration from p18f4520.h Header File

```
extern volatile near unsigned char PORTA;
```

- `extern`: PORTA is actually defined in a device specific library file
- `volatile`: This variable may be altered by something other than the code (i.e. the hardware or an interrupt routine)
- `near`: Denotes that a variable located in data memory lives in ***Access RAM*** and that no bank switching instructions are required to access it

安排變數的位置（RAM Location)

- 在**Access Bank**內的變數執行速度較快
- 利用前置處理指令 **“#pragma {udata/idata}”**來指定變數在**RAM**的特定位置

  #pragma udata [data-qualifier] [section-name [=address]]

  #pragma idata [data-qualifier] [section-name [=address]]

- 利用前置處理指令 **“#pragma udata/idata”**來結束指定節區位置的作業

  #pragma udata/idata

#pragma udata

- #pragma udata [data-qualifier] [section-name [= addr]]
  - #pragma udata [xxx]會將以下所宣告的變數作特定的位置安排，直到遇到指令“#pragma udata”才進行常態安排
  - udata :設定無初始值的變數(idata有初始值的變數)

Option

  - [data-qualifier] : 變數要擺在那一區域
    - “access” ==> 安排在 ACCESS Bank 的 RAM (0x00-0x7F)中
    - “ 空白 ” ==> 安排在非 ACCESS Bank (GPR)

Option

  - [section-name [= addr]] :變數放置的位址
    - 指定section-name: 與 Linker的連結描述檔內的section-name相同，則會依連結描述檔內的指定進行安排
    - 不指定section-name: 即獨立的名字，並不與 Linker的連結描述檔內的section-name相同，此時變數會放在unprotected區
    - = addr : 可以強制指定節區位址

#pragma udata 宣告（範例）

```
#pragma udata access AccessSection
near unsigned char Temp_Code[4];
near unsigned char Rec_Data;
near unsigned char PWM_Duty;
near unsigned char On_Flag;

#pragma udata abc=0x100
unsigned char j;
unsigned char i;
unsigned char e;
unsigned char f;

#pragma udata test
unsigned char EE_Write_Data;
unsigned char EE_Addr;
unsigned char Send_UR;
unsigned char Err;

#pragma udata
```

宣告以下之變數放在Access Bank中，由Linker自行安排位址

宣告以下之變數放在GPR中位址為0x100的地方

宣告以下之變數放在GPR中，由Linker自行安排位址，又 section name 有特別指定故會被安排在 Bank 2 的位址

SECTION NAME=test RAM=gpr2

How to Read and Write Registers

Individual Bits and Bit Fields

Syntax

```
REGNAMEbits.BITNAME
```

- Use the name of the register with the word 'bits' in lower case attached to it
- Use the name of the bit or bit field as specified in the data sheet

Example

```
LATBbits.LATB5 = 1;                 // Set bit 5 of PORTB
Flag = PORTBbits.RB5;               // Read bit 5 of PORTB
while (!ADCONbits.DONE) { … }       // While A/D converting
ADCON0bits.ADON = 1;                // Enable A/D
```

Bit Field Variable Declaration

Example for PIC18F4520

TRISEbits Variable Declaration from p18f4520.h Header File

```
extern volatile near unsigned char TRISE;
extern volatile near union {
  struct {
    unsigned TRISE0:1;
    unsigned TRISE1:1;
    unsigned TRISE2:1;
  };
  struct {
    unsigned :4;
    unsigned PSPMODE:1;
    unsigned IBOV:1;
    unsigned OBF:1;
    unsigned IBF:1;
  };
} TRISEbits;
```

Bit Field Structure Definitions

Primary Bit Names

Secondary Bit Names

Bit Field Variable Declaration

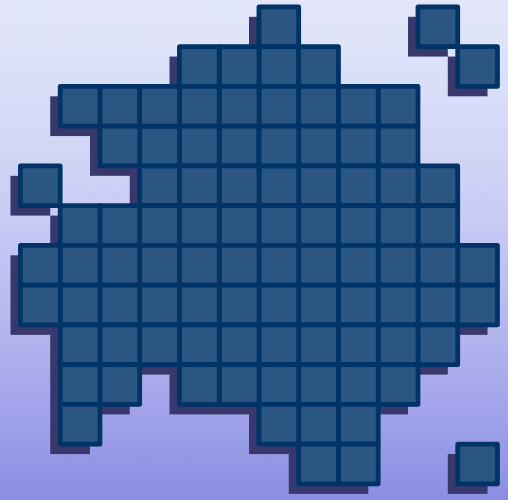
Register and Bit Field Variables

Definition in Processor Specific Library

- Both *REGNAME* and *REGNAME*bits defined in a processor specific library
- Both are allocated at the same address

Excerpt from p18f4520.asm (p18f4520.lib) Library File

```
SFR_UNBANKED0       UDATA_ACS H'F80'

PORTA
PORTAbits           RES 1     ; 0xF80
PORTB
PORTBbits           RES 1     ; 0xF81
PORTC
PORTCbits           RES 1     ; 0xF82
      :
```



練　習　二

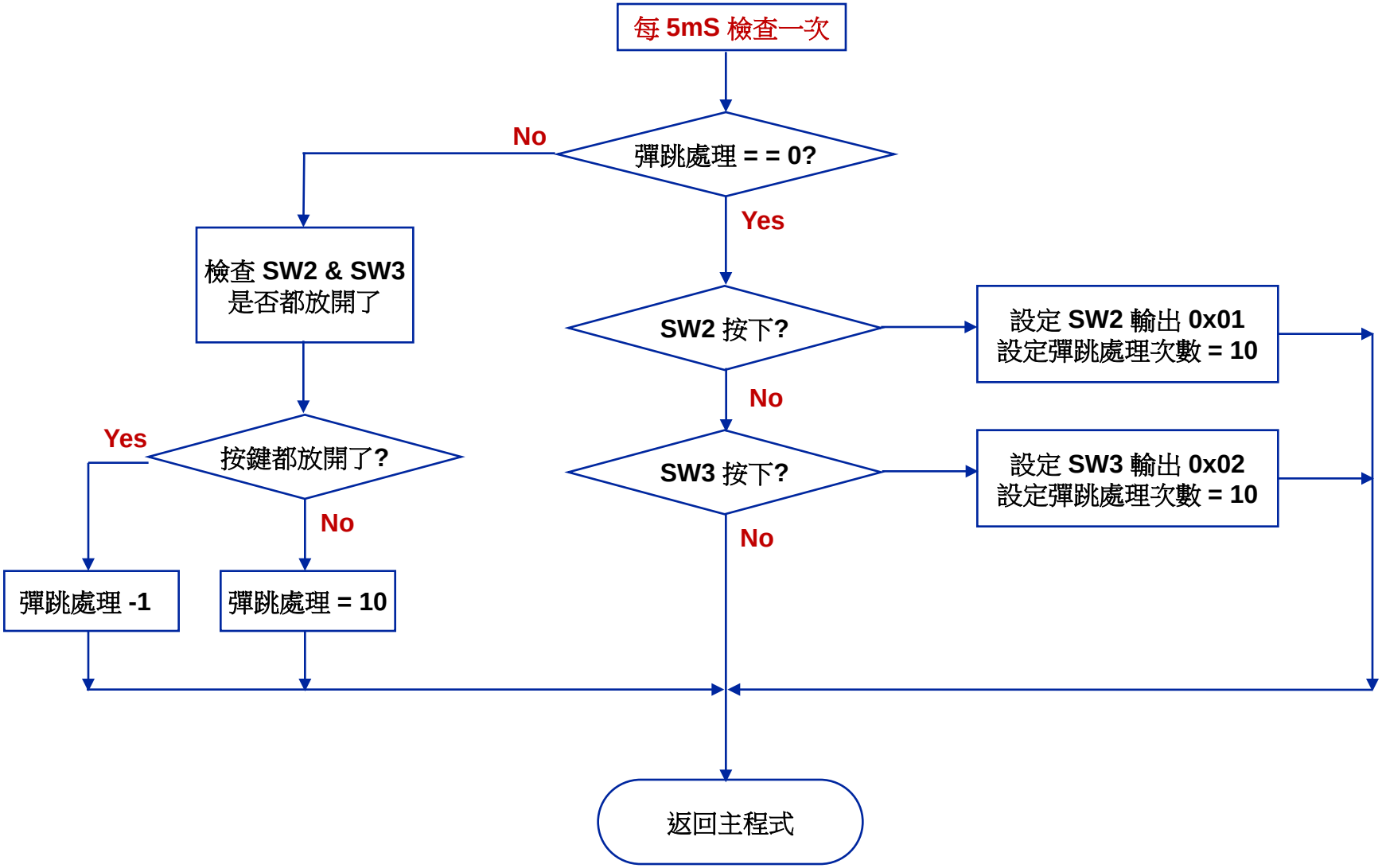
I/O Port 的規劃與運用

MPLAB SIM – 軟體模擬器的應用

LCD 副程式的應用

C18 標準函式的應用

使用軟體延遲方式處理按鍵彈跳



Lab Exercise 2

I/O 與 Library 函式的活用

Lab Exercise 2 練習目標

- **學會 I/O 的規劃**
  - 參考 APP001 的電路圖, 將按鍵與LED 的控制腳位找出來並予以正確規劃 (注意: LCD 與 LED是共用腳位的設計)
  - 使用 C18 的共用型位元結構, 練習輸入腳位的讀取與輸出腳位的控制
- **練習使用 MPLAB SIM 來觀察 delay( ) 程式所花費的時間**
- **運用 Library – TLS2118.lib 中提供的 LCD function libraries 來做游標的設定及顯示程式中的變數值.**
- **運用 MPLAB C18 提供的 standard library**
  - 知道 itoa(int value, char *string) 這標準函數的功能嗎?
  - 將程式中的變數 “SW2_Count” 及 “SW3_Count” 轉換為字串格式後使用 putsLCD( ) 印出至 LCD
- **使用軟體延遲方式處理按鍵輸入及消除彈跳現象**

Lab 2 動手做練習

Lab 2 動手做練習

- **LAB2.c 原始檔案放在: ..\2014 Summer Techer Bootcamp\Section 1\Labs\Lab Exercise\Lab2** 的目錄下
- 本練習會使用到的檔案有
  - Lab.c
  - PIC18F_LCD_APP001.c , PIC18F_LCD.h
- 請自行建立專案後開始修改 **Lab.c** 的程式

Lab2 Debounce
SW3=22 ;SW2=15

Lab2 程式執行時 LCD 的顯示

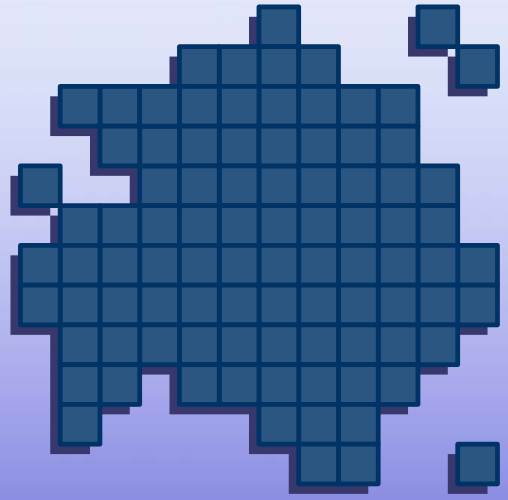
Lab2 要增寫的程式

- 尋找程式裡有 “***//???***” 的字，這表示這裡的程式需要修改。

1. 修改 **19 & 20** 行，定義 **SW2 & SW3**
2. 修改 **MS_Delay( )** 的延遲時間為 **5mS**，並使用 **MPLAB SIM+ Stopwatch** 來驗證 **(Fosc=8MHz)**
3. 參考 **Page 96** 頁的流程圖撰寫: 軟體延遲方式處理按鍵彈跳 的程式
4. 練習控制 **LCD** 顯示的位置。利用 **LCD_Set_Course(n,n)** 的函數設定顯示位置。

Lab2 按鍵消除彈跳問題

- 使用軟體延遲方式做去彈跳處理
  - 消耗過多 CPU 效能
  - 無法在背景執行按鍵掃描
- 一般都會使用 Timer 中斷來處理按鍵的掃描與彈跳
  - 即時反應
  - 背景執行，不影響主程式的執行
  - 能確實處彈跳跳問題



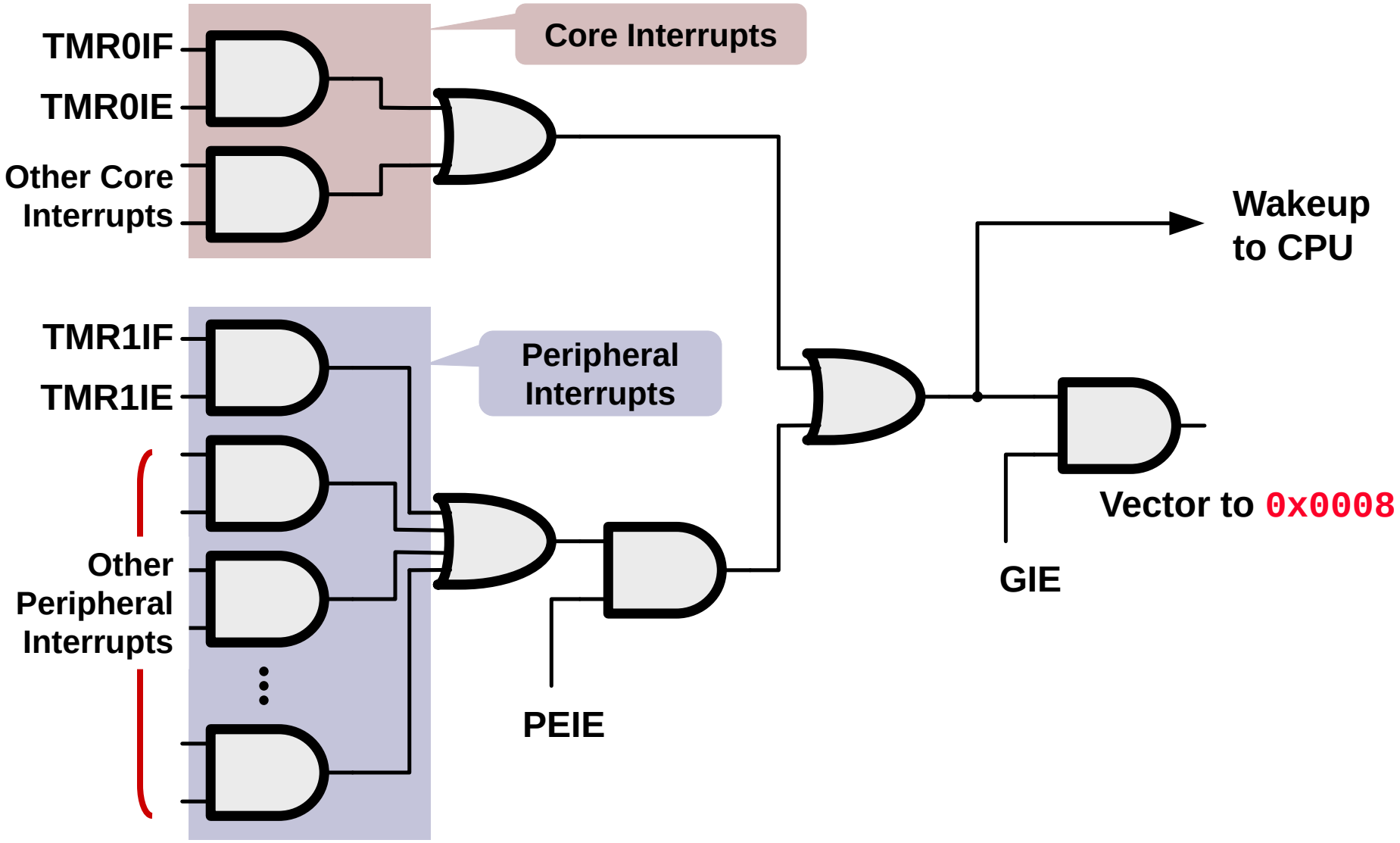
PIC18 Family 內建的基礎周邊說明
Interrupts

MICROCHIP
Regional Training Centers

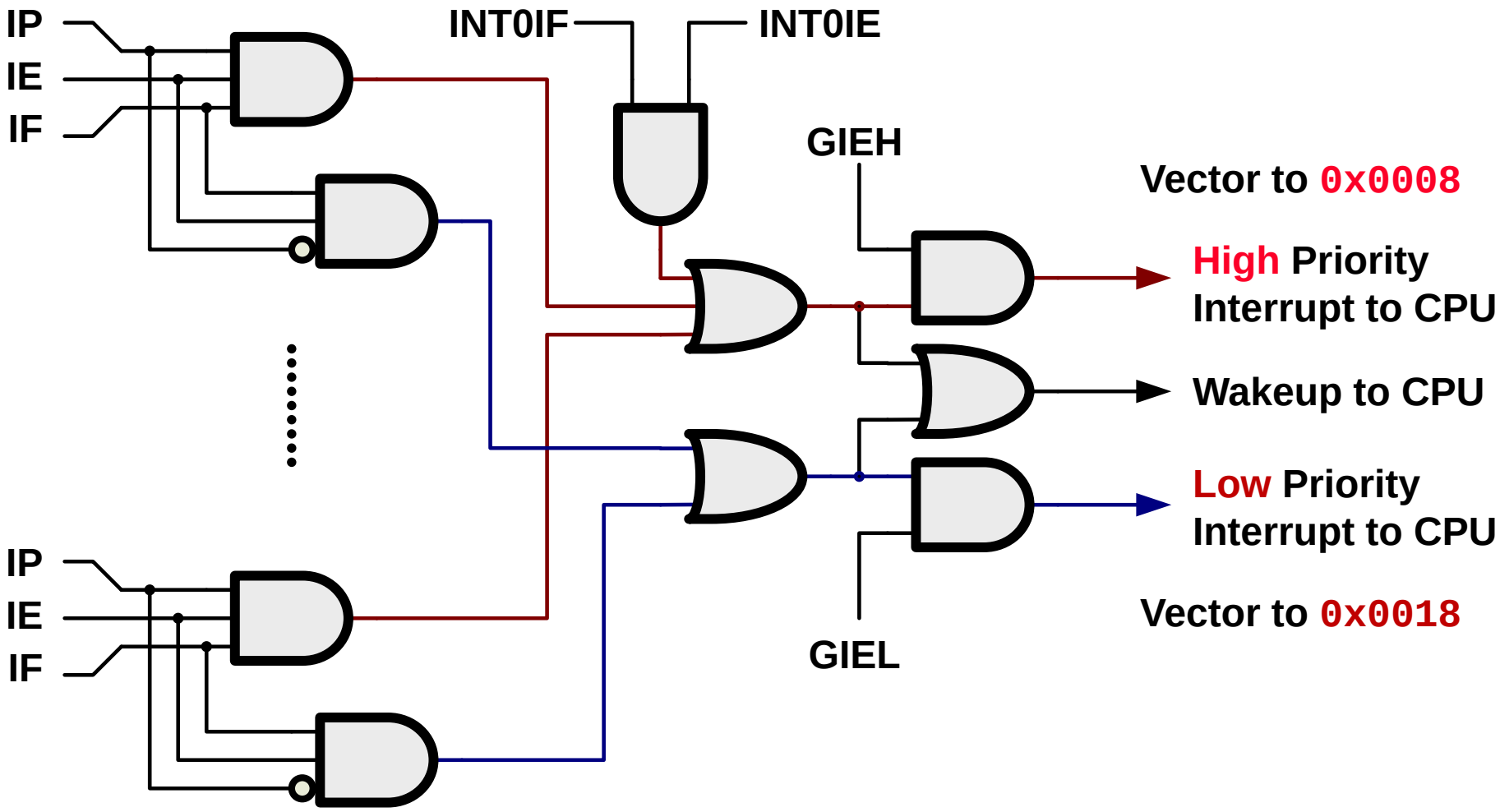
18F4520 中斷處理

- **18F4520有兩個中斷進入執行位址**
  - 高優先權==>中斷進入位址0x0008
  - 低優先權==>中斷進入位址0x0018
  - 每個中斷源均可選擇中斷優先權（二選一）
  - 每個中斷源均有獨立的中斷旗標(Flag)
  - 中斷旗標的清除==>自行用軟體清除
  - 每個中斷源均可獨自 Enable 或 Disable
- **當然 PIC18系列也可設定與PIC16Fxxx系列的中斷相容（關掉優先權的設定）**
  - 中斷進入點為 0x0008

中斷邏輯電路
傳統模式



中斷邏輯電路
優先權模式



Shadow 暫存器

- **18F4520有 “Shadow Register”的設計，能提高中斷程式對事件的反應速度**
- **高優先權中斷**
  - W，BSR，STATUS 的存入/取出使用Shadow Register
  - 程式的返回：retfie FAST
- **低優先權中斷**
  - W，BSR，STATUS 的存入/取出則必需透過軟體堆疊
  - 程式的返回：retfie 0
- **以上處理，在 C18 編譯器均打點完畢，程式設計者無需傷神考慮**
- **結論: C 的中斷處理比組合語言更簡單方便**

設定高優先權中斷服務程式

- 利用前置處理指令 “#pragma code” 將程式區間的起點設定於高優先權中斷向量位址 - 0x0008，再使用 goto 指令將控制權轉移給中斷服務程式
- 利用前置處理指令 “#pragma interrupt”來指定函數為高優先權中斷服務程式 (可在程式任何位址)，處理完畢會自行用retfie FAST 返回

#pragma interrupt func-name save=*symbol list*

  - func-name :高優先權中斷服務程式名稱
  - save= symbol list : 在中斷服務程式中，須被額外保存的變數(例: save= FSR0, PRODL)

高優先中斷及其 ISR 的設定範例

```
#pragma code hi_vector=0x0008   // 設定中斷進入點
void isr_high_code(void)
{
    _asm
    goto       isr_high
    _endasm
}
#pragma code
//*************************************************
//* Function: isr_high(void)                      *
//*   - Received a serial data from RS-232        *
//*   - Save the received data to Rec_Data        *
//*************************************************
#pragma interrupt isr_high
void isr_high(void)
{
    Rec_Data=ReadUSART();
    PORTD=Rec_Data;
}
#pragma code
```

使用內建組合語言功能，轉移控制權到中斷服務程式(isr_high)

中斷服務程式(isr_high)

中斷的前置設定

- 由上一頁來看中斷服務的設定是很簡單
- 但別忘了還有一些暫存器須設定後，中斷才會真正的動作(以USART的接收為例)

1. 開啟中斷優先權的設定: RCONbits.IPEN=1;
2. 設定UART接收為高優先權: IPR1bits.RCIP=1;
3. enable USART的接收中斷: PIE1bits.RCIE=1;
4. 啟動高優先權中斷: INTCONbits.GIEH=1;

- 完成中斷的基本設定

設定低優先權中斷服務程式

- 利用前置處理指令 “#pragma code” 將程式區間的起點設定於低優先權中斷向量位址 - 0x0018，再使用 goto 指令將控制權轉移給中斷服務程式
- 利用 “#pragma interruptlow”來指定函數為低優先權中斷服務程式，返回方式是 retfie

#pragma interruptlow func-name save=*symbol list*

- func-name：低優先權中斷服務程式名稱
- save= symbol list：在中斷服務程式中，須被額外保存的變數 (例: save= FSR0, PRODL)

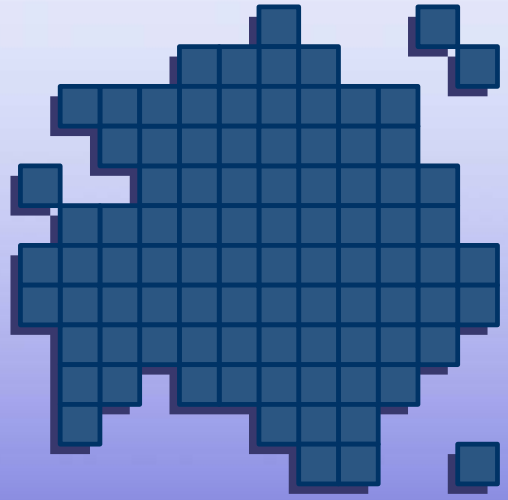
中斷處理程式 (ISR)注意事項

- 中斷函數無法傳遞參數
- 中斷所使用到的變數需加入 volatile 的宣告
- 中斷函數儘量不要使用 Local 變數，影響中斷響應時間
- ISR 應越短越好，不要做太多的處理，可以設定 Flag 後交給主程式處理
- 許多運算都會用到 tmpdata 節區，若ISR 中有複雜的計算 (乘、除、比較的運算)，就必須要 save tmpdata 這個節區，避免與主程式的暫存資料相衝突。
- 新版的 C18，會自動儲存 tempdata

tmpdata vs. 中斷

- tmpdata 在主程式與中斷函數共享
- 新版的 C18 會自動儲存 tmpdata，可防止中斷程式出錯，但也因要儲存 tmpdata 所以響應速度會變慢
- 如果中斷程式只是很簡單的處理並未使用到 tmpdata 就可以用底下的方式以加快中斷響應速度

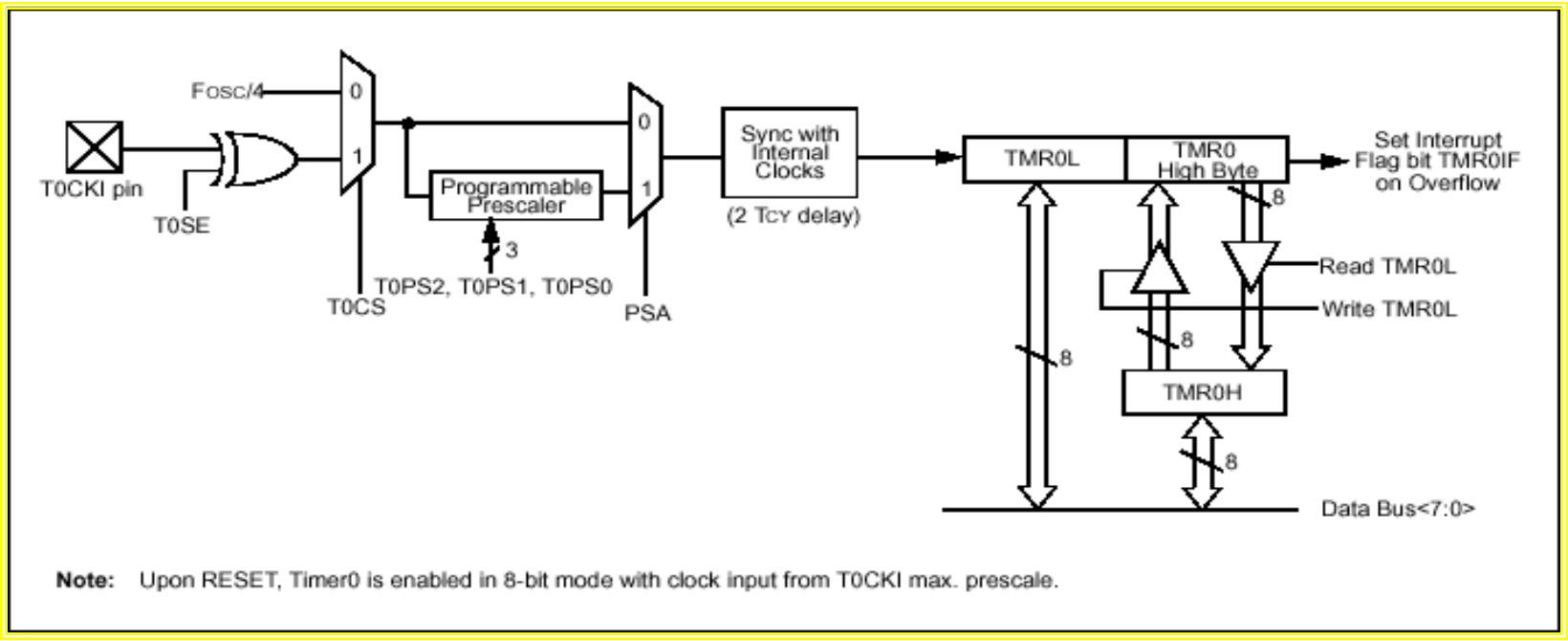
```
#pragma interrupt isr nosave=section(".tmpdata")
void isr (void)
{
...
}
```



PIC18 Family 內建的基礎周邊說明
Timers

PIC18 系列的周邊：Timer0

- **Timer0 可設定為 8-Bits 或 16-Bits 模式**
- **16-bit mode 時， TMR0H 會在讀寫 TMR0L 時才真正的被讀出或寫入 Timer0**
  - 可準確的由 8-bit 的 Data Bus 來讀取 16-bit 的 Timer 值
- **計時器產生溢位時 FFh to 00h (FFFFh to 0000h)，即產生中斷**

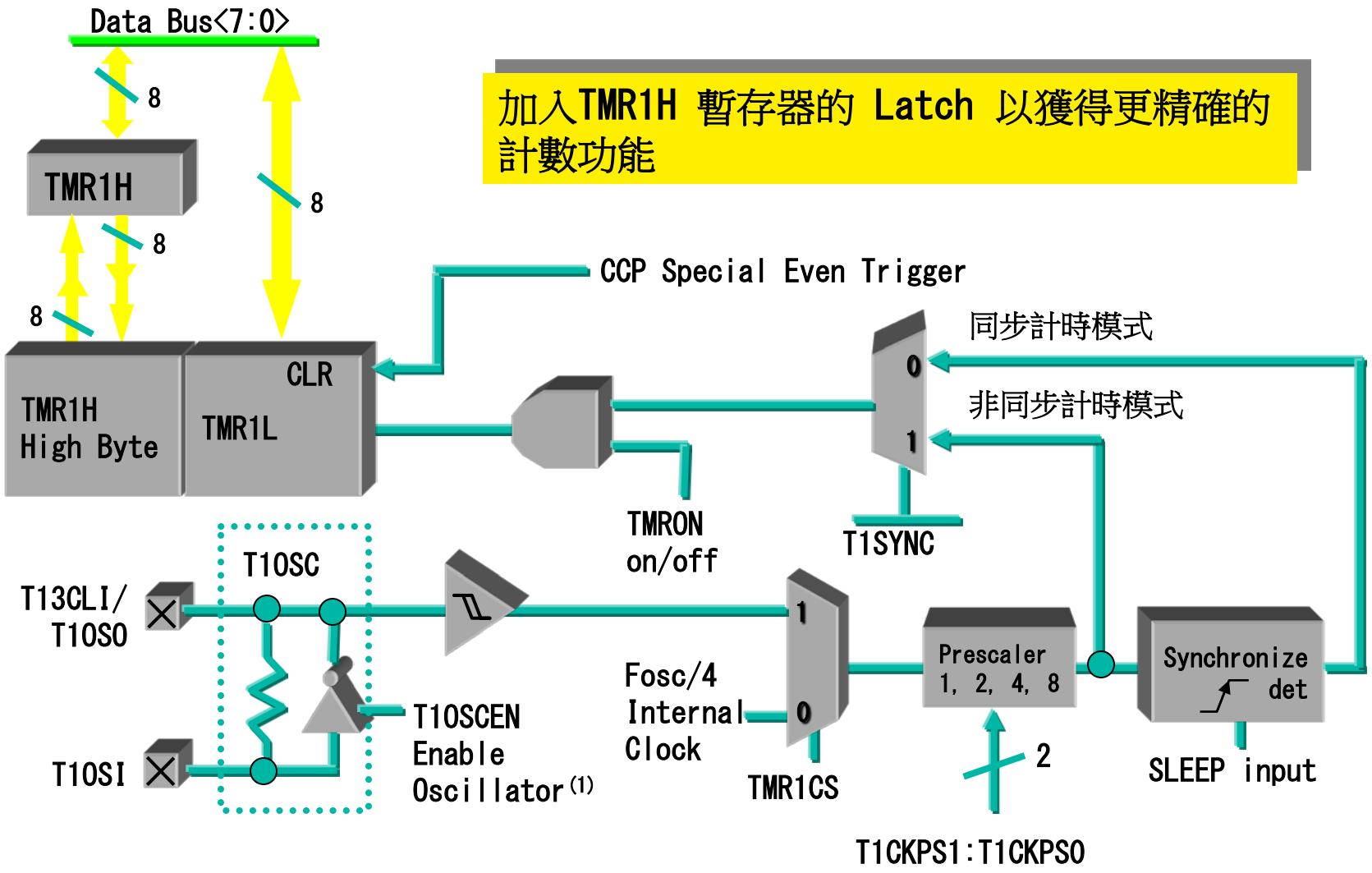


Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

PIC18 系列的周邊
Timer1 and Timer3

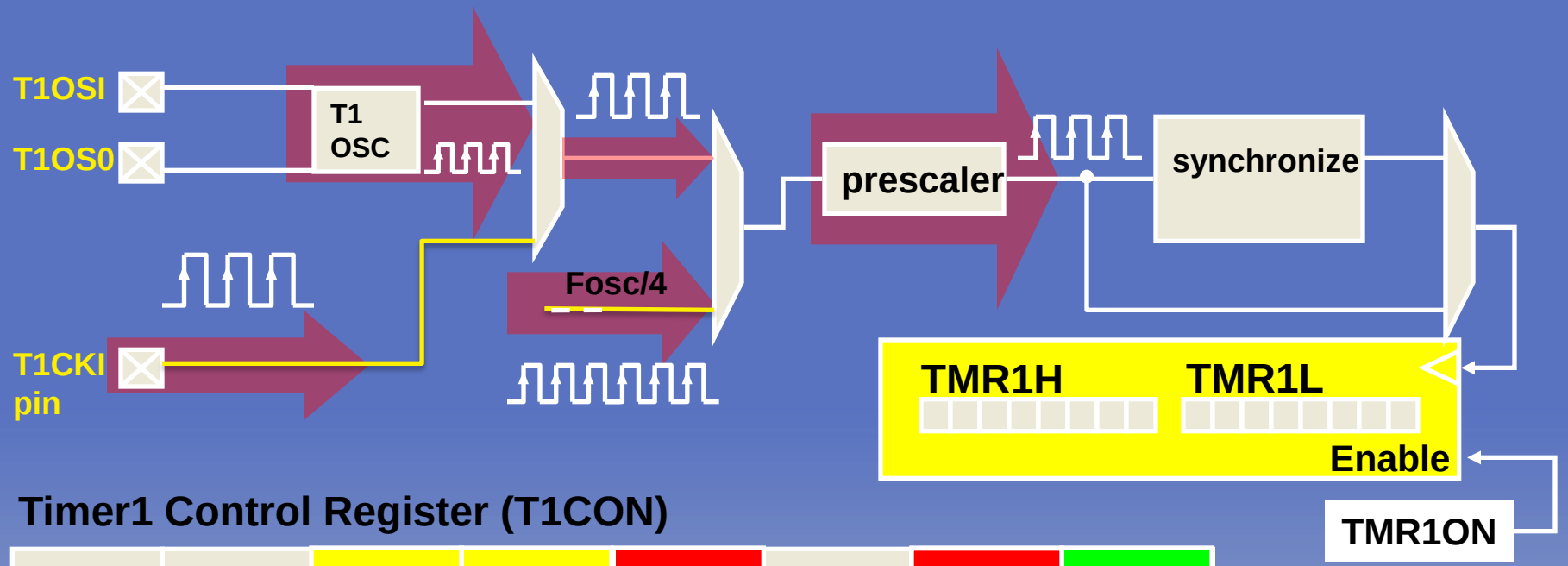
- 16-bit 模式的計數器或計時器
- 由兩個可讀/寫的 8-bit 計數器串聯而成
- 預除器有四種選擇 ：÷1， ÷2，÷4，or ÷8
- 三種功能：計時器，同步模式計數器 ，非同步模式計數器 (睡眠模式下使用非同步時序喚醒)
- 專用石英振盪電路可作為外部計數時序或第二系統時序 (System Clock) 選擇
- 當計數器或計時器產生溢位時 FFFFh to 0000h，即產生中斷

PIC18 系列的周邊
Timer1 and Timer3 *(continued)*



加入TMR1H 暫存器的 Latch 以獲得更精確的計數功能

Timer1 的運行圖解



Timer1 Control Register (T1CON)

RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

T1CKPS1	T1CKPS0	scale
1	1	1:8
1	0	1:4
0	1	1:2
0	0	1:1

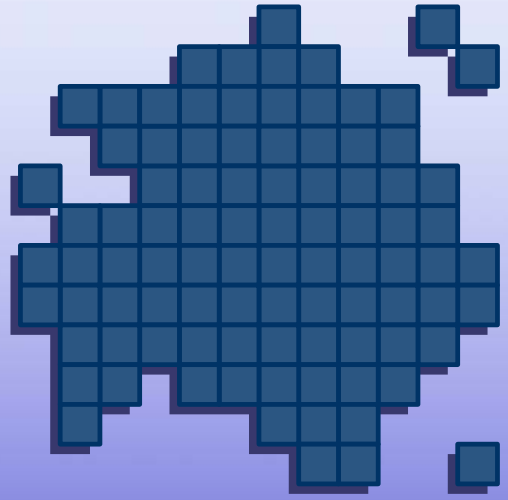
LP Oscillator Enable
1 = T1OSC selected
0 = T1CKI can be used

Clock Source Select
1 = External (T1CKI)
0 = Internal (F_{OSC}/4)

Timer1 On
1 = Enable Timer1

PIC18 系列的周邊
Timer2

- 8-bit 模式的計時器，有預除器及後除器之功能
- PWM 輸出模式下，基本的頻率來源
- TMR2 為一可讀、寫具有自動載入功能的計時器
- TMR2 會自動加一並與設定的值相比；若相等則送出訊號至後除器或產生中斷，並自動將自己清除為零，重新計時
- 可做為MSSP (SPI™) 傳送速率的設定



Lab3

Timer1 的規劃與應用

中斷服務程式的安排與處理

C18 對於 Timers 函式的應用

使用 Timer 中斷方式處理按鍵彈跳問題

周邊函數庫參考手冊

- 周邊函數的參考文件
  - C:\Program Files (x86)\Microchip\mplabc18\v3.43\doc\PIC18F Peripheral Library Help Document.chm

- **C18 所支援的標準函數庫 (clib.lib)**
  - ..\v3.43\doc\hlpC18Lib.chm
    - **Software Peripheral Library**
    - **General Software Library**
    - **Math Library**

PIC18F 周邊函數庫手冊

- 先選定所使用的元件 **(PIC18f4520)**
- 選擇 **Timer (45xx Family)** 的周邊，進入 **Timer** 的函數
- 先參考底下的 **Timer Examples** 範例的撰寫即使用到的函數
- 開啟**OpenTimer1( )** 的函數使用方式及了解參數的定義

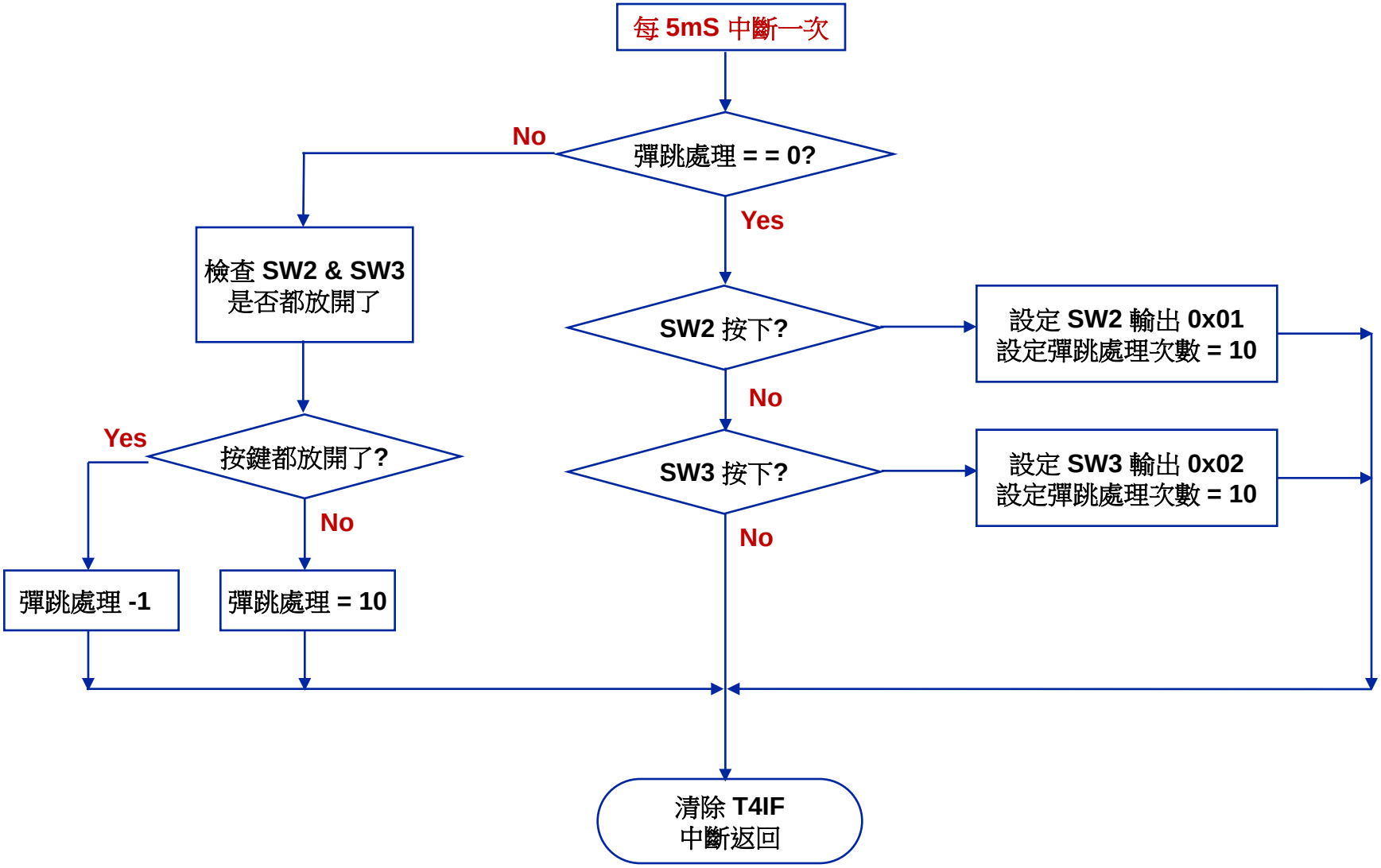
Lab3 動手做

Timer1 與 Interrupt 的應用

Lab Exercise 3 練習目標

- **Lab3 專案位置： ..\2014 Summer Techer Bootcamp \Section 1\Labs\Lab Exercise\Lab3\Lab3.mcp**
- **在 C 程式中安排高優先權的中斷服務程式**
- **學習使用 C18 的 Library 來規劃 Timer1**
  - 讓 Timer1 能每 5mS 中斷 CPU 一次，5mS 要怎樣計算
- **學習使用 MPLAB SIM 來觀察 Timer1 中斷的時間**
- **參考下一張投影片的流程圖，將 Lab2 的軟體延遲改成 Timer1 的中斷延遲**
- **使用 Timer1 中斷處理按鍵輸入及彈跳**
  - 這種處理方式稱之為“背景處理”，最大功能為主程式無需管他。
  - 這種處理方式有即時性的功能

使用 Timer 中斷方式 處理按鍵彈跳



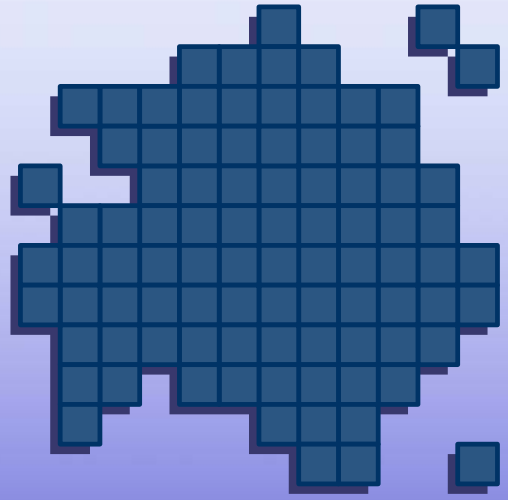
Lab3 要增寫的程式

- 尋找程式裡有 “**//???**” 的字，這表示這裡的程式需要修改。

1. 查詢一下 **PIC18F4520 Data Sheet** 來啟動高、低優先權中斷的設定修改
   - 行號 76 ~ 78
2. 修改 **Timer1** 的中斷函數
   - 行號 55 ~ 62
3. 參考周邊函數庫文件設定 **Timer1** 的輸入參數
   - 行號 116 ~ 117

Lab3 Interrupt
SW3=22 ;SW2=15

Lab3 程式執行時 LCD 的顯示



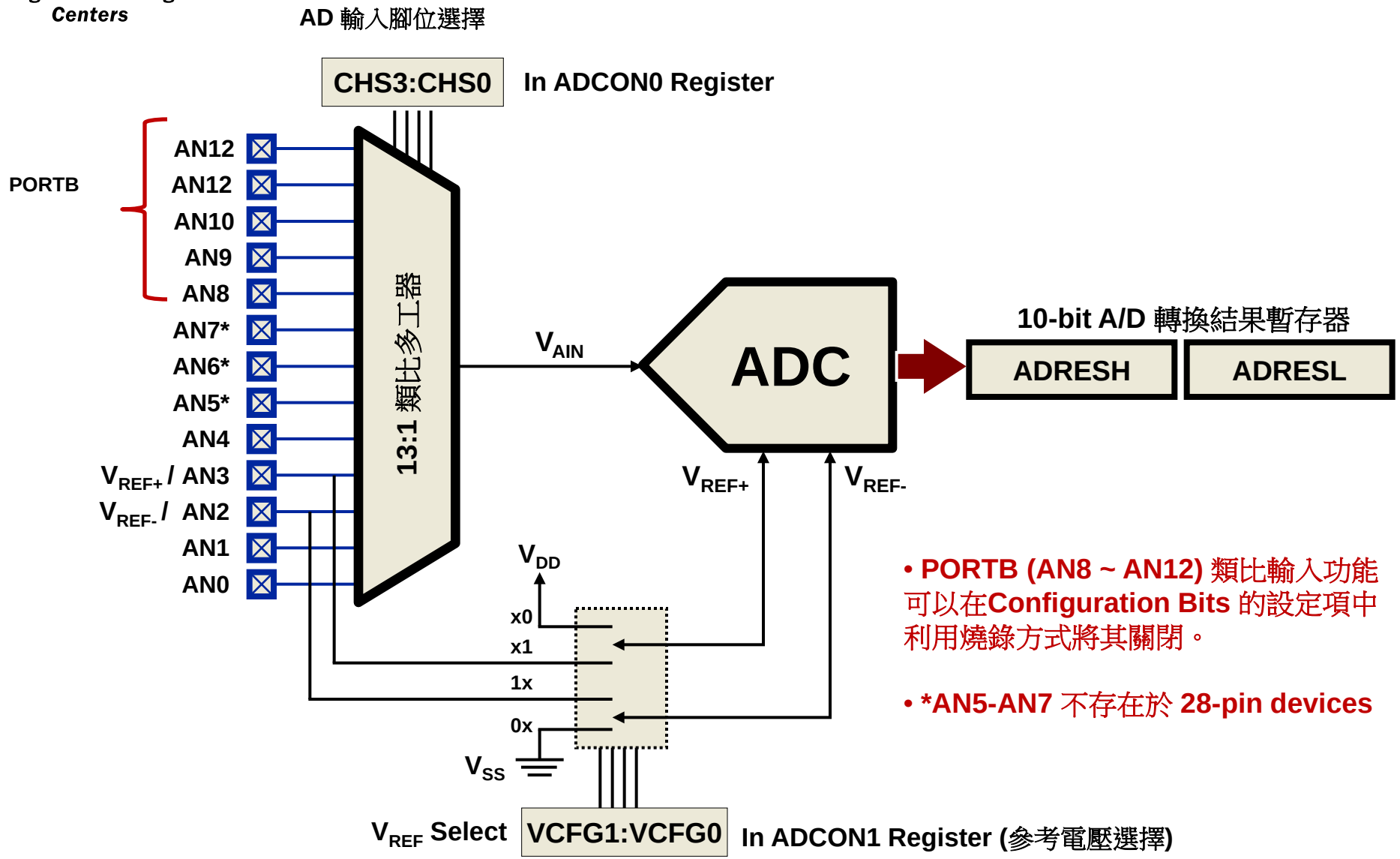
PIC18 Family 內建的基礎周邊說明

10-Bit Analog-to-Digital Converter

PIC18F4520 10-bit A/D 轉換器

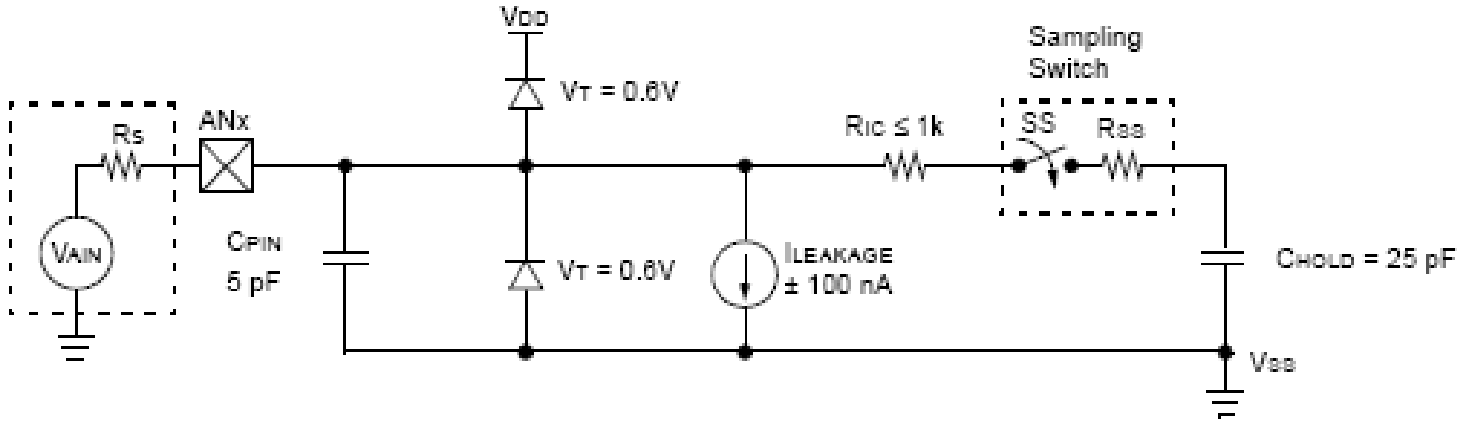
- **13** 組類比轉換多工輸入選擇，**10 bits** 解析度
- **AD** 最小時脈週期 **(T_{AD}) : 0.7uS (**外部震盪器**)**
- 類比輸入取樣時間：**1.4 µS (**輸入阻抗**<10K)**
- 類比輸入轉換時間：**8.4 µS (12 T_{AD})**
  - 8.4 µS (12 T_{AD}, T_{AD} 的最小時間為 0.7uS)
- **10-bit** 解析度時，只有一位元的誤差
- 允許使用外部參考電壓： **VREF+ & VREF-**
- 轉換的結果允許自動向左、向右對齊修正
- 完整的轉換時間共須 **9.8 µs**

10-bit A/D 方塊圖



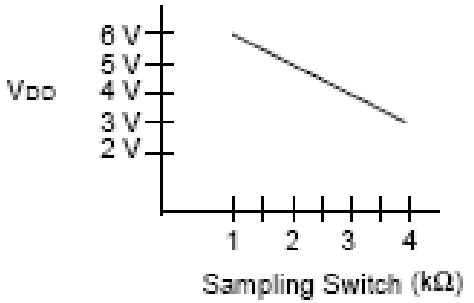
- **PORTB (AN8 ~ AN12)** 類比輸入功能可以在**Configuration Bits** 的設定項中利用燒錄方式將其關閉。
- ***AN5-AN7** 不存在於 **28-pin devices**

PIC18F4520 ADC 的輸入模型

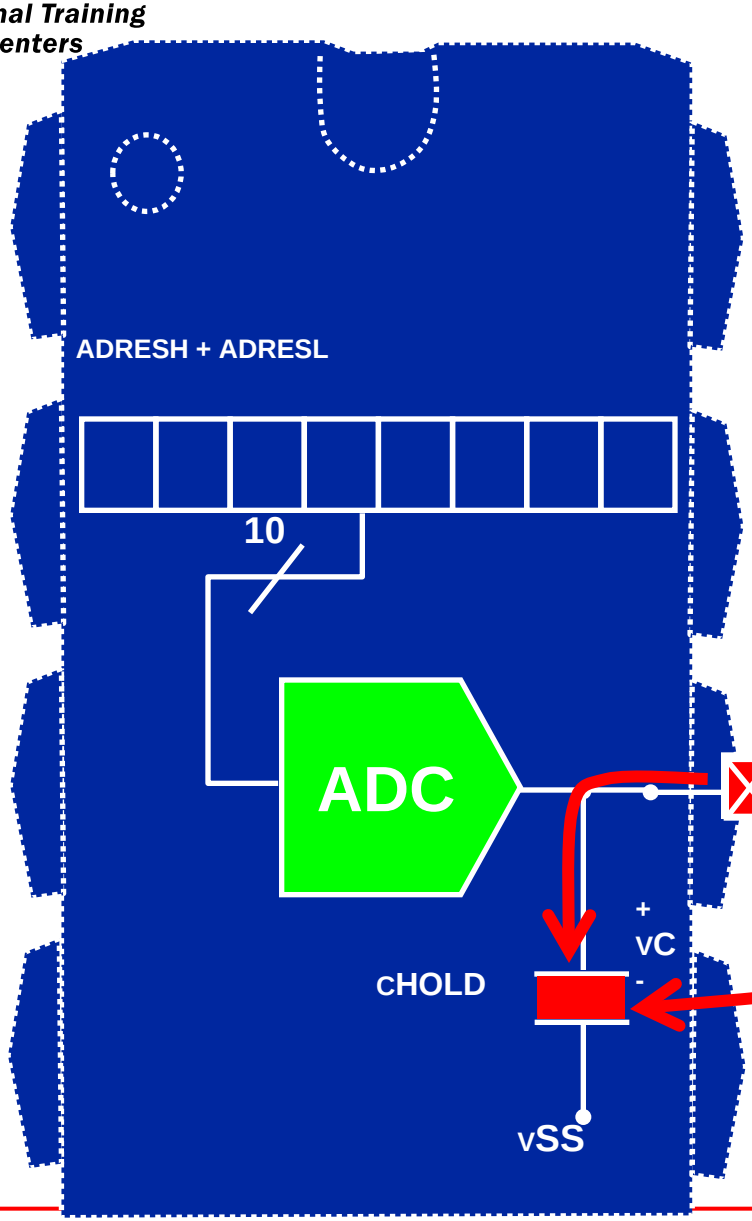


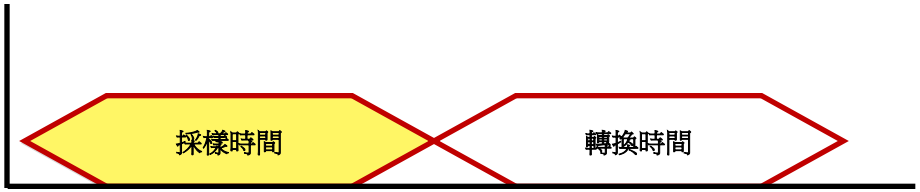
Legend:

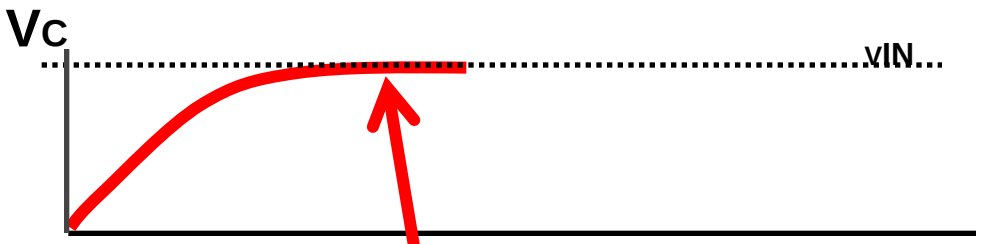
CPIN	=	input capacitance
VT	=	threshold voltage
ILEAKAGE	=	leakage current at the pin due to various junctions
RIC	=	interconnect resistance
SS	=	sampling switch
CHOLD	=	sample/hold capacitance (from DAC)
RSS	=	sampling switch resistance



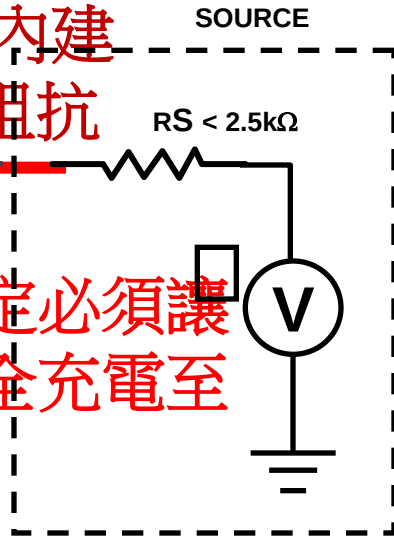
輸入訊號採樣時間





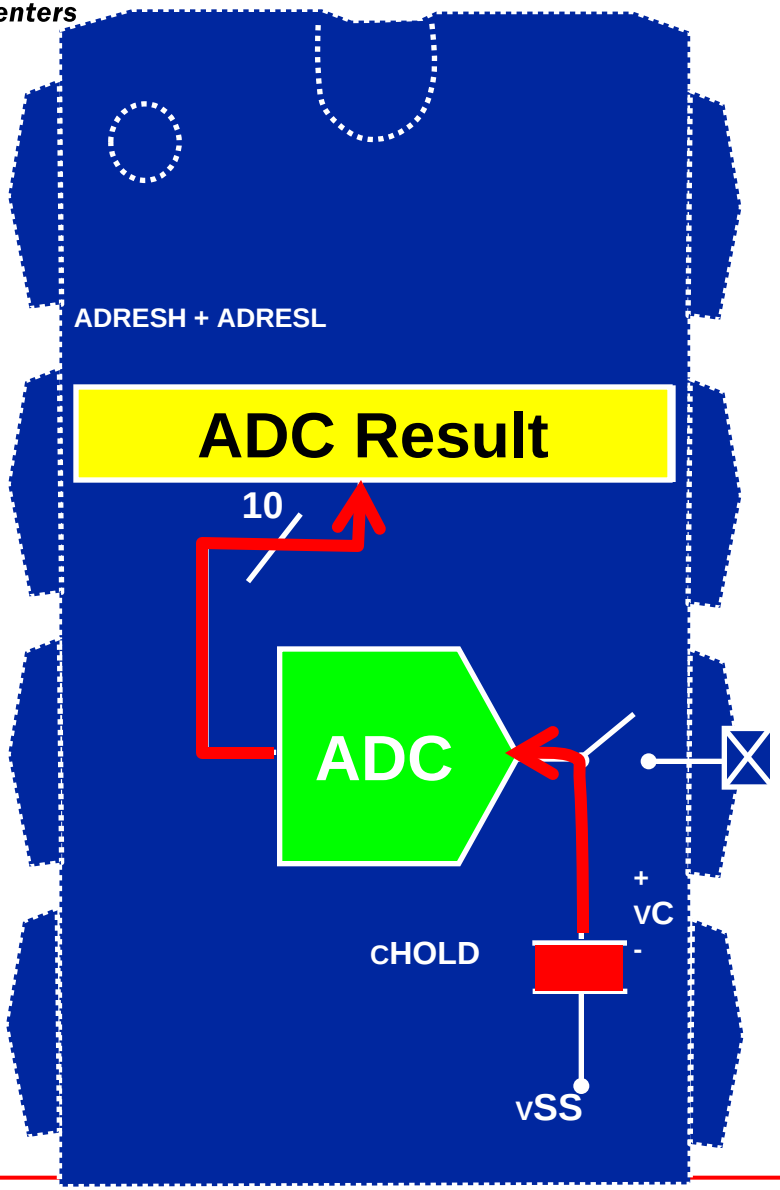


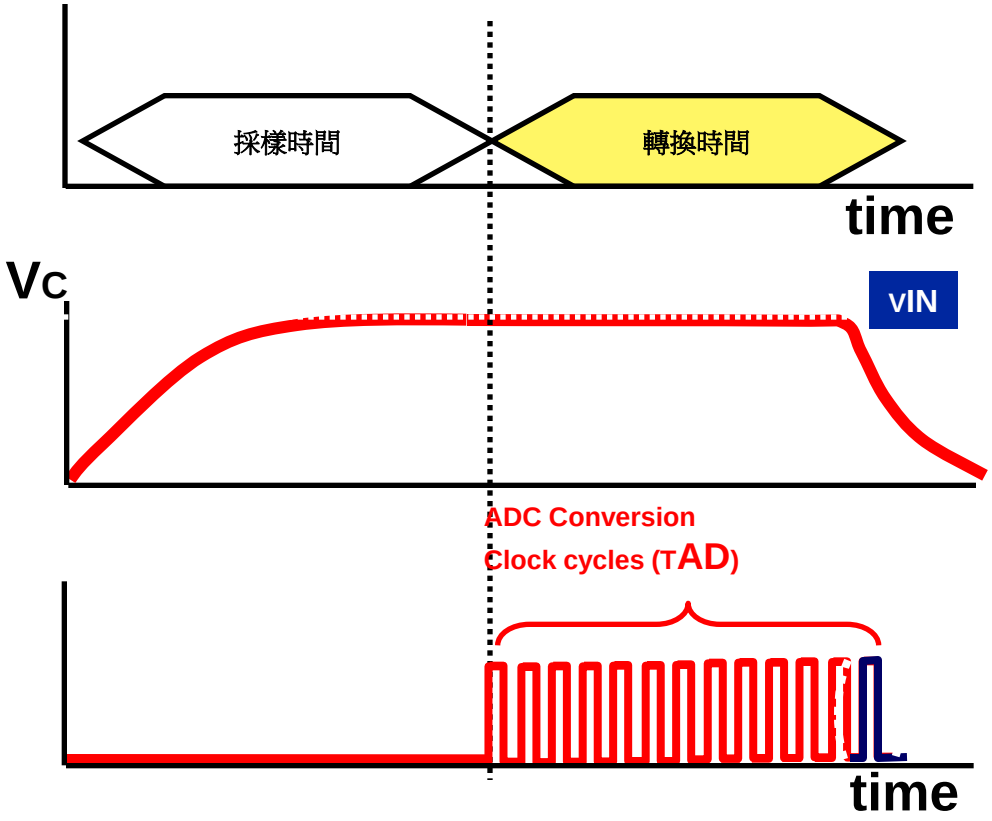
採樣時間取決於
輸入腳的寄生電容，內建
的採樣電容 及 輸入阻抗
(建議值 <10kΩ)



採樣時間的設定必須讓
採樣電容能完全充電至
輸入電壓 (V_{IN})

ADC 轉換時間





A/D 控制暫存器(3)

- 與 ADC 共用腳位的操作模式控制

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

A/D 控制暫存器(1)

ADCON0 REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

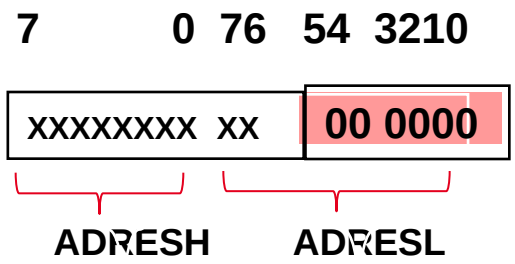
ADCON2 REGISTER

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

ADMF=0

ADC 的轉換結果為 10-bit
需要 2 Byte 才可容納

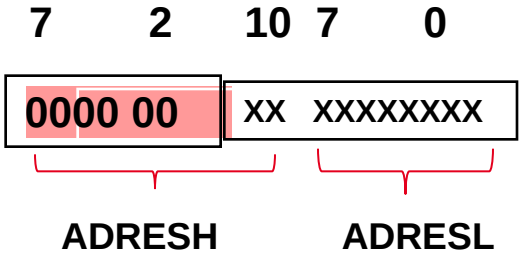
ADMF=1



10-bit

向左靠齊

A/D轉換結果可以
向左、向右調整



10-bit

向右靠齊

A/D 控制暫存器(2)

- **ADCON1**
  - 設定 ADC module 使用的參考電壓
  - 設定共用接腳的操作模式

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

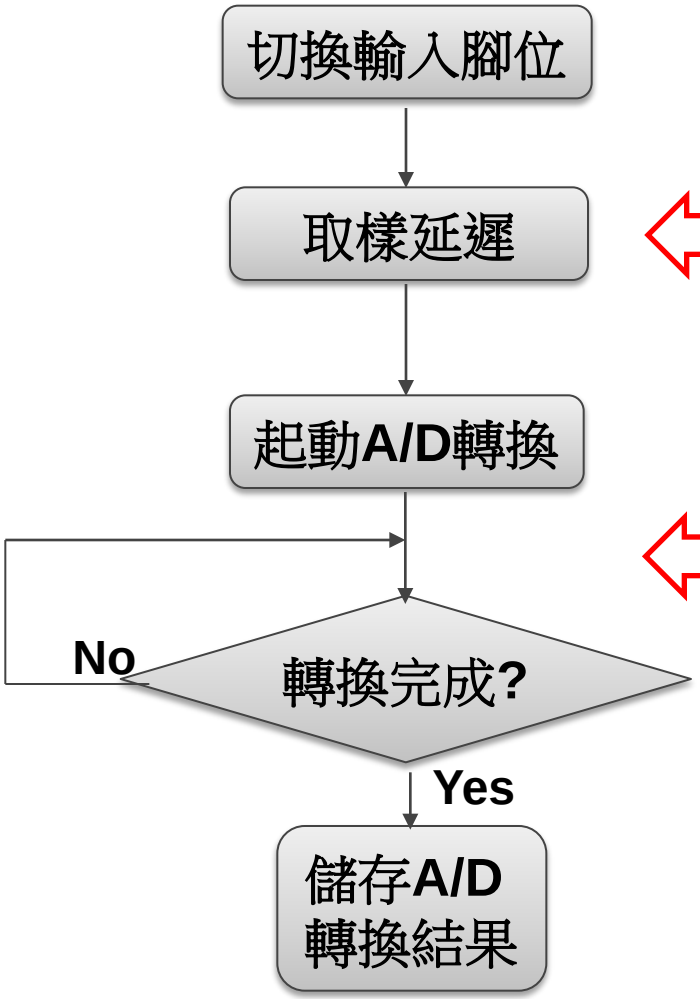
bit 7-6 **Unimplemented:** Read as '0'

bit 5 **VCFG1:** Voltage Reference Configuration bit (V_{REF-} source)
1 = V_{REF-} (AN2)
0 = V_{SS}

bit 4 **VCFG0:** Voltage Reference Configuration bit (V_{REF+} source)
1 = V_{REF+} (AN3)
0 = V_{DD}

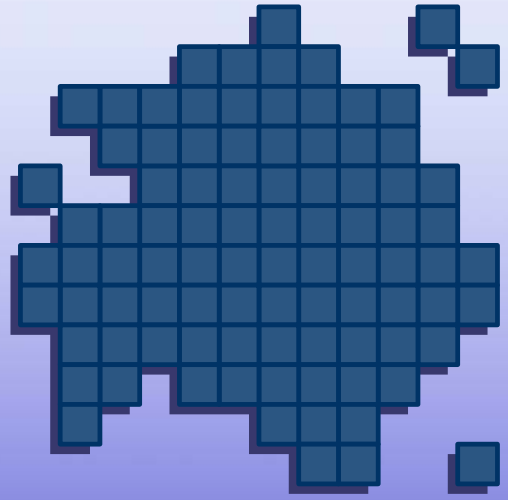
bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits:

A/D 轉換基本流程



根據不同的 MCU 延遲時間會有不同

PIC18F4520 可以用 ADCON2 來設定要用硬體取樣及取樣所需時間

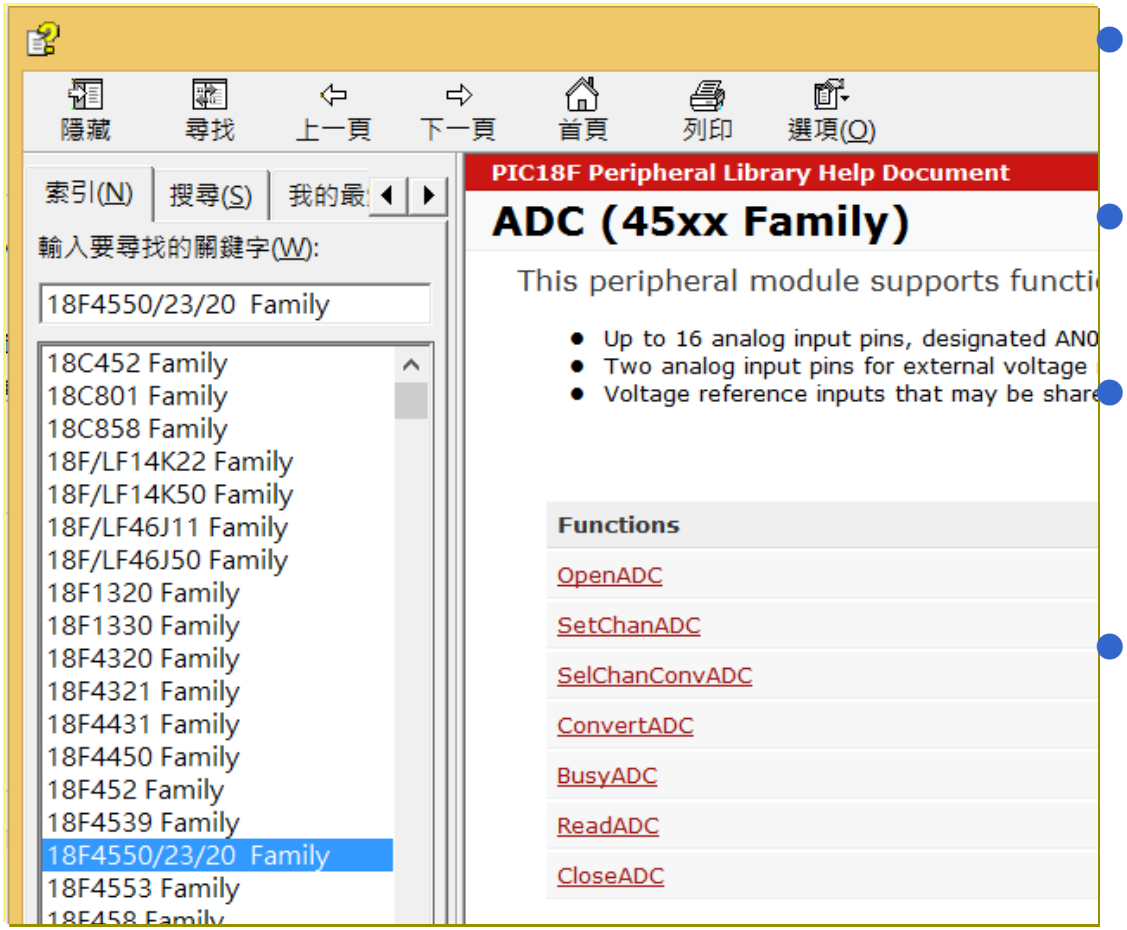


練　習　四

10-Bit Analog-to-Digital 轉換器的規劃與使用

活用MPLAB C18 ADC 周邊函式

PIC18F 周邊函數庫手冊



- 先選定所使用的元件 **(PIC18f4520)**
- 選擇 **ADC** 的周邊，進入 **ADC** 的函數
- 先參考底下的 **ADC Examples** 範例的撰寫即使用到的函數
- 開啟**OpenADC( )** 的函數使用方式及了解參數的定義

使用 C18 的 A/D 轉換函數

- 使用A/D轉換函數庫時，須同時使用定義檔 “adc.h”

  使用範例 ： #include <adc.h>

- ADC_V5 版本下常用的 ADC 函式
  - void OpenADC (unsigned char config ,
    unsigned char config2,
    unsigned char portconfug);

    使用範例 ： OpenADC (ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_4_TAD,
    ADC_CH0 & ADC_INT_OFF & ADC_REF_VDD_VSS,
    ADC_1ANA);

  - void ConvertADC (void)

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

  - void SetChanADC (unsigned char channel)

```
SetChanADC(ADC_CH0);
```

  - int ReadADC (void)

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

C18的 A/D 版本的宣告

- C18 使用 p18cxxx.h 做為元件暫存器的名稱定義
- C18 也會使用 GenericTypeDefs.h
- C18 對周邊有各種不同的版本宣告，參考 pconfig.h 檔
  - 啟用元件編號搜尋，就可得知周邊涵式的版本
  - 如需修改原始程式的函式必須知道版本的宣告

```
#ifdef  __18F4520
/*############################################################*/
/*        Configuration for device =  'PIC18F4520'        */
/*############################################################*/
/* ADC */
#define ADC_V5

/* ECC */
/*No configuration chosen for this peripheral*/
/* CC */
#define CC_V2

/* EPWM */
#define PWM_V5
```

如有需要修改周邊函數的原始程式時
就需知道該周邊是屬於那個的版本後
再去修該該版本週邊的原始程式

Lab4 開啟 ADC

- **OpenADC (ADC_FOSC_16 & ADC_RIGHT_JUST & ADC_4_TAD , ADC_CH0 & ADC_INT_OFF & ADC_REF_VDD_VSS , ADC_1ANA);**
  - 開啟 **ADC** 所使用的參數

- **ADC** 轉換程式

```
SelChanConvADC(ADC_CH0) ;
for (i=0;i<=10;i++);     // 延遲一小段取樣電容充放電時間
while(BusyADC()) ;
Update_ADC(ReadADC( )) ;
```

練習四

10-Bit ADC MPLAB C18 Library 的應用

Lab Exercise 4 練習目標

- **學習 ADC module 的規劃**
  - 使用 MPLAB C18 提供的 ADC 周邊 Library 來規劃 ADC module
- **使用 練習 3 完成的 1ms Timer1 中斷搭配軟體技巧得到一個 200ms 的 Time Event**
  - LCD 更新太快容易發生顯示閃爍的問題
- **當 100ms 的 Time Event 發生時，以 ADC 對 APP001 上的 VR1 做電壓的轉換**
  - 旋轉 VR1 ，當 ADC 轉換完成後可以看到 LCD 顯示 ADC的值

Lab4 ADC
ADC=1023 :0x03FF

Lab4 程式執行時 LCD 的顯示

Lab4 要增寫的程式

- 尋找程式裡有 “**//???**” 的字，這表示這裡的程式需要修改。

1. 使用 **ADC** 所提供的函數做 **CH** 的輸入選擇及 **ADC** 的轉換 **(SelChanConvADC & BusyADC)**
   - 行號 91 ~ 94
2. 參考 **OpenADC( )** 所需的參數項，依所需填入相關的參數設定
   - 行號 134 ~ 136
3. 利用 **puthexLCD( )** 來印出 **ADC Result** 的 **16** 進制值顯示到 **LCD** 上
   - 行號 112 ~ 113

LCD 與 LED 共用 PORTD

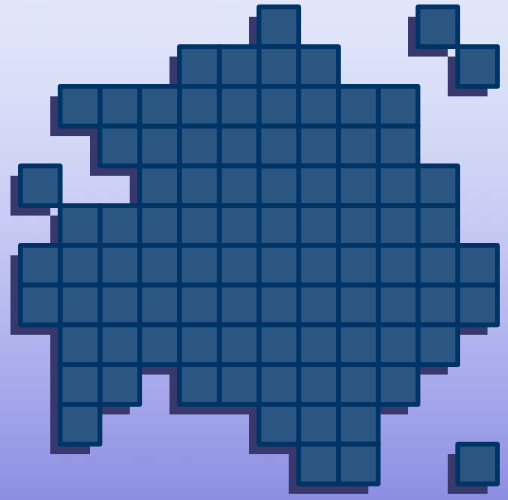
- 因為 LCD 與 LED 共用 PORTD 所以當更新LCD 時可以明顯看到 LED 也跟著閃爍
- 如何避免再更新 LCD 時，LED 的顯示依然不受 LCD 影響

Lab4_LED 動手做

LCD 與 LED 共用 PORTD

Lab4_LED 練習目標

- **Lab4_LED 專案位置： ..\2014 Summer Techer Bootcamp \Section 1\Labs\Lab Answer\ Lab4_LED\Lab4_LED.mcp**
- **在 LCD 函數裡，有那些函數會動到 PORTD 的**
  - WriteCmdLCD ( )
  - WriteDataLCD ( )
- **像處裡堆疊一樣，加入 PORTD 的儲存及取回**
  - #define LCD_DATA LATD
  - Temp_LCD_DATA = LCD_DATA ;
  - LCD_DATA = Temp_LCD_DATA ;

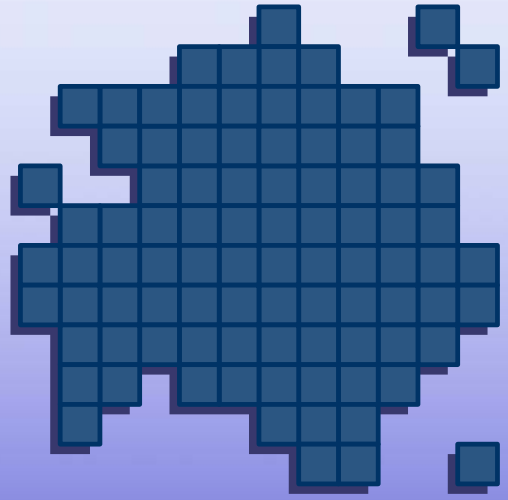


參考資料

MPLAB C18 Runtime environment

Memory Models

How to Override the Default Characteristics of Variables and Functions



The C Runtime Environment

A Foundation for C Programs

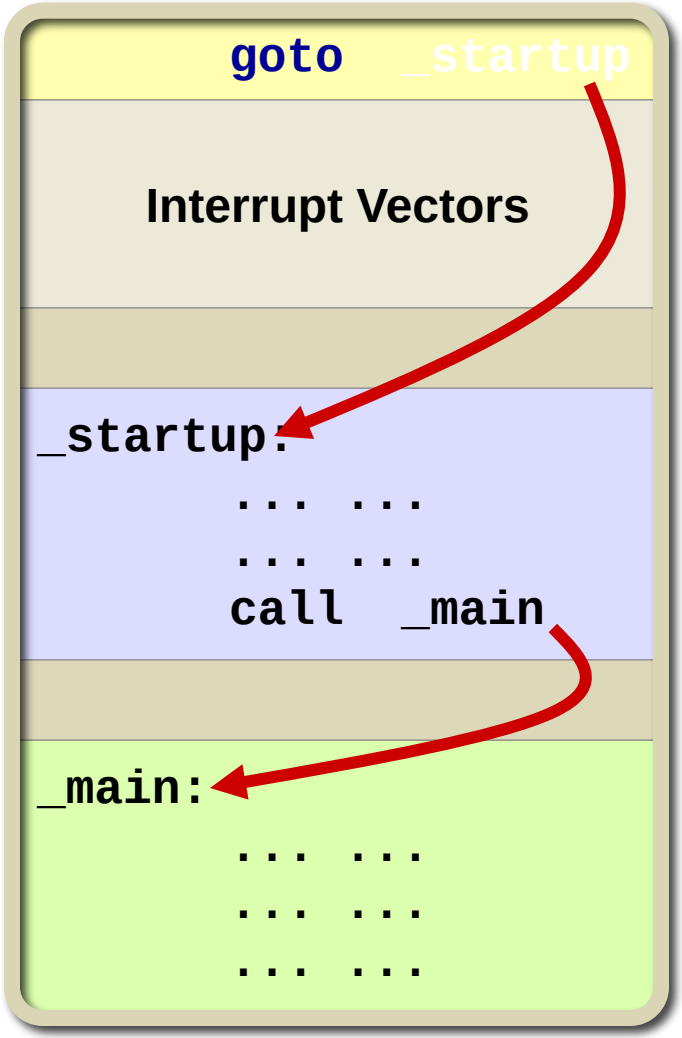
The C Runtime Environment

Linker Sections

Section Type	Default Name	Purpose
`code`	`.code_`*`filename`*	Program Code
`romdata`	`.romdata_`*`filename`*	Non-Volatile Data
`udata`	`.udata_`*`filename`*	Uninitialized Data
`idata`	`.idata_`*`filename`*	Initialized Data
`stack (udata)`	`.stack`	C Runtime Stack

- Linker creates sections of the appropriate types for each input file (*.o)
- You can create your own sections to group related items together in memory

Startup and Initialization



Reset Vector (Address 0x000000)
Populated automatically by MPLINK
Calls runtime environment setup code in c018.o (`_startup` label)

c018.o c018i.o
c018_e.o c018i_e.o
C Runtime Environment Setup Code
Inserted automatically by MPLINK Linker
(Source files: c018*.c)

main.c
Your C code's `main()` routine.
Included by you in your project and placed in memory by MPLINK Linker

Startup and Initialization

Tasks Performed by Default Startup Module `c018i.o`

- Initialize SP (FSR1) and FP (FSR2)
- Copy data from `.dinit` in flash to initialized data sections in RAM
- Call `main()` with no parameters
- `main()` is called inside an infinite loop
- Does **NOT** clear uninitialized data sections as per ANSI requirements – must use c018iz.o instead to be ANSI compliant
- Extended mode uses c018i_e.o instead

Startup and Initialization

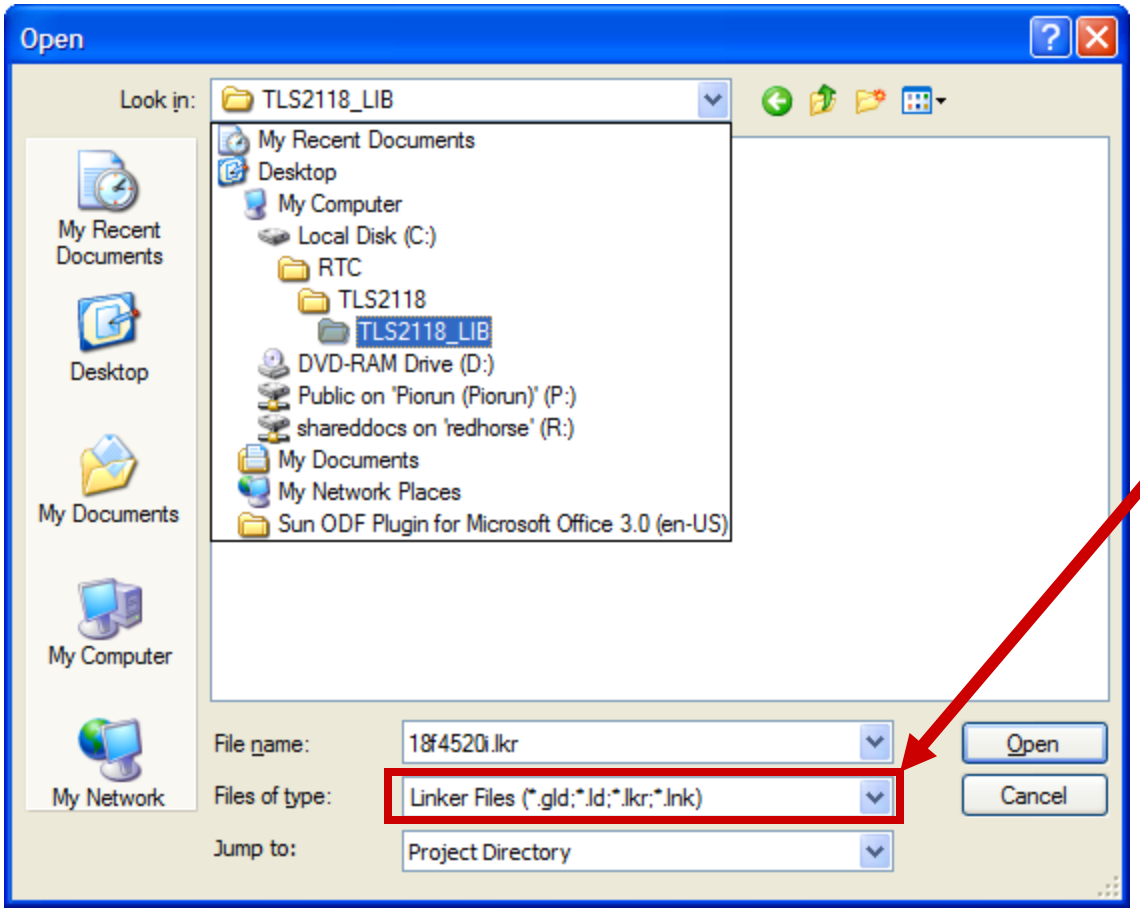
Tasks Performed by Alternate Startup Module `c018.o`

- **Performs all steps taken by c018i.o except:**
  - Does NOT load initialized data sections with values from `.dinit`
- **Alternate startup module is much smaller**
- **Can be selected to conserve program memory if data initialization is not required**

Startup and Initialization

How to Select an Alternate Startup Module

1 Open the Linker Script File



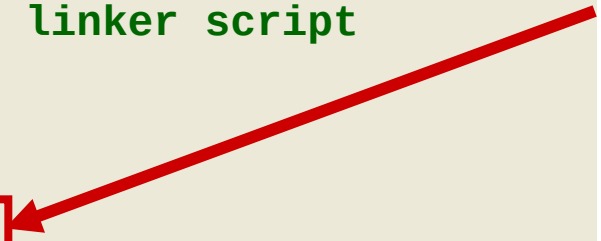
- If linker script isn't already in the project tree, from the menu select File ▸ Open...
- Select Linker Files in the Files of type combo box
- Navigate to the location of linker script and open it (*.lkr)

Startup and Initialization

How to Select an Alternate Startup Module

2 Edit the appropriate FILES entry to specify the desired startup file

```
// File: 18f4520i.lkr
// Sample ICD2 linker script

LIBPATH .

FILES c018i.o
FILES TLS2118.lib
FILES clib.lib
FILES p18f4520.lib

CODEPAGE   NAME=page       START=0x0
CODEPAGE   NAME=debug      START=0x7D
CODEPAGE   NAME=idlocs     START=0x20
CODEPAGE   NAME=config     START=0x30
CODEPAGE   NAME=devid      START=0x3F
CODEPAGE   NAME=eedata     START=0xF0
```



Change file name to one of the following:

Traditional Instruction Set

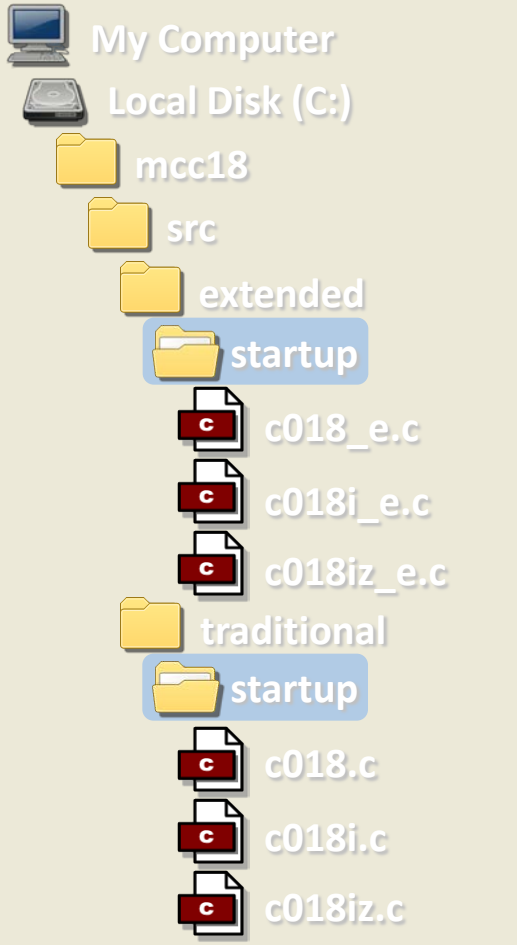
- **c018i.o** idata & udata
- **c018iz.o** idata & zeroed udata
- **c018.o** udata only

Extended Instruction Set

- **c018i_e.o** idata & udata
- **c018iz_e.o** idata & zeroed udata
- **c018_e.o** udata only

Startup and Initialization

Modifying the Startup Module



- **Startup modules may be modified if needed**
  1. Make desired changes
  2. Compile single file (.c → .o)
  3. Copy new c018*.o to: c:\mcc18\lib
- **To run code at power up before anything else is done, add it to the `_entry()` function of the startup module**

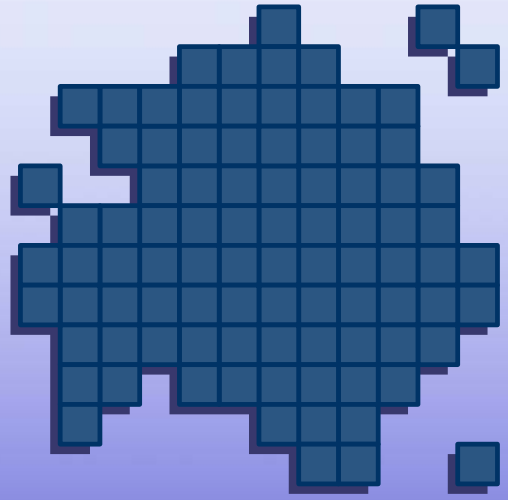
MICROCHIP

Regional Training Centers

課程結束

後面的投影片為參考附錄

介紹 C18 更詳細的用法



Memory Models

Object Allocation Schemes

Memory Models

Overview

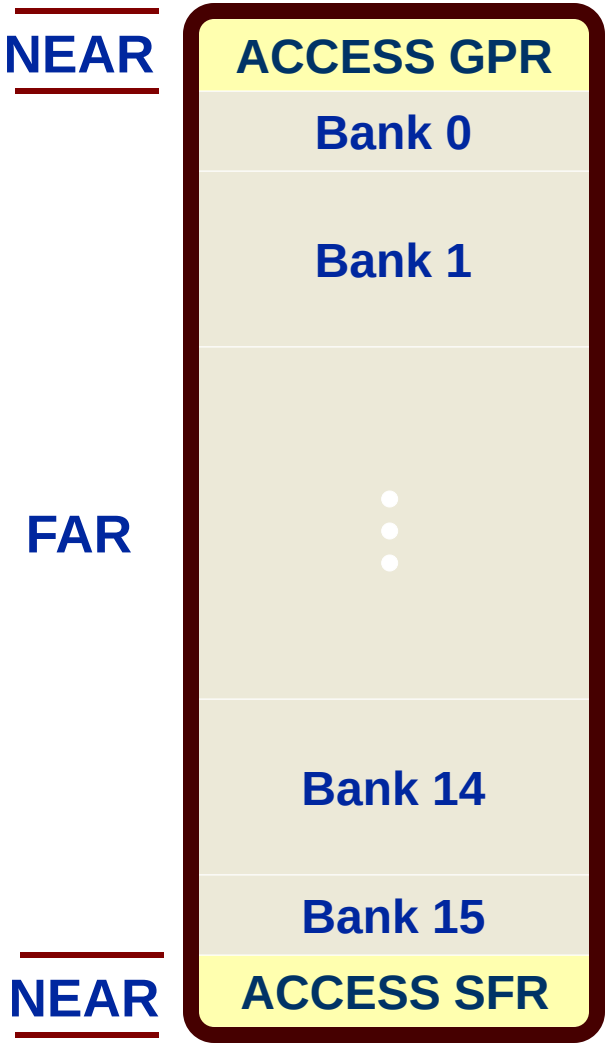
Option	Description		
Small `-ms`	For programs < 64KB Pointers to Data Space: Pointers to Program Space:	 16-bits 16-bits	For data < 128B All data resides in the Access bank
Large `-ml`	For programs > 64KB Pointers to Data Space: Pointers to Program Space:	 16-bits 24-bits	For data > 128B Data may reside in any bank

- **May override on a case by case basis with the `near` and `far` qualifiers**

The General Purpose RAM portion of the Access bank is not 128 bytes on all devices. It may be smaller for devices with more than 128 bytes of Special Function Registers (e.g. PIC18F8720: 0xF60-0xFFF vs PIC18F4520: 0xF80-0xFFF).

Memory Models

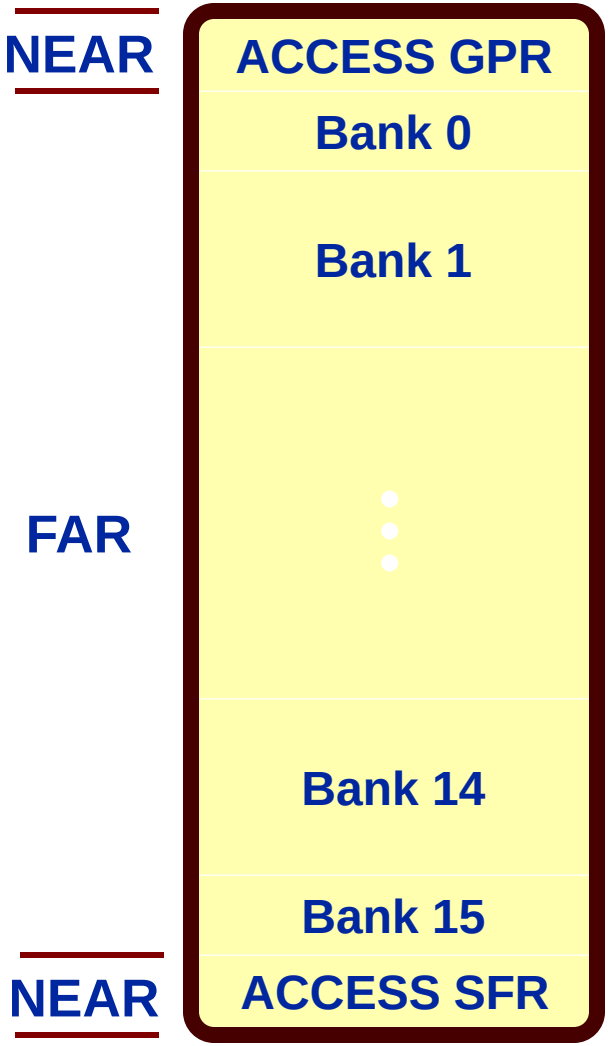
Small Data



- All data fits in ACCESS bank (usually 128B)
- No bank switching required
- Fastest data access
- Smallest code to access data
- Causes warnings when using libraries

Memory Models

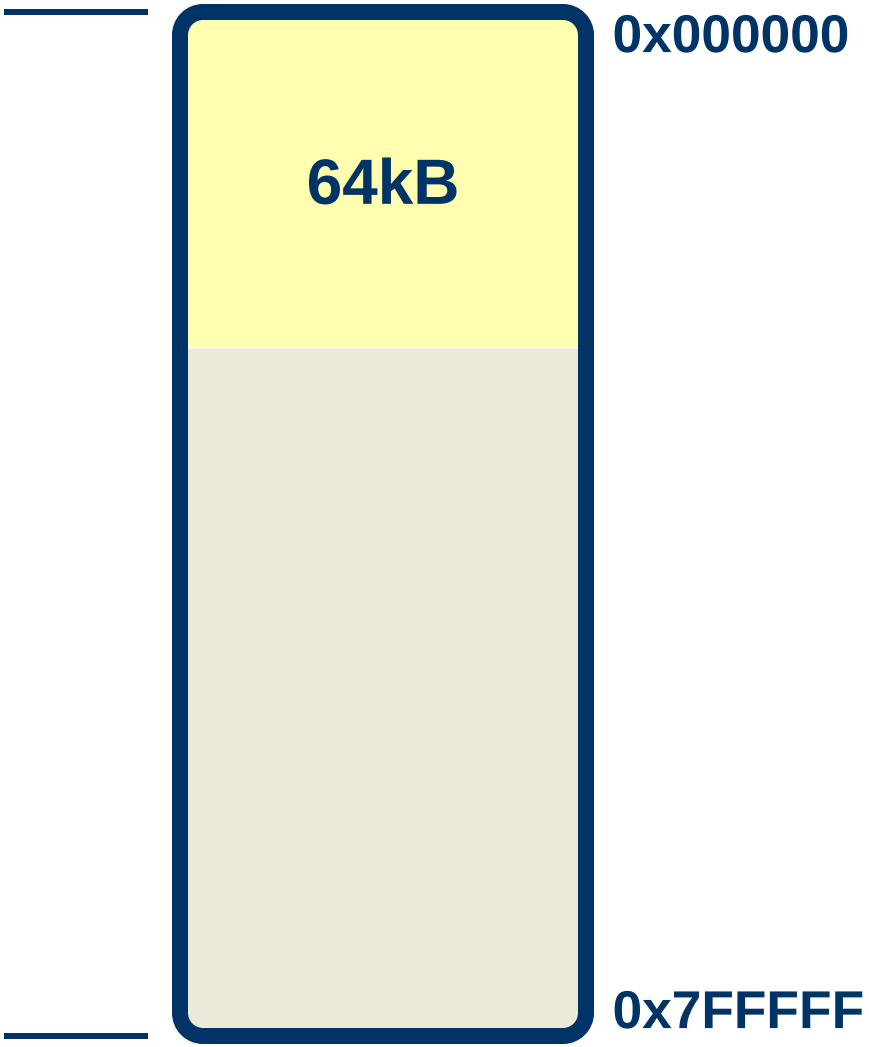
Large Data



- All data *cannot* fit in ACCESS bank
- Data can reside anywhere
- Bank switching required
- Data treated as if it were in `far` space
- Libraries are built using this memory model

Memory Models

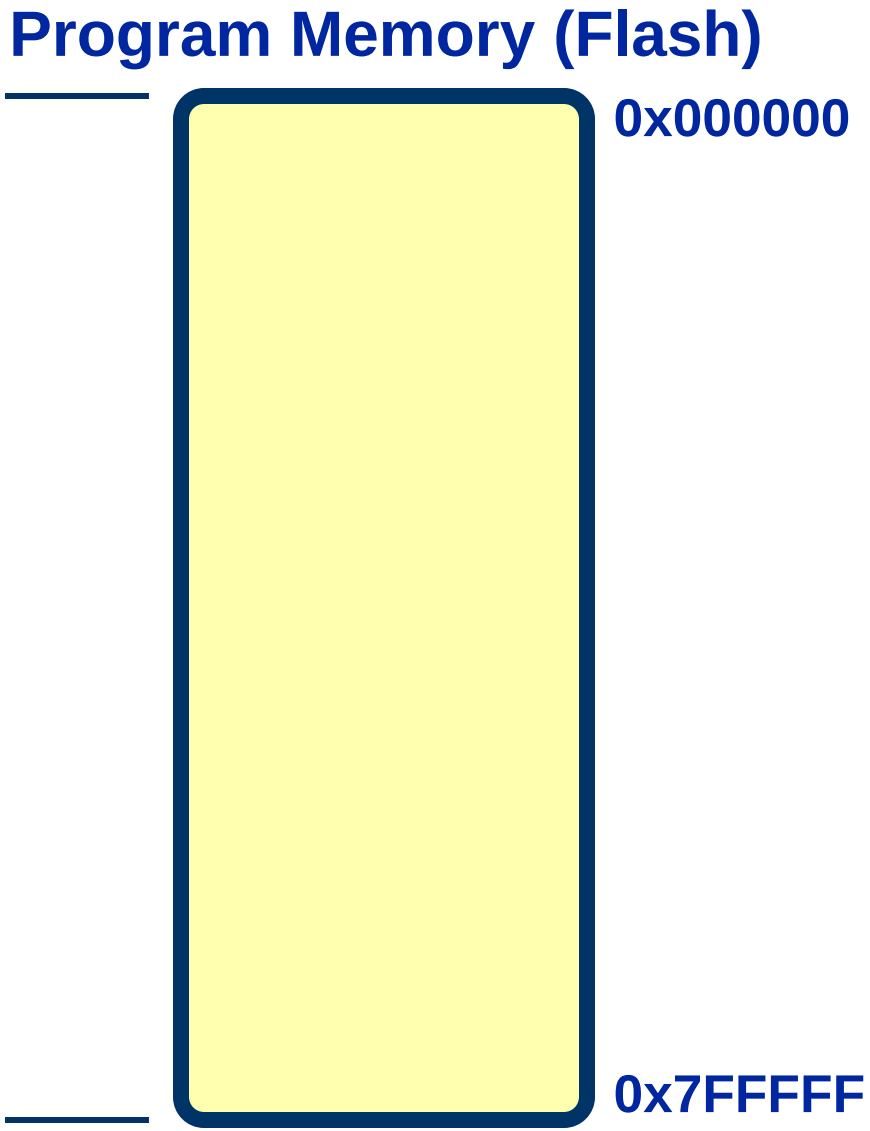
Small Code



- Default Model
- Program fits within 64kB
- ROM pointers (data and function) are 16-bits (same as RAM)

Memory Models

Large Code

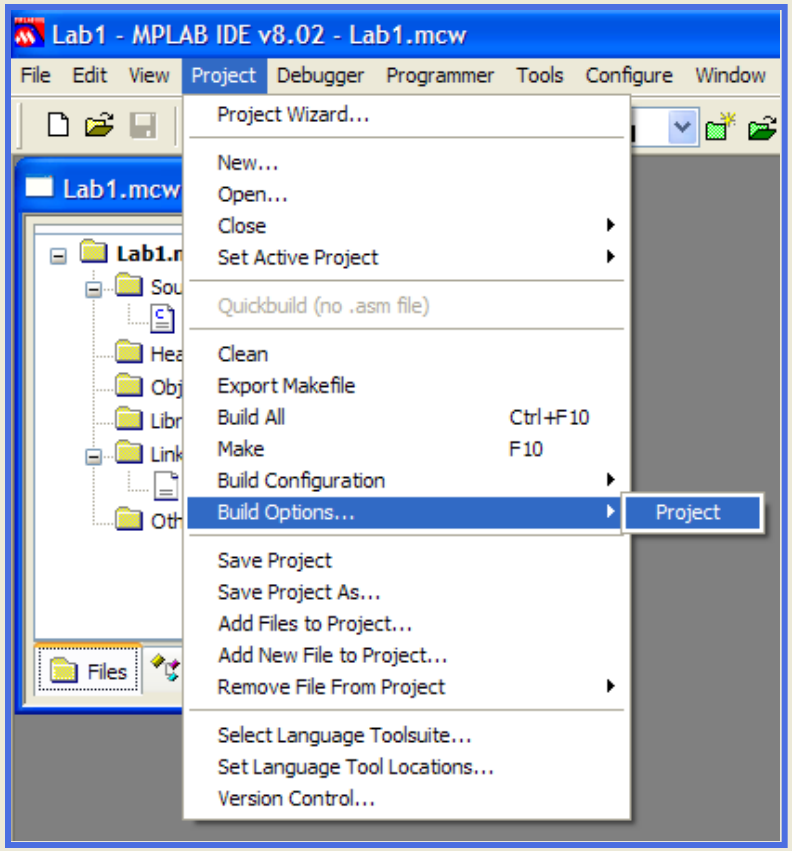


- Program *cannot* fit within 64kB
- ROM pointers (data and function) are 24-bits

Memory Models

How to Select the Memory Model

1 Open Project Build Options

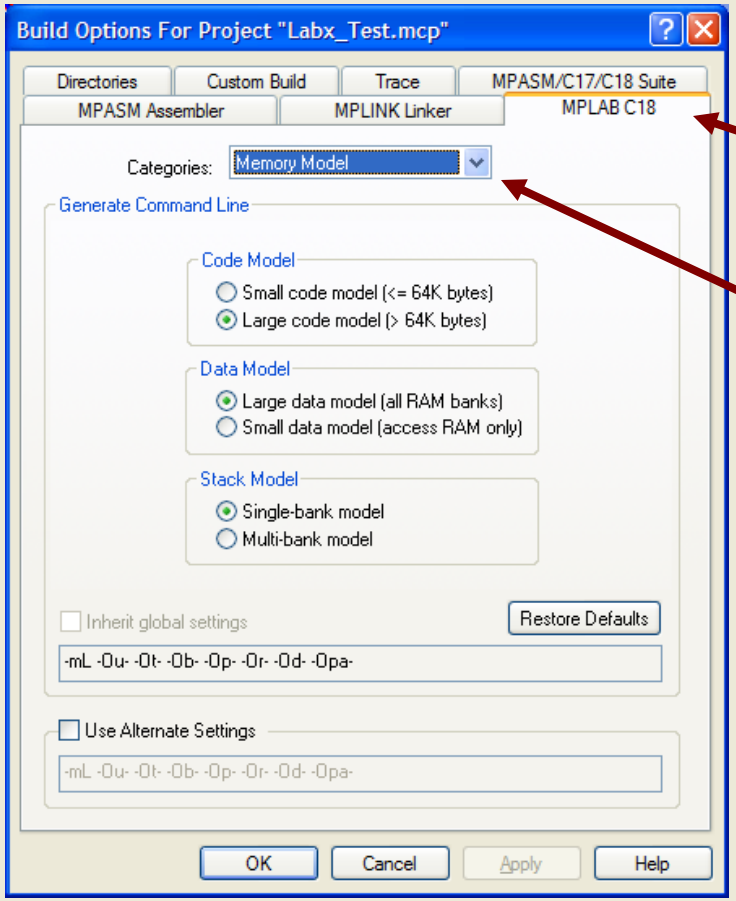


- From the menu bar, select:
  Project ▶ Build Options... ▶ Project

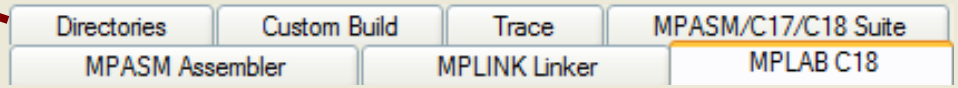
Memory Models

How to Select the Memory Model

2 Go to Memory Models



▶ Select the **MPLAB C18** tab

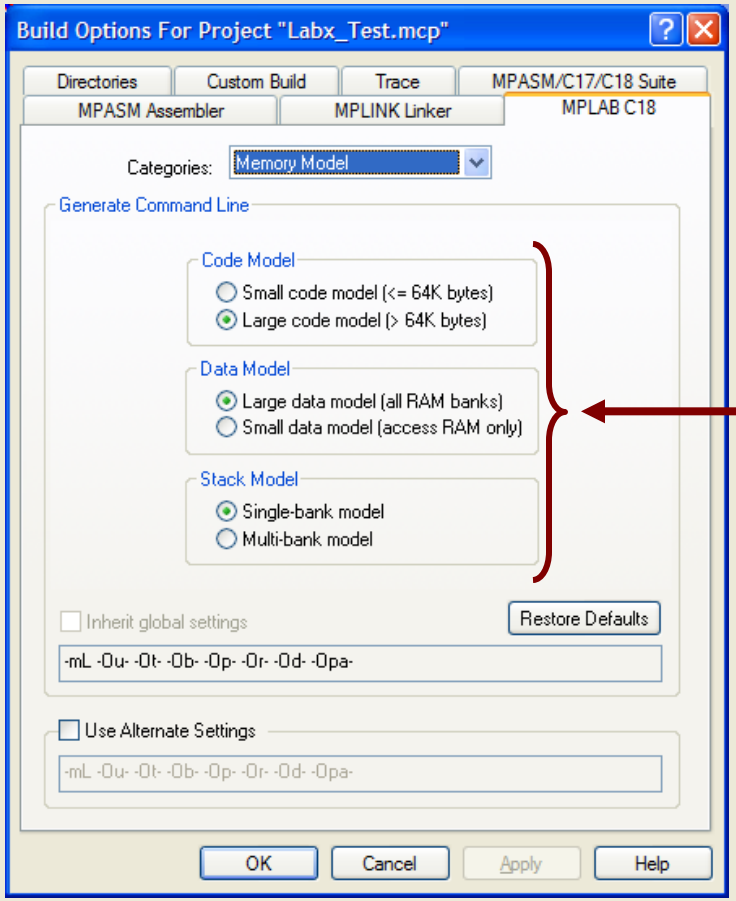


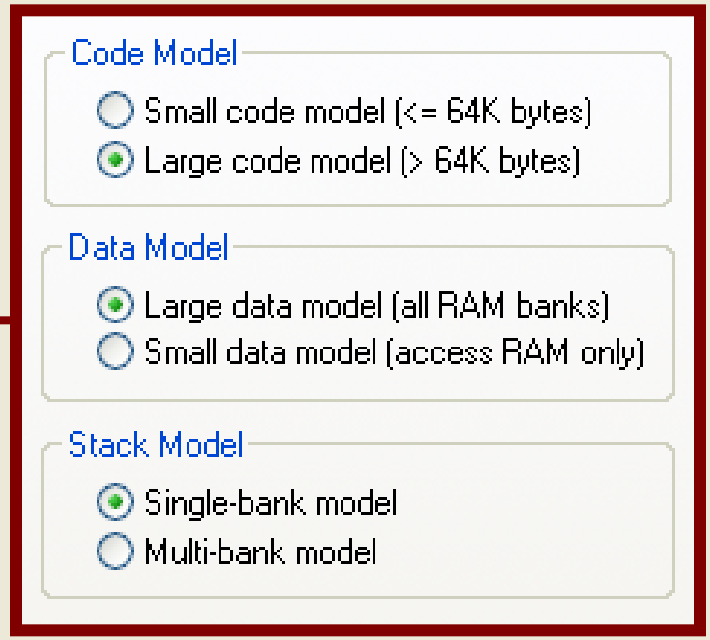
▶ Select the **Memory Model** category

Memory Models

How to Select the Memory Model

3 Select Desired Models





Memory Models

Tips & Tricks

- **Inappropriate model for your program can cause compile or link errors**
- **As your program grows, you may need to change the memory model**
- **If desired, you have full control over where objects are placed in memory**
  - Use small model, but force some objects into far memory
  - Compile different modules with different models

Memory Models

Tips & Tricks

- **Compiler can often generate more compact code if variables in the Access bank**

Option	Tips for Optimal Memory Use
Small Data	Use if all variables for the application can fit in Access bank If all data doesn't fit in near space, define some variables with the `far` qualifier so others have space to fit in near data.
Large Data	1. Compile some individual modules using Small Data. Then include their compiled object modules in the Large Data project. 2. Define individual variables with the `near` qualifier

Following the use of a `far` data pointer in a small memory model program, the TBLPTRU byte must be cleared by the user. MPLAB C18 does not clear this byte.

Memory Models

Tips & Tricks

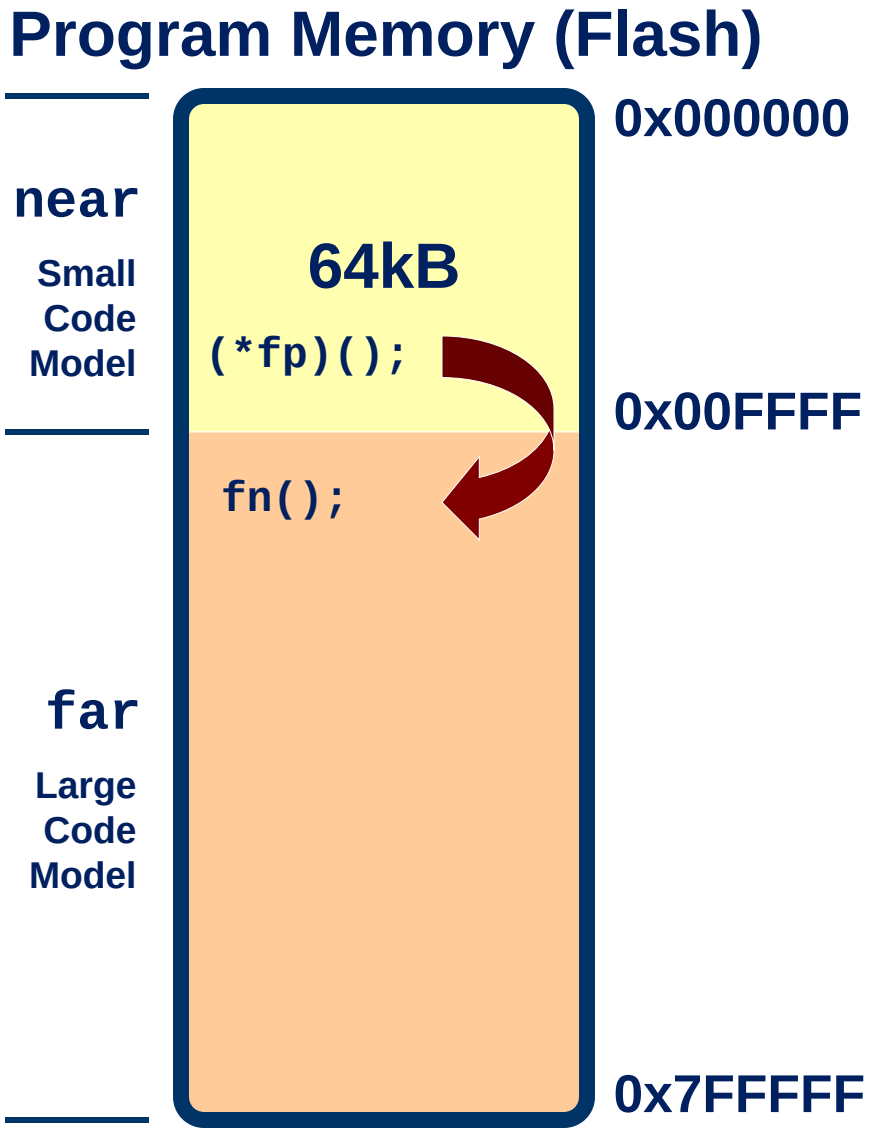
- **Functions that are near (in first 64kB) may call each other more efficiently**

Option	Tips for Optimal Memory Use
Small Code	1. Use if all functions are within first 64kB. 2. Compile some modules using Small Code and include their object files in a Large Code project. 3. If not all functions are in first 64kB, define some functions with the `far` qualifier.
Large Code	1. Define some functions with the `near` qualifier. An error will be generated if the function cannot be reached by one of its callers using the more efficient form of the function call.

Following the use of a `far` function pointer in a small memory model program, the PCLATU byte must be cleared by the user. MPLAB C18 does not clear this byte.

Mixing Memory Models

Small Code application with Large Code libraries



- Project built with small code model for efficiency
- Libraries built with large code model for flexibility
- Function pointers may be used to call far code library from small code application

Mixing Memory Models

Small Code application with Large Code libraries

Function in library built with large code model

```
void LCDPutStr(rom char *str);
```

Application built with small code model

1. Declare a function pointer and initialize it to point to the library function

```
far rom void (*fp)(far rom char *s)=(far rom void *)LCDPutStr;
```

Give function pointer and pointer to parameter ability to address beyond 64k

Typecast from pointer to `void` function to a pointer to `far rom void` function (Eliminates suspicious pointer conversion warning)

Mixing Memory Models

Small Code application with Large Code libraries

Function in library built with large code model

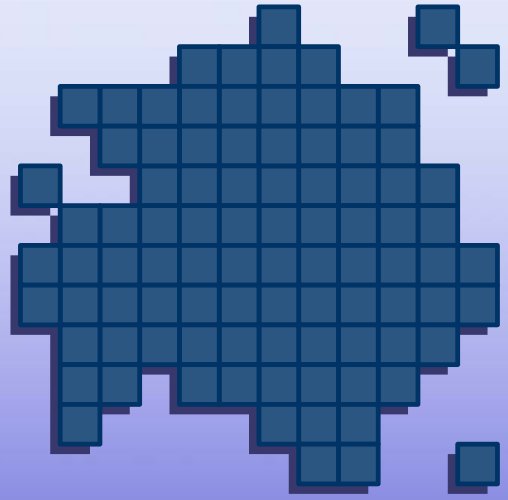
```
void LCDPutStr(rom char *str);
```

Application built with small code model

2. Call the library function via the pointer

```
fp((far rom char *) "Hello, world!");
```

Typecast from pointer to `rom char` to a pointer to `far rom char`
(Eliminates suspicious pointer conversion warning)



How to Override the Default Characteristics of Variables and Functions

Using type qualifiers and #pragma

Locating Variables and Code

Section Types

General Syntax

```
#pragma sectiontype sectionname=address
```

- Section types define the overall region of memory where an object is to be located

Section Name	Memory Region	Usage
udata	RAM	Static Uninitialized Data (e.g. char x;)
idata	RAM	Static Initialized Data (e.g. char x = 55;)
udata access	ACCESS RAM	Static Uninitialized Data in Access RAM
idata access	ACCESS RAM	Static Initialized Data in Access RAM
romdata	Program/Flash	Variables and Constants
code	Program/Flash	Executable Code

Locating Variables

How to place a variable in the Access Bank

Syntax

```
#pragma sectiontype access sectionname
near type identifier;
```

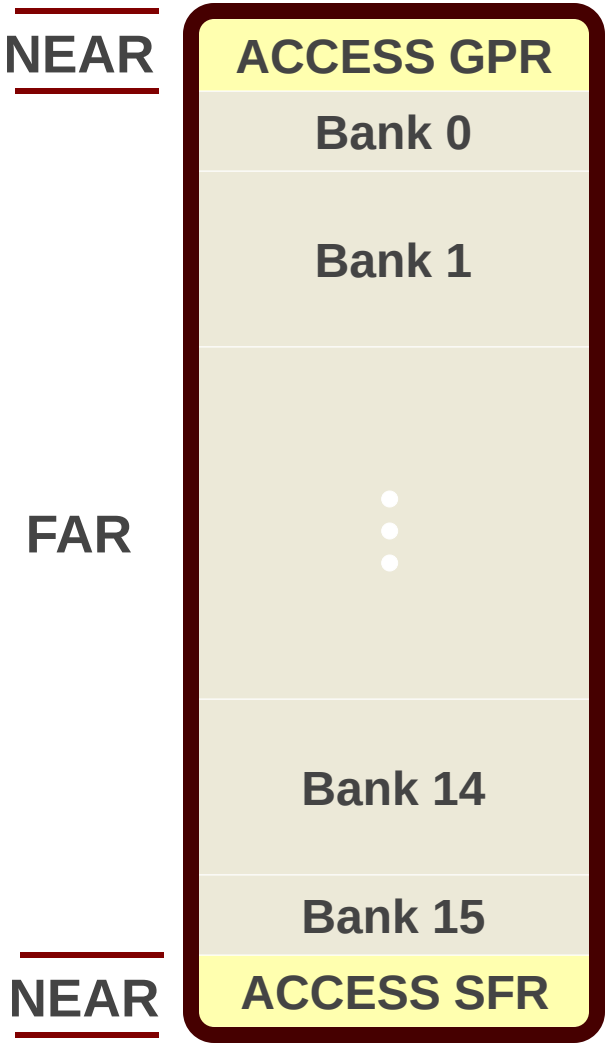
- The #pragma tells linker to place the following variable(s) in the access bank
- near tells the compiler not to generate bank switching instructions
- Can be accessed more efficiently

Example

```
#pragma udata access mySection
near int x;
near char a[10];
```

Locating Variables

Manually Optimizing Data Memory Use



- Use large data model
- Define frequently accessed variables with `near` qualifier

Locating Variables

Variable Allocation Example

near Variables with the Large Data Model

```
#include <p18f4520.h>

char a[124];

#pragma idata access myData
near char x = 0xAA;

int main(void)
{
    ...
}
```

1. **Variables are `far` by default (banked)**
   - a[] may be located anywhere
2. **To create `near` variables with the large data model:**
   - Declare a new `access` section and give it some name
   - Define variables with the `near` qualifier

Locating Variables

How to place a variable in far memory

Syntax

```
far type identifier;
```

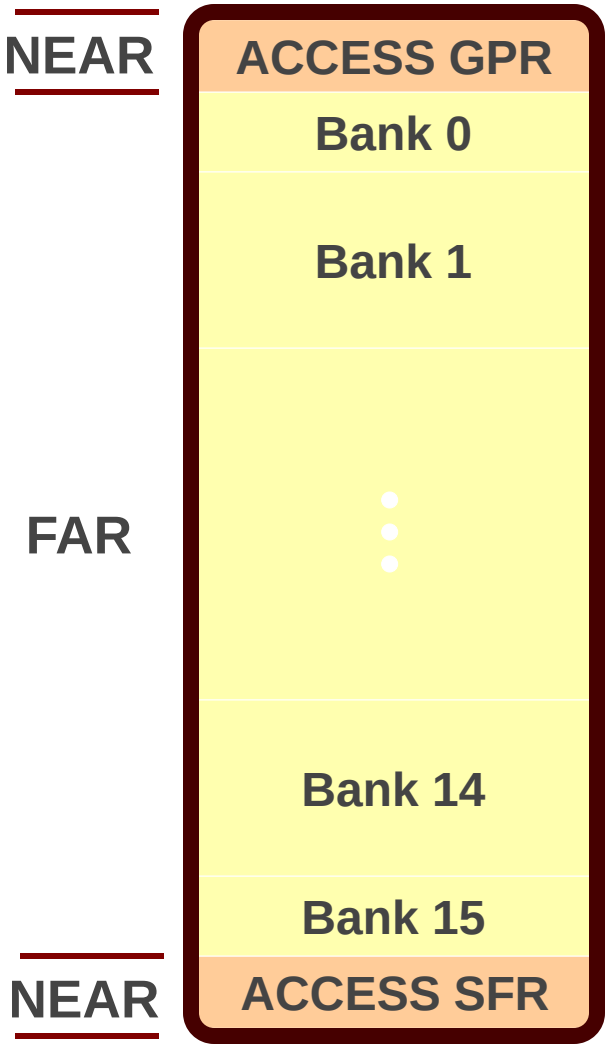
- Specifies that a variable *may* be located in far memory (non-Access RAM)
- Generates bank switching instructions to set the BSR before accessing the variable

Example

```
far int x;
far char a[10];
```

Locating Variables

Manually Optimizing Data Memory Use



- **Use small data model**
- **Let linker place most variables in the Access bank**
- **Define infrequently accessed variables with `far` qualifier**
  - Makes room in access bank for frequently accessed variables

Locating Variables

Variable Allocation Example

far Variables with the Small Data Model

```
#include <p18f4520.h>

char a[124];

#pragma idata myData
far char x = 0xAA;

int main(void)
{
    ...
}
```

1. **Variables are `near` and in Access Bank by default**
   - a[] fills Access bank along with 4 bytes claimed by `tempdata` and `MATH_DATA`
2.

- **To create far variables with the small data model:**
  - Declare a new section and give it some name
  - Define variables with the `far` qualifier

Locating Variables

How to place a variable in Flash

Syntax

```
rom type identifier;
```

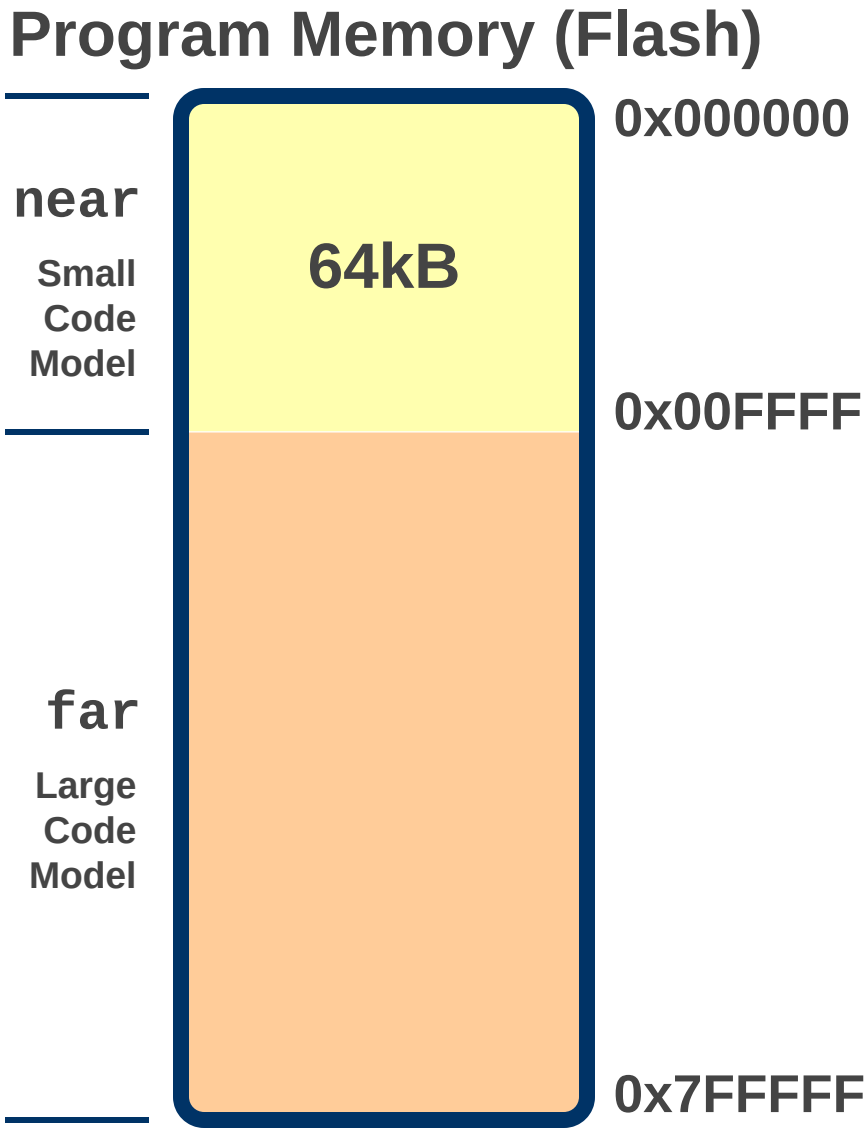
- Specifies that a variable should be located in program memory
- Uses TBLRD to read variable
- Effectively becomes `const` though writes may be done with extra code using TBLWT

Example

```
rom int x = 320;
rom char a[10] = "abcedfghi";
```

Locating Variables

How to place a variable in Flash



- **`near` and `far` also apply to `rom` variables**
- **`near rom`:**
  - Variable in first 64kB
  - ROM Pointers are 16-bit
- **`far rom`:**
  - Variable anywhere
  - ROM Pointers are 24-bit

Locating Variables

Variable Allocation Example

far Variables in Flash with the Small Code Model

```
#include <p18cxxx.h>

#pragma romdata myNear
rom int a = 0x5555;

#pragma romdata myFar=0x10000
far rom int x = 0xAAAA;

int main(void)
{
    ...
}
```

1. Variables are **near** and in first 64kB by default
2. To create **far** variables with the small code model:
   - Declare a new section and give it some name
   - Define variables with the `far rom` qualifier

Locating Variables

Variable Allocation Example

near Variables in Flash with the Large Code Model

```
#include <p18cxxx.h>

#pragma romdata myFar          1
rom int a = 0x5555;

#pragma romdata myNear=0x100
near rom int x = 0xAAAA;       2

int main(void)
{
    ...
}
```

1. **Variables are `far` and located anywhere by default**
2. **To create `near` variables with the large code model:**
   - Declare a new section and give it some name
   - Define variables with the `near rom` qualifier

Locating Variables

Variable Allocation Example

Allocating Variables in Different Regions of Memory (large model)

```
#pragma udata access myUAccess
near char w;
near char a[5];

#pragma idata access myIAccess
near char x=0xAA;
near char b[] = "Microchip";

#pragma udata
char y;
char c[5];

#pragma idata
char z = 0x55;
char d[] = "PIC MCUs";

rom char e[] = "Hello, world!";
```

Place uninitialized variables in Access bank

Don't generate bank switching instructions

Place initialized variables in Access bank

Don't generate bank switching instructions

Place uninitialized variables anywhere

Place initialized variables anywhere

Place variable in program memory

Locating Variables

How to place a variable in EEPROM data memory

Syntax

```
#pragma romdata eedata_scn = 0xF00000
rom identifier = value;
```

- Specifies that a variable should be located in EEPROM data memory
- May be used to initialize EEPROM data
- EEPROM variables may not be read/written like ordinary variables…

Example

```
#pragma romdata eedata_scn = 0xF00000
rom char eedata_values[4] = {0x01, 0x02, 0x03, 0x04};
rom int xee = 0x1234;
```

Locating Variables

How to place a variable in EEPROM data memory

Reading variables stored in EEPROM data memory

```
#include <p18f4520.h>

#pragma romdata eedata_scn = 0xF00000
rom char xee = 0xAA;

#pragma udata
char x;

int main(void)
{
    EECON1bits.EEPGD = 0;   // Access EEPROM data memory
    EEADR = (int)&xee;      // Load address of data to read
    EECON1bits.RD = 1;      // Start read operation
    x = EEDATA;             // Result in EEDATA register
}
```

Locating Variables

How to place a variable in EEPROM data memory

Writing variables stored in EEPROM data memory

```
#include <p18f4520.h>

#pragma romdata eedata_scn = 0xF00000
rom char xee;

int main(void)
{
    EECON1bits.EEPGD = 0;   // Access EEPROM data memory
    EECON1bits.WREN = 1;    // Enable writes
    EEADR = (int)&xee;      // Load address to write to
    EEDATA = 0x99;          // Data to write in EEDATA
    EECON2 = 0x55;          // Unlock sequence
    EECON2 = 0xAA;
    EECON1bits.WR = 1;      // Start write operation
    while (!PIR2bits.EEIF)  // Wait for write complete
    ;                       //  before next write
    PIR2bits.EEIF = 0;      // Clear EEIF flag
}
```

Interrupts must be disabled for unlock sequence

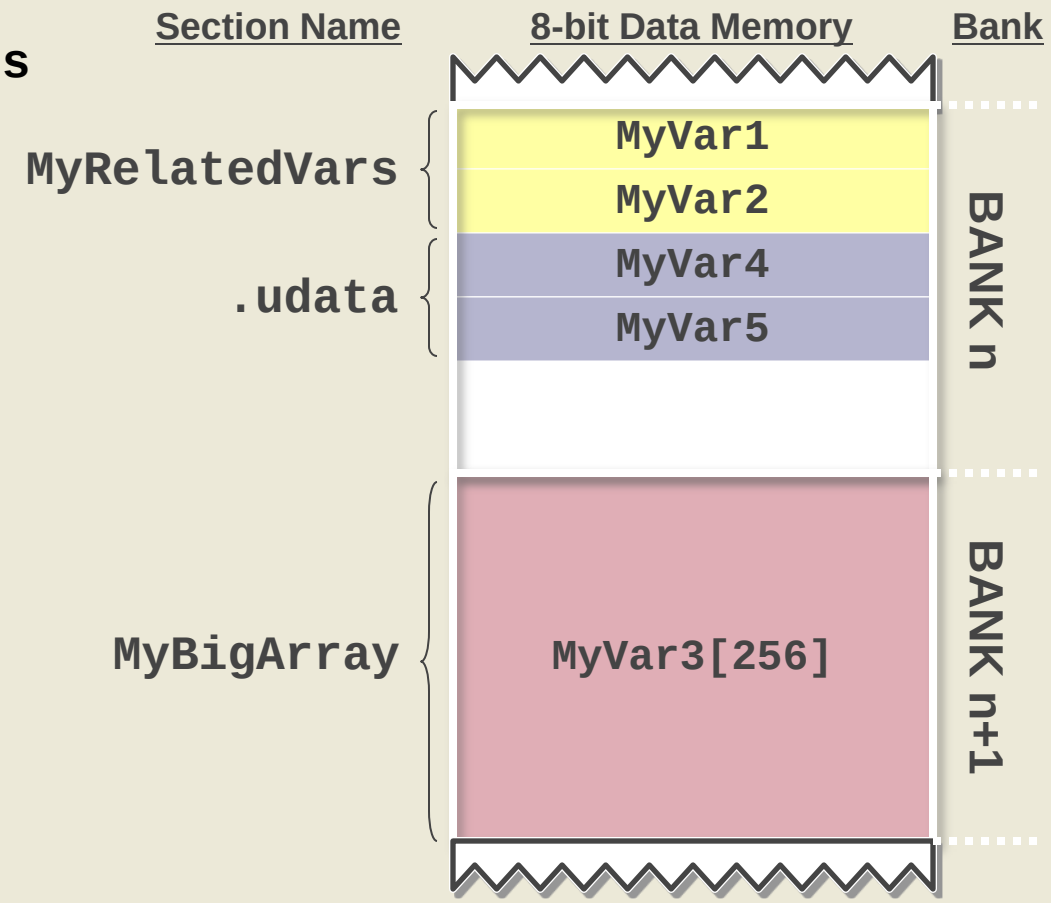
Locating Variables

How to put related variables in the same bank

Hand Optimization of Variable Allocation

Global Variable Declarations in Program

```
#pragma udata MyRelatedVars
unsigned char MyVar1;
unsigned char MyVar2;

#pragma udata MyBigArray
unsigned char MyVar3[256];

#pragma udata
unsigned char MyVar4;
unsigned char MyVar5;

#pragma code
//Function Prototypes and
//Function Definitions
//Go Here
```



Locating Variables

How to place a variable at a specific address

Syntax

```
#pragma sectiontype sectionname = address
```

- Specifies the address where a variable should be located
- Use sparingly – program will be harder to optimize by linker
- Won't work if ***sectiontype*** is wrong

Example

```
#pragma udata xSection = 0x100
far int x;
```

Locating Code

How to place code at a specific address

Syntax

```
#pragma code sectionname = address
```

- Specifies the address where code should be located
- Use sparingly – program will be harder to optimize by linker

Example

```
#pragma code myCode = 0x2000
int foo(void) { … }
```

MICROCHIP
Regional Training Centers

Resources

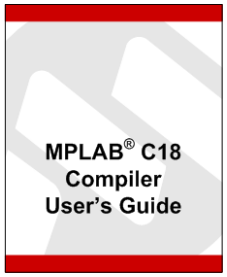
Microchip PIC18 material

- **PICmicro® 18C MCU Family Reference Manual (DS39500A)**
- ***PIC18F4520 Data Sheet* (DS39631E)**
- ***PICDEM2 Plus Users Guide* (DS51275D)**
- ***More than 100 App notes, design guides and other reference material!***

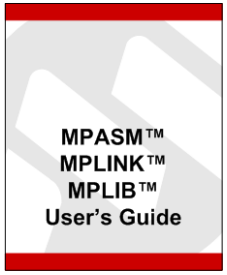
www.microchip.com

Resources

Books – Compiler Specific



- **MPLAB® C18 C Compiler User's Guide**
  Current Edition (PDF)
  Microchip Technology
  DS51288J
  http://www.microchip.com

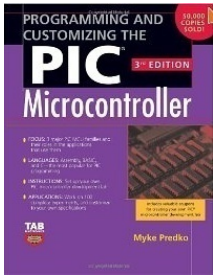


- **MPASM™ MPLINK™ and MPLIB™ User's Guide**
  Current Edition (PDF)
  Microchip Technology
  DS33014K
  http://www.microchip.com

The older books on C are much more relevant to embedded C programming since they were written back when PCs and other computers had limited resources and programmers had to manage them carefully.

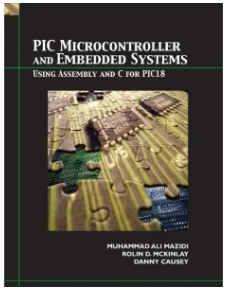
Reference Books

Books on PIC18



- **Programming and Customizing PIC Microcontrollers**
  3rd Edition (September 27, 2007)
  Myke Predko
  ISBN-10: 0071472876
  ISBN-13: 987-0071472876

- **Embedded Design with the PIC18F452 Microcontroller**
  1th Edition (August 15, 2002)
  John B. Peatman
  ISBN-10: 0130462138
  ISBN-13: 978-0130462138



- **PIC microcontrollers and Embedded Systems**
  4th Edition (October 19, 2006)
  Muhammad Ali Mazidi , Rolin McKinlay, Danny Causey
  ISBN-10: 0131194045
  ISBN-13: 978-0131194045

MICROCHIP

Regional Training Centers

Thank You

MICROCHIP
Regional Training Centers

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KeeLoq, KeeLoq logo, MPLAB, PIC, PICmicro, PICSTART, PIC^{32} logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.