MICROCHIP

Regional Training Centers

Microchip XC8

for 8-bit PIC Family

XC8 參考資料手冊

- **C:\Program Files (x86)\Microchip\xc8\v1.21\docs**
  - XC8 編譯器使用手冊
    - MPLAB_XC8_C_Compiler_User_Guide.pdf
  - XC8 周邊函數庫使用手冊
    - MPLAB_XC8_Peripheral_Libraries.pdf
- **C18 轉換至 XC8 參考手冊**
  - C18 to XC8 Migration Guide 50002184A.pdf
- **XC8 註冊參考手冊 (中文)**
  - XC8 Online Activation.pdf

MICROCHIP

Regional Training Centers

8-bit PIC 基本架構

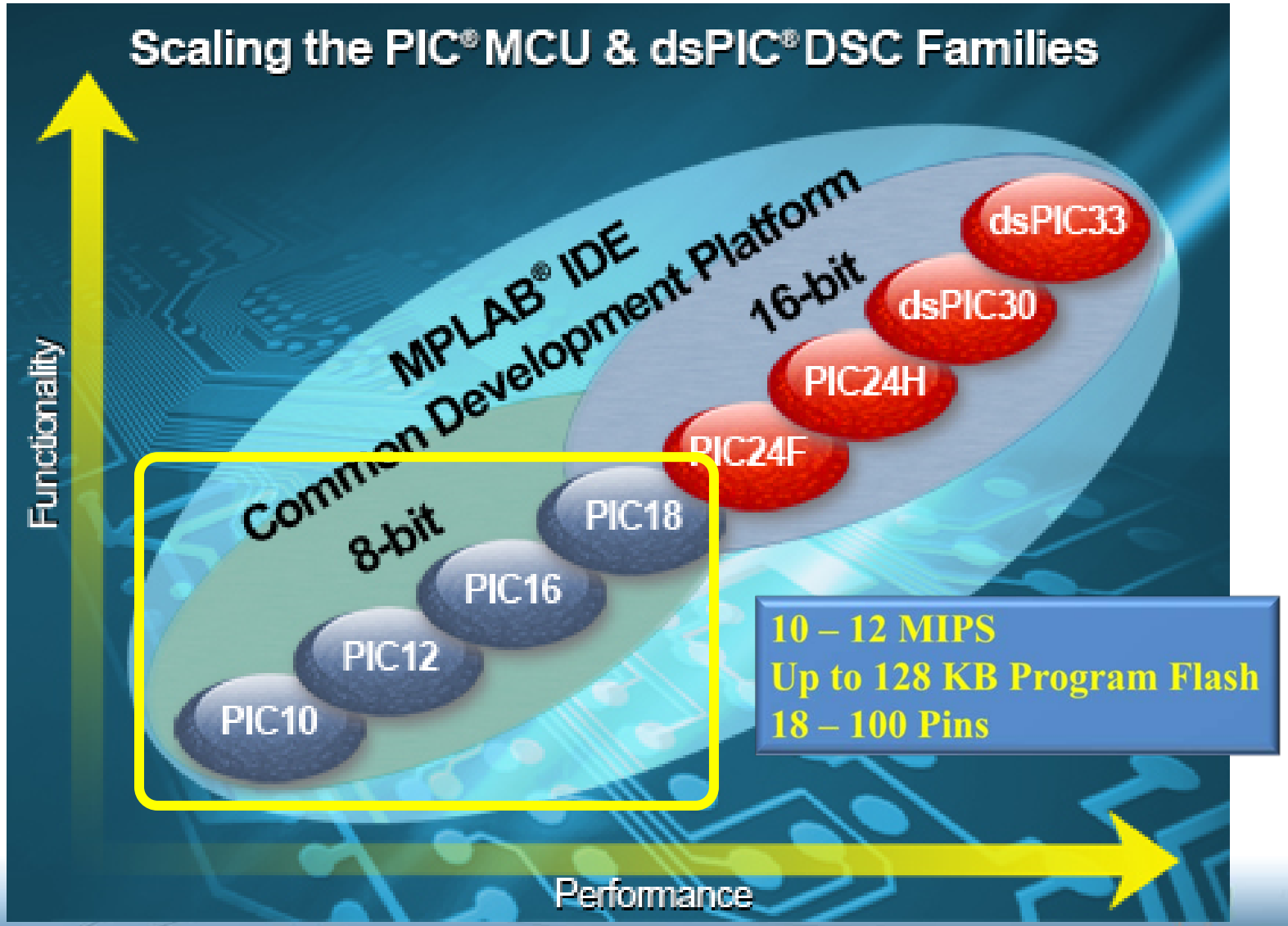
PIC16F1xxxx

PIC18Fxxxx

PIC18FxxJxx

PIC18FxxKxx

XC8 所支援的家族



PIC16 新、舊架構比較

舊 PIC16Fxxx

High-Performance RISC CPU:

Only 35 instructions
- 所有指令執行時間為一個指令週期，除了”分歧“指令

Operating speed:
- DC – **20MHz** oscillator/clock input
- DC – **200ns** instruction cycle

具中斷能力

8 層硬體中斷堆疊

直接，間接及相對定址模式

新 PIC16F1xxx

High-Performance RISC CPU:

Only 49 instructions
- 所有指令執行時間為一個指令週期，除了”分歧“指令以外

Operating speed:
- DC – **32 MHz** oscillator/clock input
- DC – **125 ns** instruction cycle (8MHz)

具有背景自動儲存能力的中斷功能

16 層硬體中斷堆疊，內建堆疊溢位/借位的 **Reset** 功能

直接，間接及相對定址模式：
- 兩個可定址到 **16-bit FSRs** 暫存器
- **FSRs** 可讀取程式及資料空間

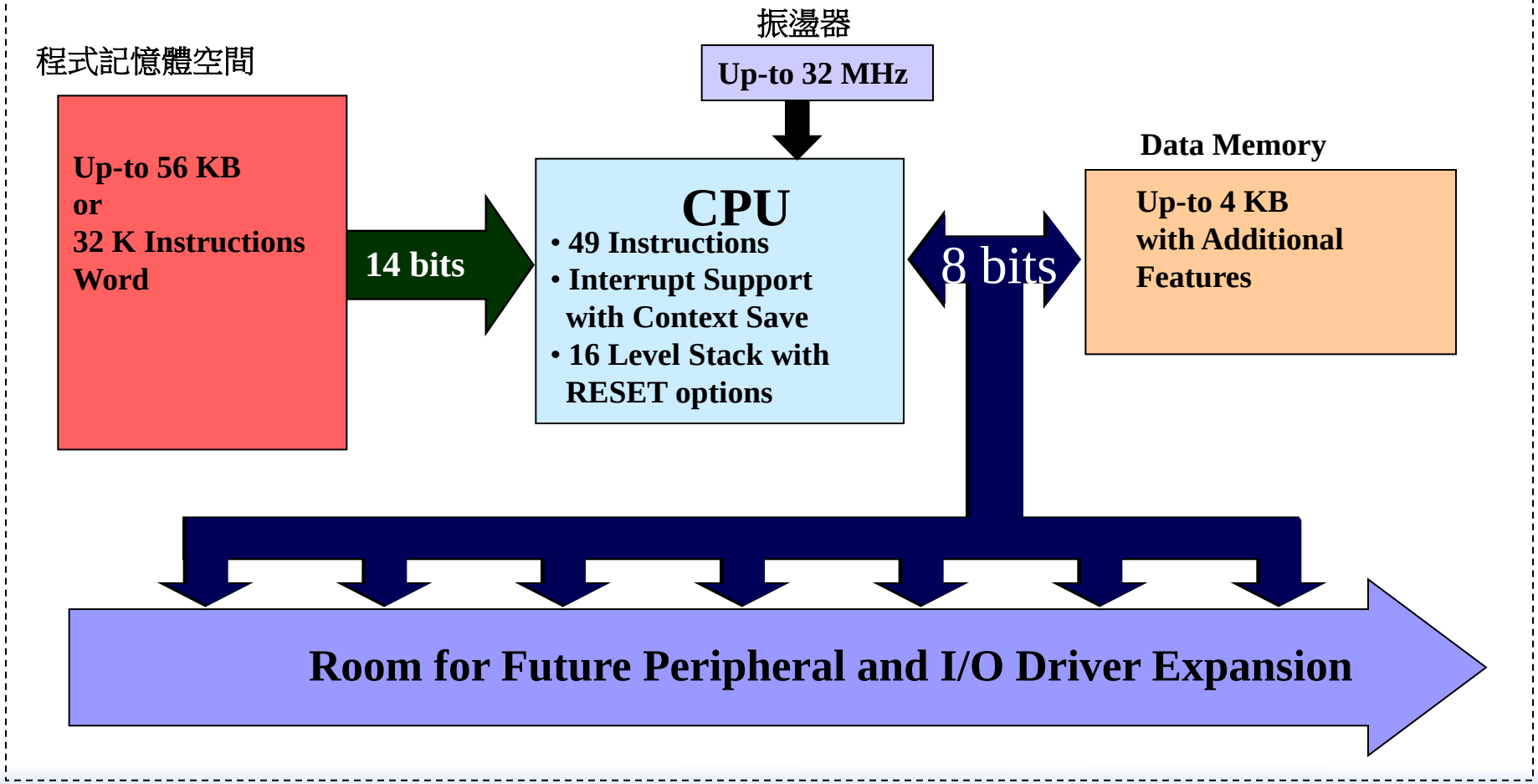
MICROCHIP
Regional Training Centers

8-bit PIC 快速比較表

	PIC16F	Enhanced PIC16F1	PIC18F
Max GPR/SFR	336 / 110	2496 / 316	4096/159+ 有些周邊較多的 PIC18F 元件，部分的周邊並沒有放在 access bank.
Max Program	8Kx14	32Kx14	1Mx16
FSRs	1	2 可以透過 FSR 讀取程式記憶體	3
Instruction Count	35	49	75 加入擴展型指令共有 85 個指令
Stack	8	16 with over/under flow reset	31 with over/under flow reset
Interrupts	1	1 hardware context save	2 optional hardware context save
Program Memory Read	透過 RETLW 指令讀取，有些元件可以透過 EEPROM 周邊介面讀取 (PIC16F877)	透過 RETLW 指令讀取、也可以透過 EEPROM 周邊介面讀取或使用 FSR 定址方式讀取	所有元件須透過 TABLRD instructions.

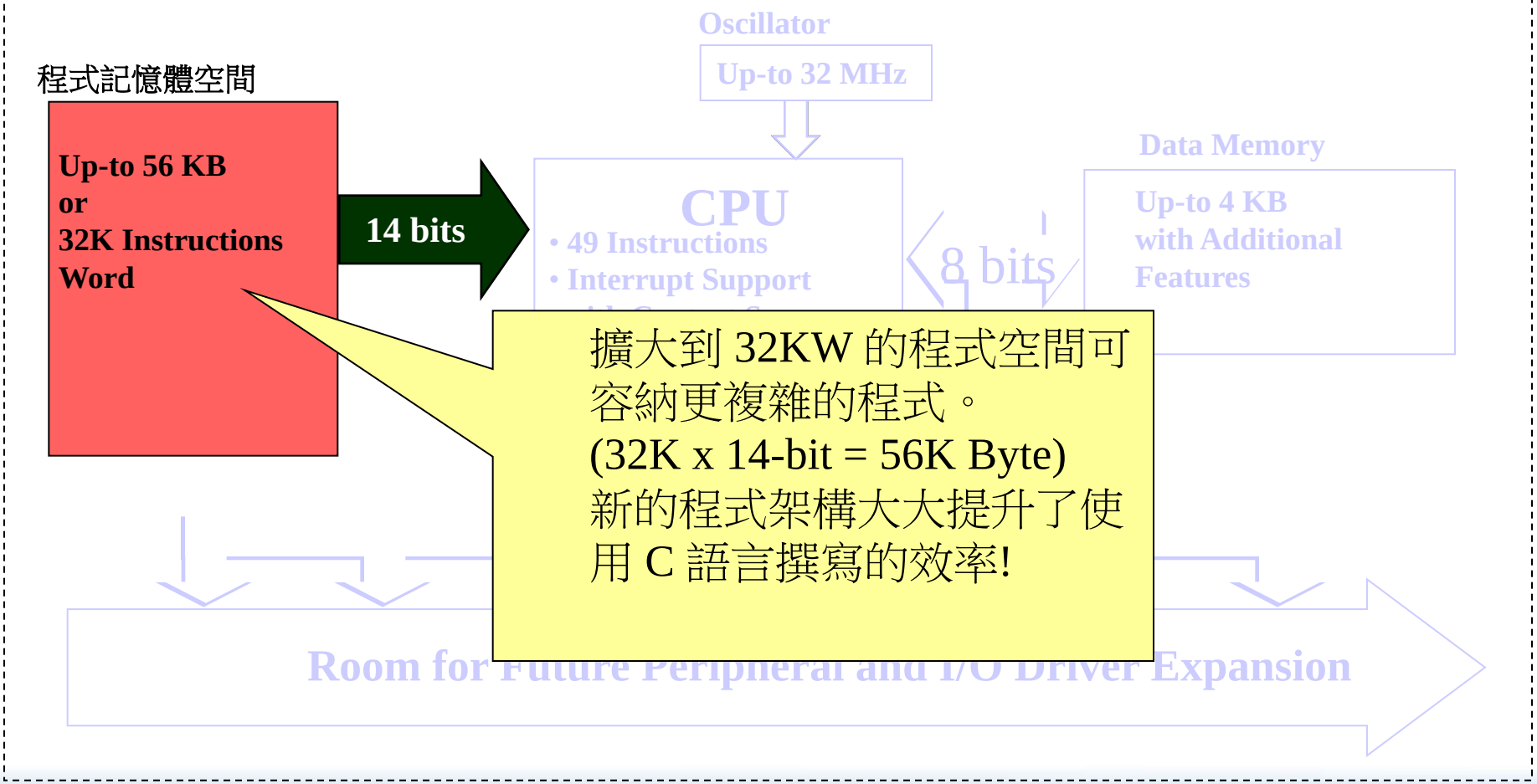
Enhanced PIC16F1xxx 基本架構

Enhanced Mid-Range PIC® Microcontroller



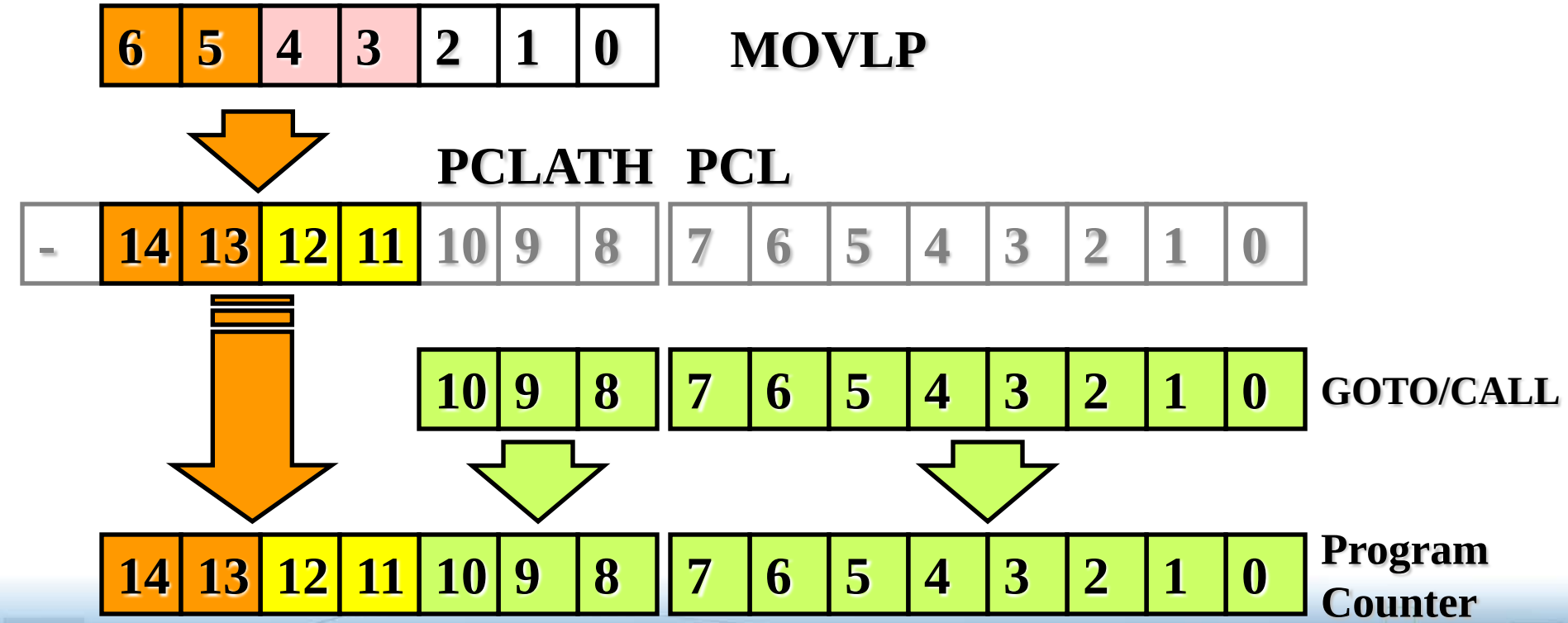
PIC16F1系列容量更大的程式空間 (Flash Memory)

Enhanced Mid-Range PIC® Microcontroller



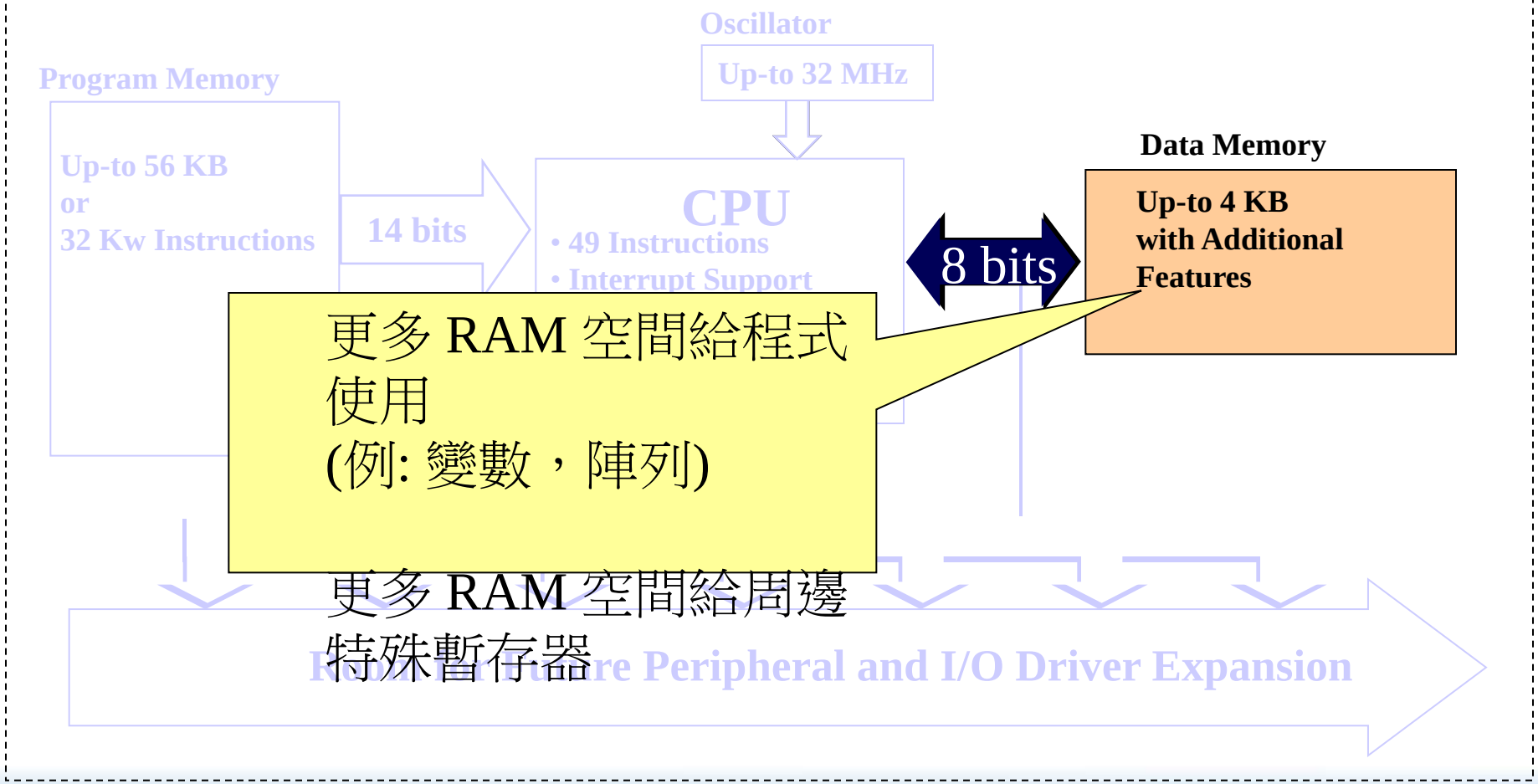
程式可定址到 32KW 的程式空間 (PIC16F1系列)

- 程式空間擴充到 16 個程式頁 (page) ，每頁受限於 goto/call (11-bit 位址長度) 指令仍維持 2kw 的大小。
- 使用 MOVLP 指令來簡化程式頁 (Page) 的切換 (舊的使用 PAGESEL)。



更多的資料記憶空間 (RAM & SFRs)

Enhanced Mid-Range PIC® Microcontroller



新的資料記憶體配置 (PIC16F1系列 RAM)

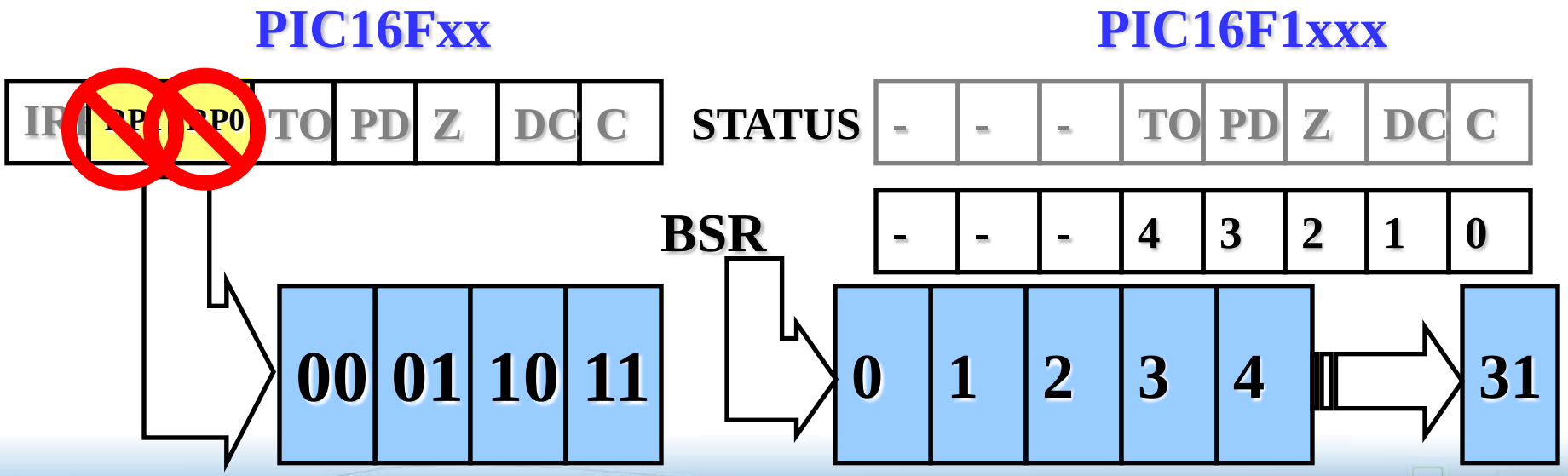
- 總共 **32 banks** 資料記憶空間
  - 每個 **Bank** 一樣為 **128Bytes**
  - 目前保留 **15 banks** 給以後新元件使用
- 更簡單使用的 **RAM**：
  - 保留每個 **Bank** 的最初的 **12 bytes RAM** 空間做為 **CPU** 共用 **Core SFR. (0x00 ~ 0x0B)**
  - **SFRs** 及周邊佔用每個 **Bank 20 bytes RAM** 空間 **(0x0C ~ 0x1F)**
  - 每個 **Bank** 最後 **16 bytes RAM** 為共用 **RAM (0x70 ~ 0x7F)**
- 新加入的功能
  - **W** 為 **RAM (0x09)** 的一員稱之為 **“w reg” (**與 **PIC18** 一樣**)**
  - **Banks 16-30** 給以後新元件使用
  - **Bank 31** 給堆疊及除錯使用

PIC16F1 RAM 的配置

	Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	◄·····►	Bank 31
0x000 0x00B	**12 Common CORE SFRs**							
0x00C 0x01F	SFRs 20	SFRs 20	SFRs 20	SFRs 20	SFRs 20	SFRs 20		Bank 31 Special Functions Stack Access & Debugging Registers
0x020 0x06F	GPR 80 Bytes	GPR 80 Bytes	GPR 80 Bytes	GPR 80 Bytes	GPR 80 Bytes	GPR 80 Bytes	Banks 6-30	
0x070 0x07F	**Common Memory (16 bytes)**							

有關 RAM Bank 的切換
直接定址模式

- 在舊的 **PIC16， Bank** 的切換是透過 **Status** 暫存器的 **RP0**及**RP1** 位元
- 在新的 **PIC16F1** 架構下，這兩個位元已經不存在了
- 新架構下，使用 **BSR** 暫存器來管理 **Bank** 的切換
- 使用新的 **MOVLB** 指令來做 **Bank** 的選擇

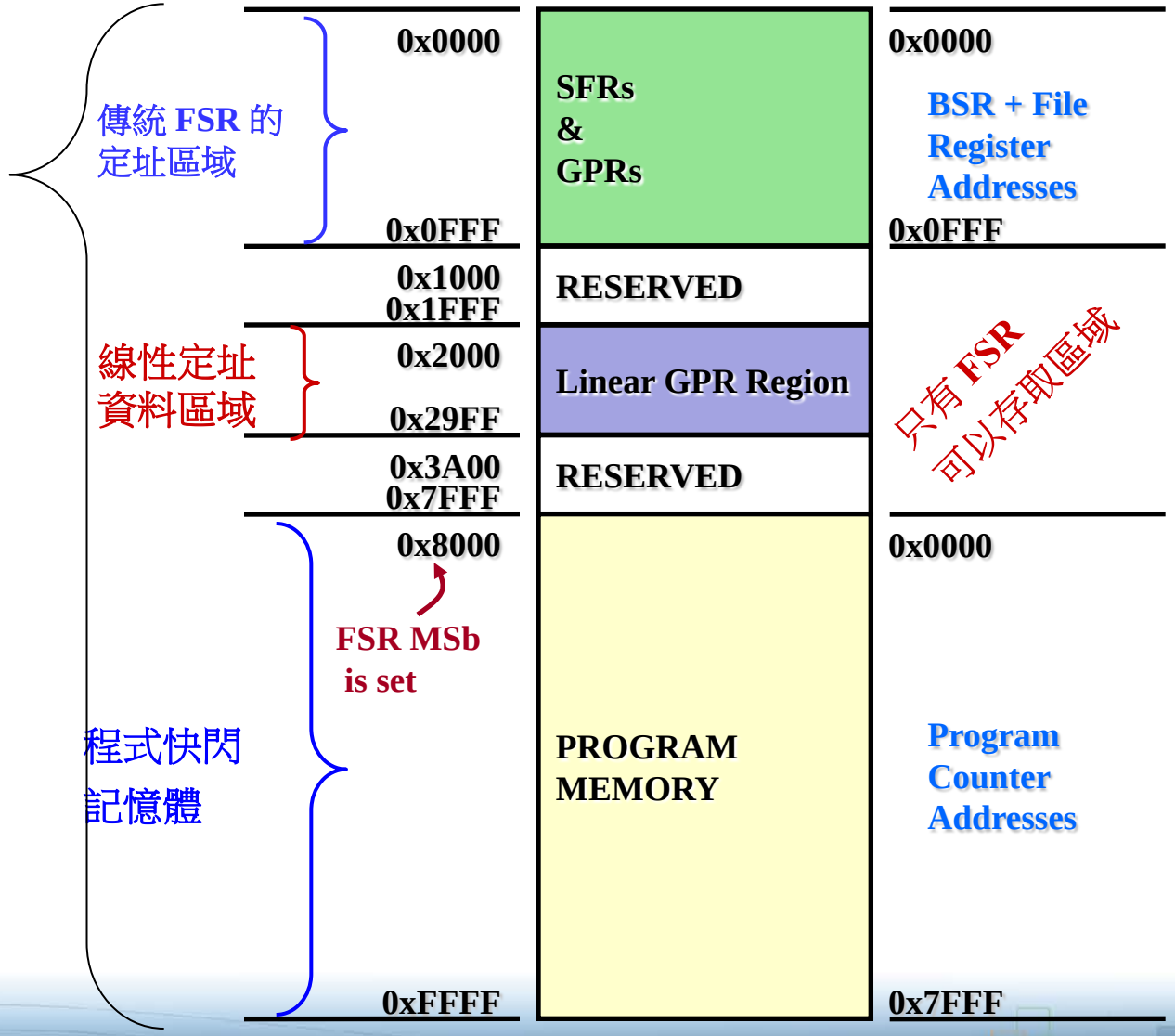


PIC16F1 間接定址模式 (FSR)

- **IRP bit** 已經不存在了，用 **FSRxH** 取代了
- 可以直接存取超過 **256 bytes**以上的資料，只要更新 **FSRxH** 暫存器
- 透過 **W** 暫存器的寫入，可以更快速有效的修改 **FSRxH** 暫存器
- 提供更有效率的查表指令
- **BANKSEL** 仍可繼續使用，直接修改 **FSRxH** 暫存器

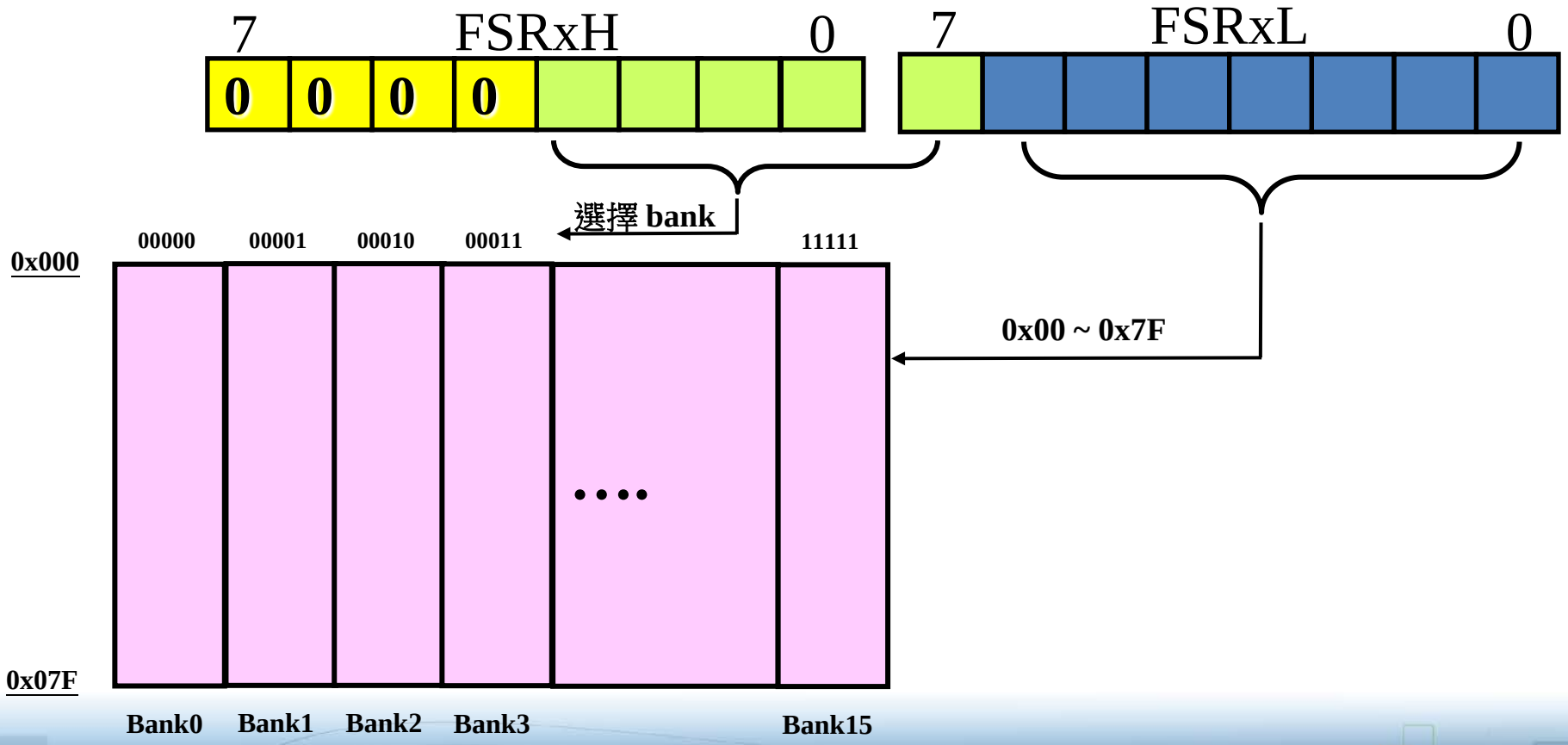
PIC16F1 新架構下的 FSRs

- 兩組 16-bit FSRs
- 視野可達 64K 全範圍
  - 傳統定址資料區域
  - 線性定址資料區域
  - 程式快閃記憶體
- FSR 最高位元
  - = 0，RAM 空間
  - = 1，程式空間
- FSRs 有更多的功能支援以方便查表的應用



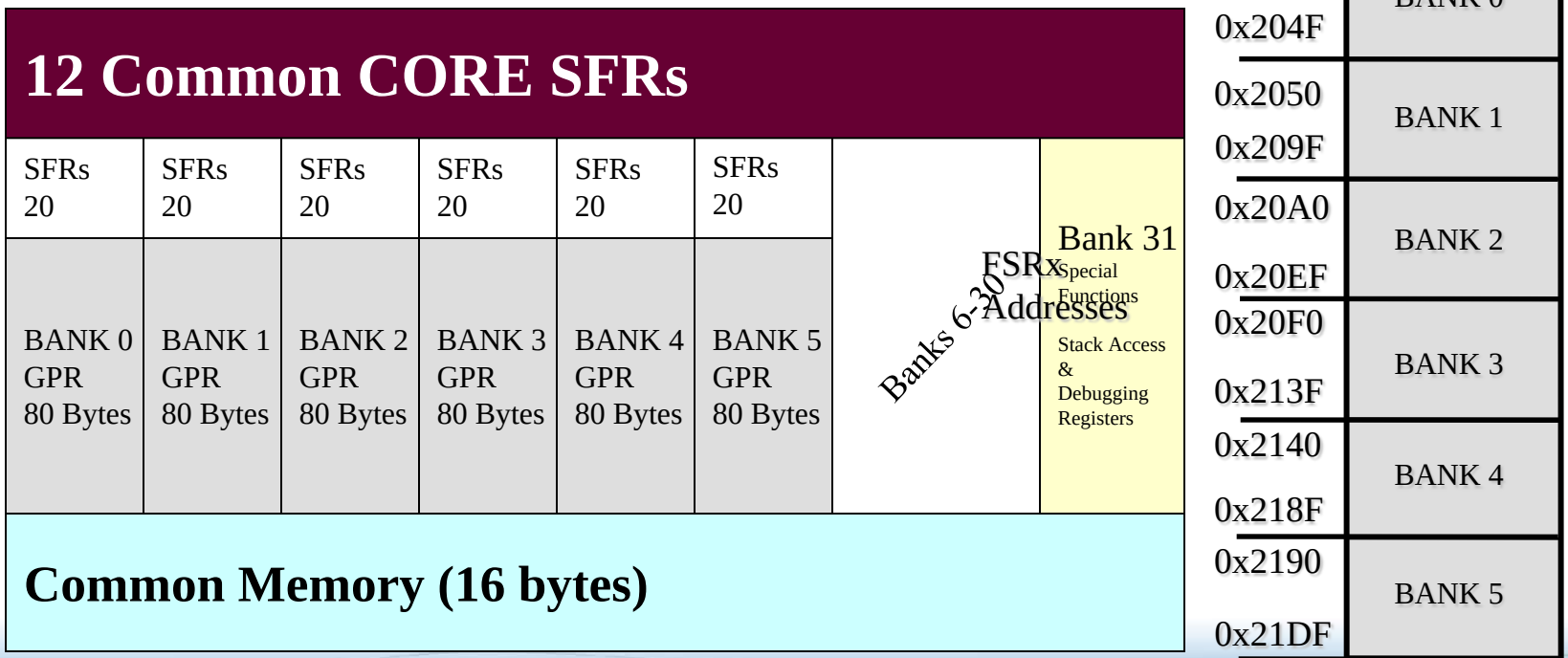
FSR 傳統定址

- 直接定址到 **RAM (0x0000 ~ 0x0FFF)**
  - 因為每個 **banks SFR** 佔用最前面的位址，所以傳統的索引還是有 **bank** 設定問題

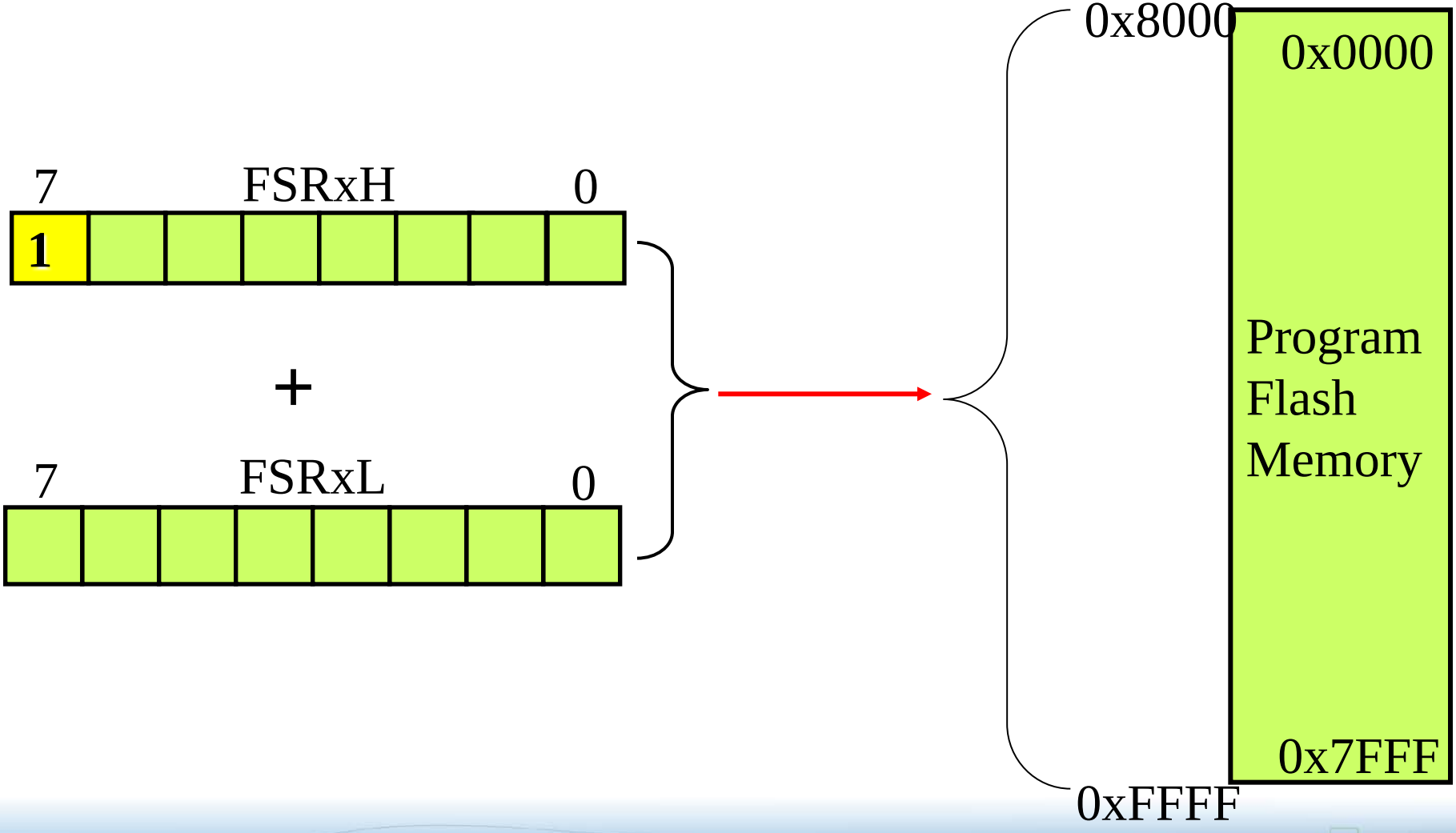


PIC16F1 FSR 資料區線性定址

- 整合每個**BANK**裡 **80Bytes RAM** 為線性定址區域
- **FSRx** 線性定址起始位址 **0x2000 ~ 0x29AF (BANK31** 不列入**)**
- 允許使用較大的資料堆疊、陣列、暫存區 **…** 等
- 不需切 **BANK** 直接**,** 透過 **FSRx** 就可以存取
- 定址範例 **: unsigned char InputBuffer[100] @ 0x2000;**

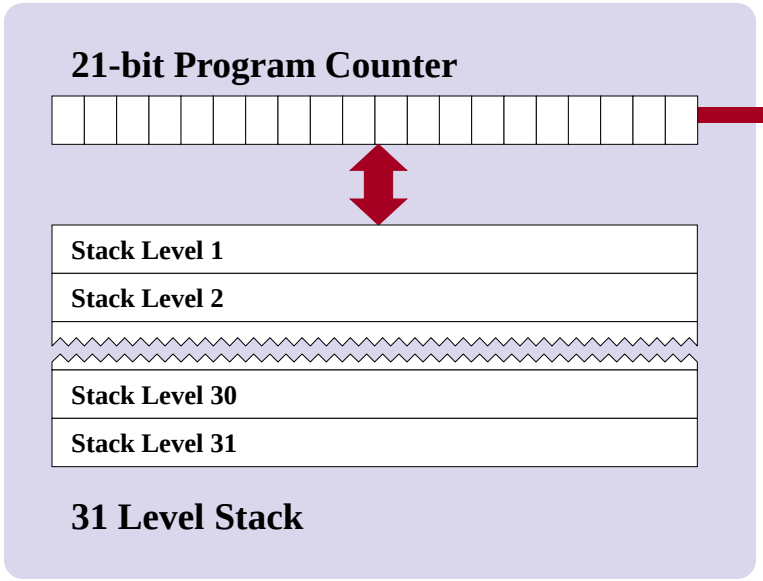


PIC16F1 FSR 程式記憶線性定址



程式記憶體的配置
PIC18 Program Memory Architecture

- 連續線性定址的程式記憶空間
- 最大可達 **2MB (1M Words)**



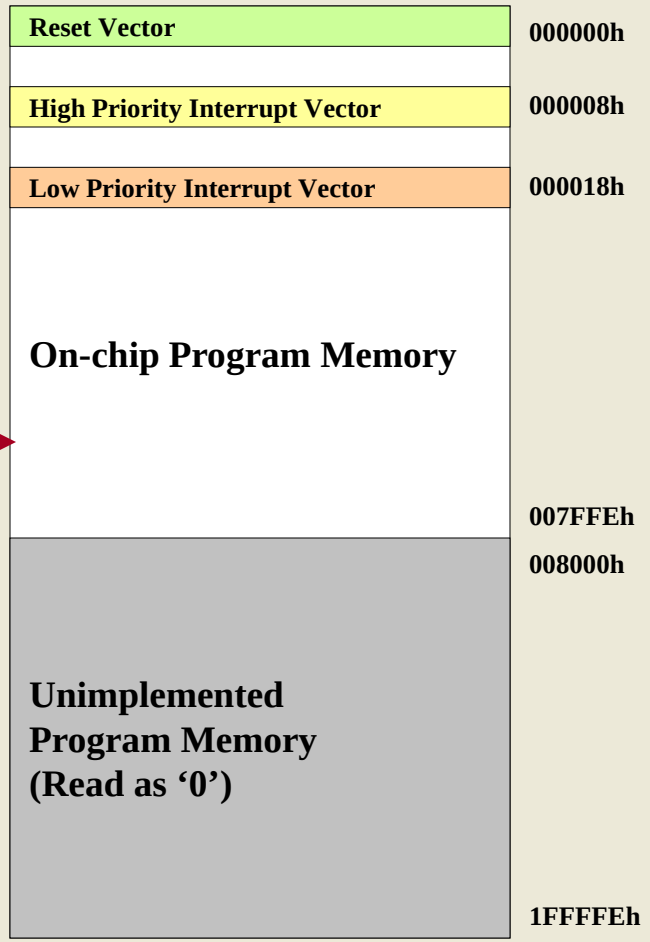
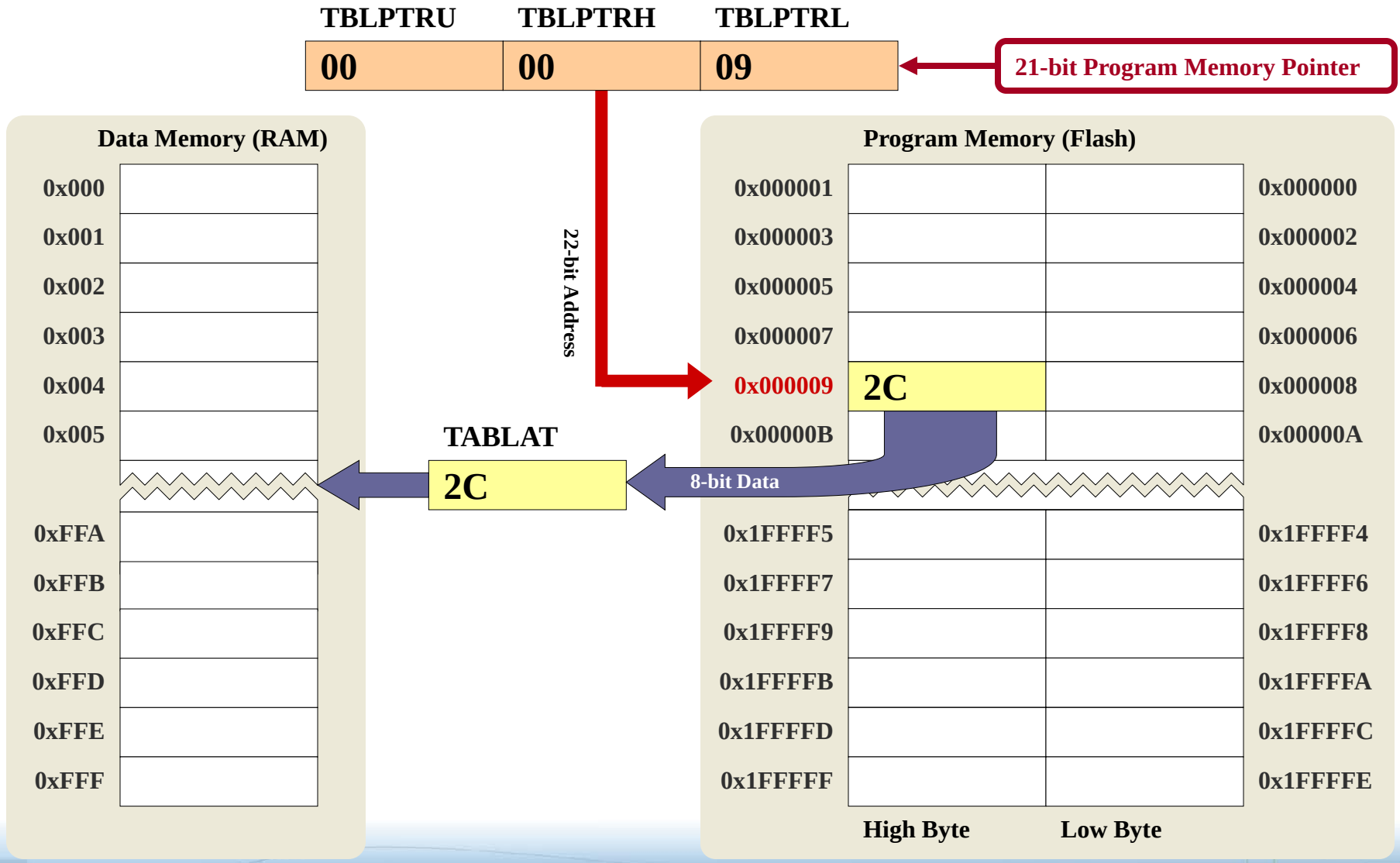


Table Read
8-bit PIC® Architecture



MICROCHIP

Regional Training Centers

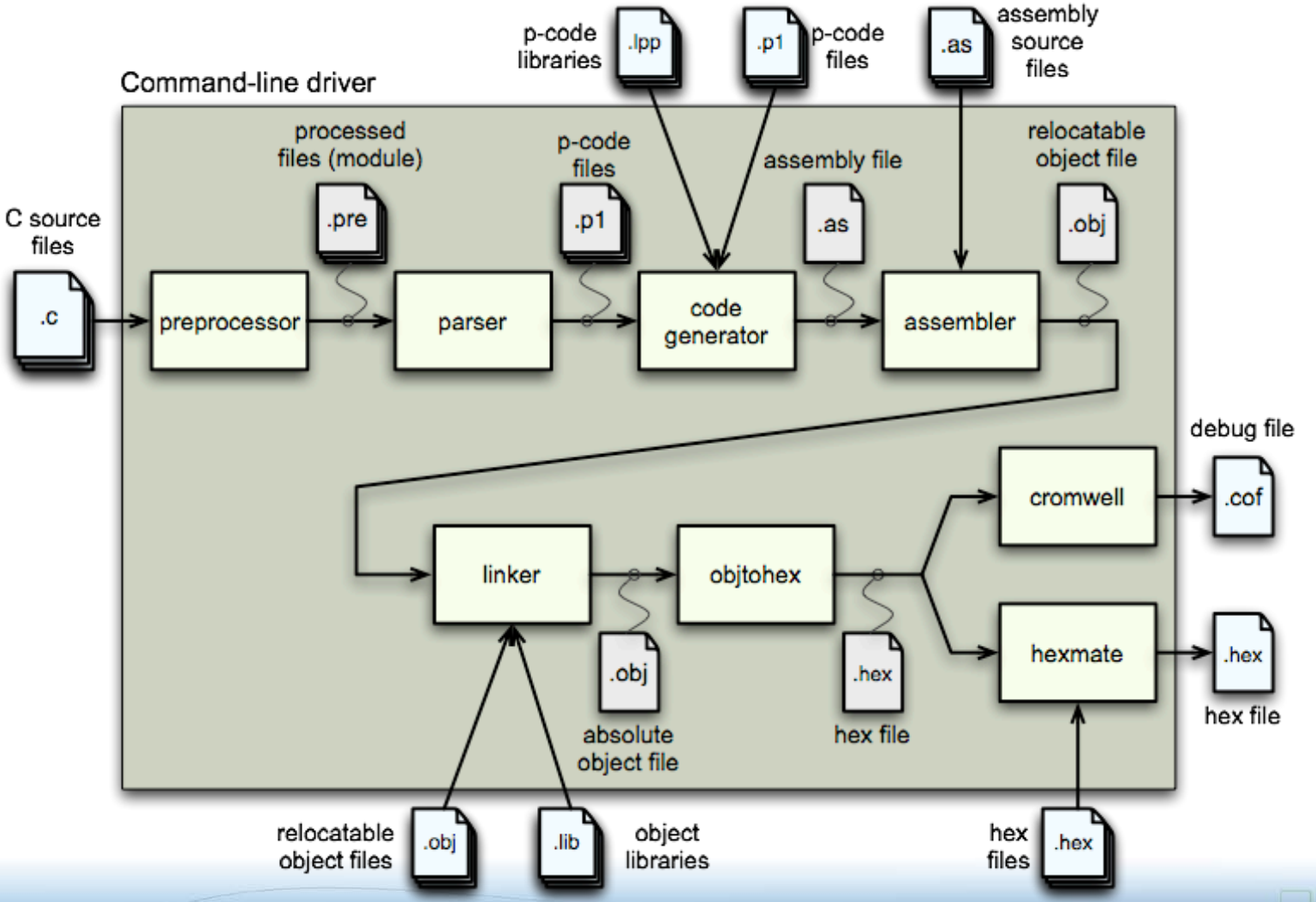
XC8 基本認識

C18 與 XC8 編譯程式

編譯階段	使用 C18 編譯	使用 XC8 編譯
C 編譯階段	MCC8.exe	XC8.exe
連結階段	MPLINK.exe	XC8.exe
建立函式資料庫	MPLIB.exe	XC8.exe

XC8.exe 接收來自 IDE 的命令列後決定要使用 PICC.exe (for PIC10/12/16) 或 PICC18.exe (for PIC18F) 來編譯，並決定所有的編譯動作。

一般標準編譯動作流程



XC 編譯器版本

- **Pro / Standard mode (授權版)**
  - 全功能的 Omniscient Code Generation (OCG) 最佳化功能
- **Lite mode (免費版)**
  - 限制性的 OCG 最佳化功能
  - 不受限於記憶體大小、元件編號及使用時間的限制
  - 原始程式相容於 OCG 的語法
- **評估版 45 天內使用為 PRO 模式，超過時間則轉為 Lite 模式**

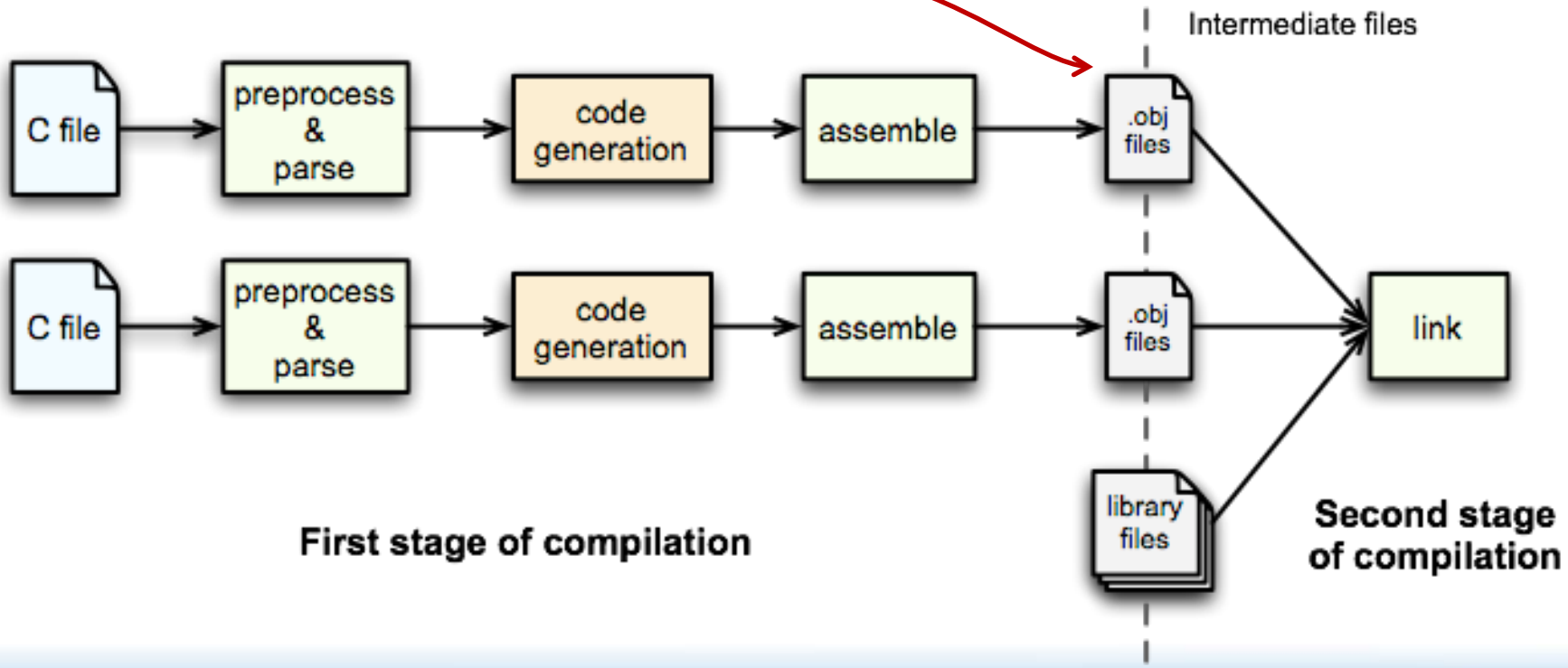
Microchip XC
OCG Technology

- 所有 **Microchip** 的 **XC** 編譯器採用新的專利 **Omniscient Code Generation™ (OCG)** 全域性程式編譯技術。
- **OCG** 具有全視野範圍的編譯架構
- 在正式編譯前先掃描過所有 **C** 檔案的關聯
- 強化最佳化編譯功能，縮減程式碼的空間，加快執行的速度

編譯的動作流程

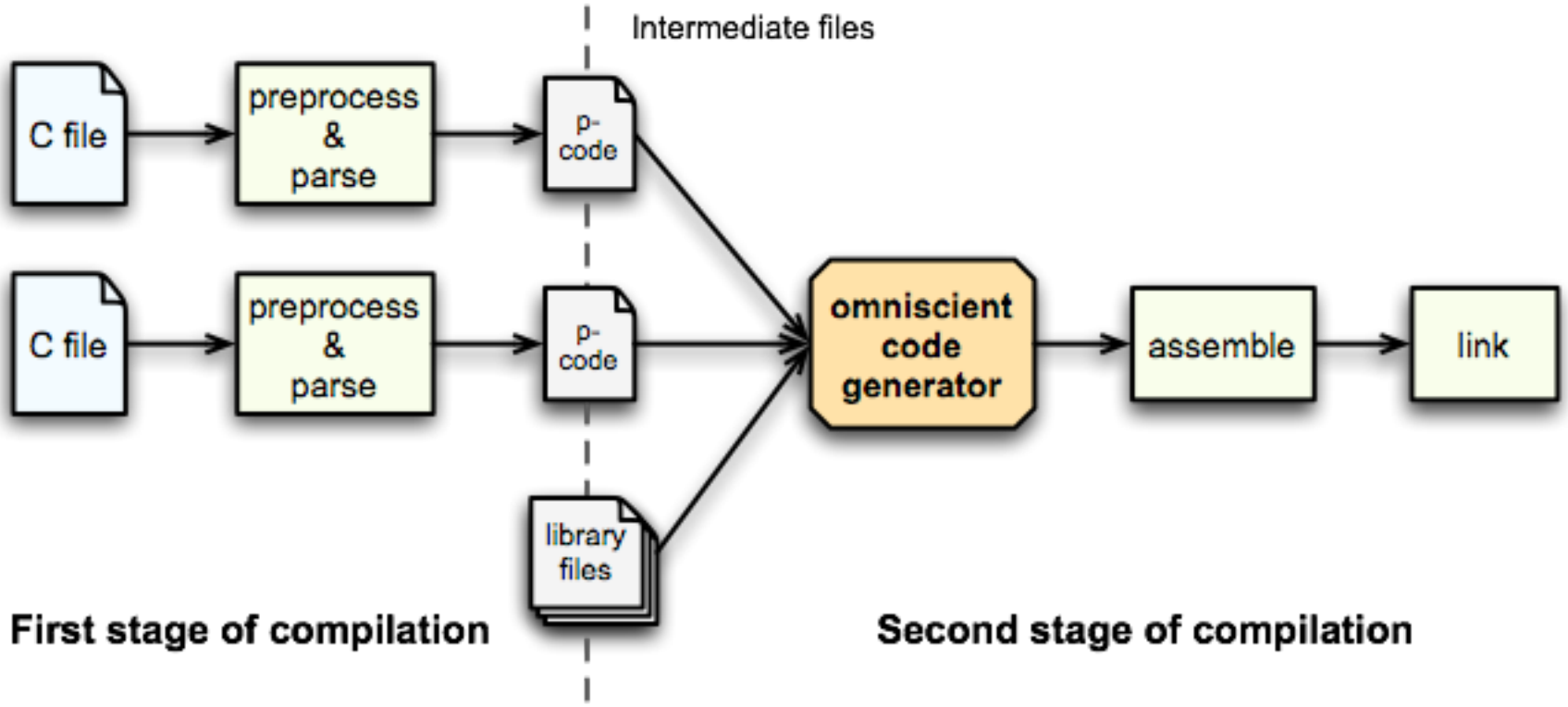
傳統方式編譯

- C 檔案是個別編譯處理的
  - 編譯出的中間檔案是要經連結器做連結處理的obj 檔



編譯的動作流程
全域性編譯方式 (OCG)

- 所有的 C 程式的編譯是同時處理的
  - P-code 檔是中間檔案，一起送入 OCG 編譯



XC8 整數的資料型別

Type	Bits	Min	Max
bit	1	0	1
char, unsigned char	8	0	255
signed char	8	-128	127
short, signed short	16	-32768	32767
unsigned short	16	0	65535
int, signed int	16	-32768	32767
unsigned int	16	0	65535
short long, signed short long	24	-8388608	8388607
unsigned short long	24	0	16777215
long, signed long	32	-2^{31}	$2^{31}-1$
unsigned long	32	0	$2^{32}-1$

浮點數的格式

- 支援兩種浮點數格式
  - 32-bit IEEE format

sign	exp	mantissa
x	xxxx xxxx	xxx xxxx xxxx xxxx xxxx xxxx

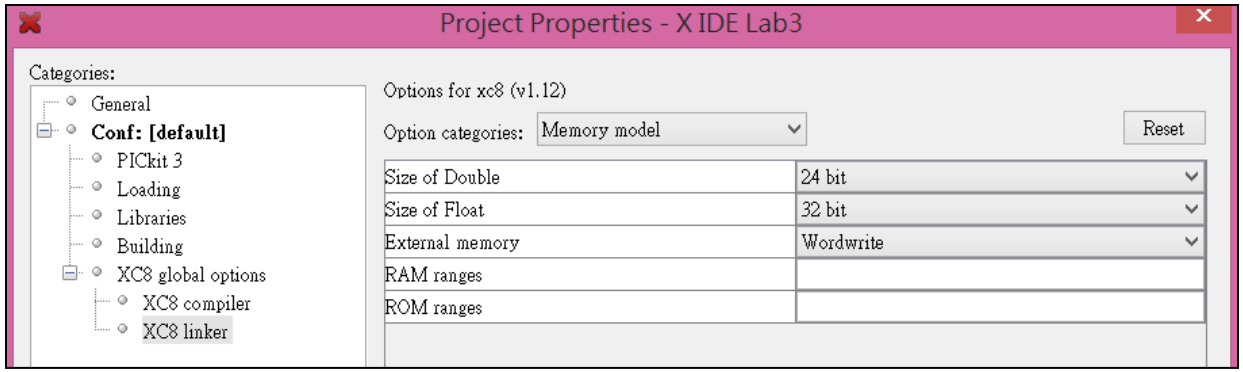
  - 24-bit truncated IEEE format (Default)

sign	exp	mantissa
x	xxxx xxxx	xxx xxxx xxxx xxxx

- **The larger format offers greater precision**

選擇 Float 的格式

- 在 Project Properties 下，選擇 XC8 Linker
- 選擇 Memory Model 後，修改浮點數格式



Type	Bits	Exp	Mant	Min	Max
float	24	8	15	~3.587324e-43 (00 00 01)	~3.402771E+38 (7f 7f ff)
double	24	8	15	~3.587324e-43 (00 00 01)	~3.402771E+38 (7f 7f ff)
	32	8	23	~1.401298E-45 (00 00 00 01)	~3.402823E+38 (7f 7f ff ff)

MICROCHIP

Regional Training Centers

Header File

了解 XC8 裡的標頭檔 (H Files) 的來龍去脈

標頭檔的路徑定義

- 編譯器所提供的標頭檔 **(Header File)** 的路徑搜尋方式
  - ..\xc8\v1.21\include\　　一般的標頭檔 (安裝時內定路徑)
  - ..\xc8\v1.21\include\plib\　　周邊函數庫的標頭檔

格式	內定的搜尋路徑
<xxx.h>	編譯器內定的標頭檔的目錄下
"xxx.h"	先搜尋原始檔的目錄； 再到編譯器設定的標頭檔的目錄 (如下頁所示)

通用標頭檔
Generic Header File (xc.h)
(第一層)

- **<xc.h>** 為第一個要加入的標頭檔
- **<xc.h>** 標頭檔須在各個程式模組加入此檔案
  - #include <xc.h>
- 舊版 **Hi-Tech C** 編譯器使用 **<htc.h>** 與使用 **<xc.h>** 是一樣的
  - #include <xc.h> 等於 #include <htc.h>
- 直接連結到下一層的 **htc.h**

htc.h
(第二層)

- **htc.h** 會檢查來自 **IDE** 所傳入的元件訊息來判斷是要使用:
  - PIC16Fxxxx 加入 <pic.h>
  - PIC18Fxxxx 加入 <pic18.h>

```
/* HI-TECH PICC / PICC-Lite compiler */
#if  defined(__PICC__) || defined(__PICCLITE__)
#include <pic.h>
#endif

/* HI-TECH PICC-18 compiler */
#if  defined(__PICC18__)
#include <pic18.h>
#endif
```

PIC16Fxxx 使用 pic.h
第三層

- **pic.h 主要功能**
  - #include <pic_chip_select.h> (下一層)
  - 定義巨集：
    - CLRWDT( ),SLEEP( ),NOP( ), RESET( )
  - 沿襲舊版的 Configuration Word 定義
  - 沿襲舊版的 ID Locations 定義
  - 沿襲舊版的 Internal EEPROM 定義

pic_chip_select.h
第四層

- **pic_chip_select.h 是很重要的一層**
  - 依據 IDE 所傳入的元件訊息找出所要使用的元件標頭檔
  - PIC16F1937 找出 <pic16f1937.h>
  - PIC16F887 找出 <pic16f887.h>

pic16F887.h
第五層

- **pic16f887.h 宣告該元件的所有暫存器**
  - 與 Data Sheet 相同的名稱
  - 暫存器的絕對位址
  - 使用 bit 定義暫存器裡的位元位址
  - 使用位元結構定義暫存器所屬的位元名稱
- **Configuration Word 的定義在 pic16f887.h 檔裡已移除。XC8 使用 PIC16Fxxxx 專用的 pic_chipinfo.html 做為 Configuration Bit 設定的參考。詳細設定會在後面說明。**

pic16f887.h 裡的定義 (SFRs)

```
// Register: PORTA
extern volatile unsigned char       PORTA          @ 0x005;
#ifndef _LIB_BUILD
asm("PORTA equ 05h");
#endif
// 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
typedef 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_t;
extern  volatile  PORTAbits_t   PORTAbits   @ 0x005;
```

定義 PORTA 的絕對位址

定義 PORTA 裡各個位元的絕對位址
(Hi-Tech PICC & XC8 專用語法)

定義 PORTA 的位元結構
(用法上與 C18 相容)

關於 PIC18Fxxxx 使用 pic18.h 第三層

- **pic18.h 主要功能**
  - #include <pic18_chip_select.h>
  - 定義巨集：
    - CLRWDT(), ClrWdt()
    - SLEEP(),Sleep()
    - NOP(), Nop()
    - RESET(),Reset()
  - 沿襲舊版的 Configuration Word 定義
  - 沿襲舊版的 Internal EEPROM 定義

pic18_chip_select.h
第四層

- **pic18_chip_select.h 是很重要的一層**
  - 依據 IDE 所傳入的元件訊息找出所要使用的元件標頭檔
  - PIC18F4520 找出 <pic18f4520.h>
  - PIC18F45K22 找出 <pic18f45k22.h>
- **pic18f4520.h 檔定義內部暫存器名稱，絕對位址與位元結構名稱。**
- **XC8 使用 pic18_chipinfo.html 做為 Configuration Bit 設定的參考。詳細設定會在後面說明。**

Hlink 所使用的元件連結描述檔

- 連結程式 **(hlink.exe)** 如何知道所選用的元件的訊息
  - PIC16Fxxx : picc.ini
  - PIC18Fxxxx : pic18-18.ini
- 修改 **ini** 的內容即改變該元件所預設的資源，如需修改需注意其內容。

PIC16F887 記憶容量及位址設定

..\microchip\xc8\v1.12\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 **PIC16F887** 為例，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
```

PIC18F4520 記憶容量及位址設定

..\microchip\xc8\v1.12\dat\picc-18.ini

底下以 **PIC18F4520** 為例，請參考 **picc-18.ini** 第一頁的說明

```
[18F4520]
ARCH=PIC18
CFGMEM=300000-30000D
COMMON=00-7F
DEVIDMEM=3FFFFE-3FFFFF
EEPROM=F00000-F000FF
ERRATA=FASTINTS
FAMILY=18f4520
FLASH_EW=40,20
GPRBANKS=080-0FF,100-1FF,200-2FF,300-3FF,400-4FF,500-5FF
ICD2RAM=5F4-5FF
ICD2ROM=7DC0-7FFF
INSTR=EXTENDED
MAKE=MICROCHIP
PLIB=1
PROCID=4520
RAMSIZE=600
REALICERAM=5EF-5FF,F9C-F9C,FD4-FD4,FDB-FDF,FE3-FE7,FEB-FEF,FFC-
FFF
REALICEROM=7D00-7FFF
ROMSIZE=8000
SFRBANKS=F80-FFF
STACKDEPTH=1F
USERIDMEM=200000-200007
```

MICROCHIP

Regional Training Centers

PIC Device Configuration

PIC 的工作配置需求的設定

設定 Configuration Bits

定義

Configuration Bits - 暫存器中的特殊控制位元，配合程式設計的需求。這些設定位元主控制著 MCU “永久” 功能的設定，只能在燒錄程式時以燒錄方式設定。

- **設定如下功能：**
  - 振盪器類型
  - 看門狗計時器
  - 掉電偵測
  - 程式保護 …
- **更多詳細資訊，請參見資料手冊中的 “PIC 的特殊功能暫存器”**

在 XC8 下設定 Config. 的語法

- 統一使用 **#pragma config** 的設定方式
  - 語法 #pragma config 設定項= 選擇項
  - 巨集定義在 <pic.h> , <pic18.h>
  - PIC16Fxxx 設定參考檔案：pic_chipinfo.html
  - PIC18Fxxxx 設定參考檔案：pic18_chipinfo.html

使用 #prgama config 的範例：

```
#pragma config FOSC=XT, BOREN=OFF, WRT= OFF, DEBUG=ON
```

取代舊的 **PICC** 語法 **__config(x)**

```
__CONFIG(XT & WDTDIS & BORDIS& WRTEN & DEBUGEN);
```

Configuration Bit 的巨集

16F877A

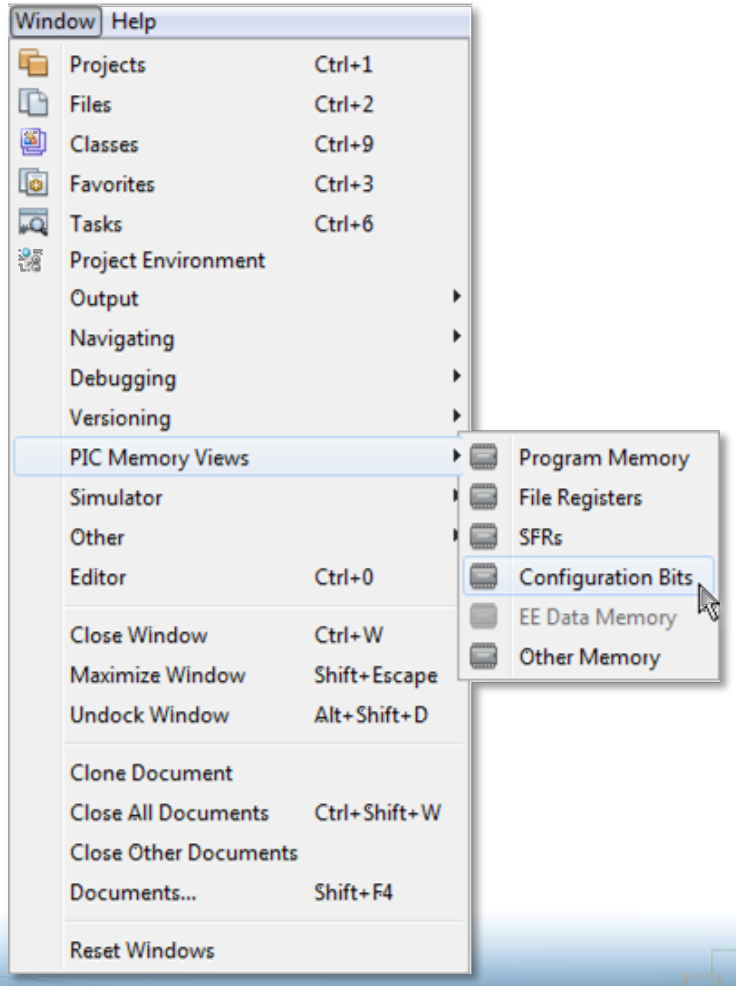
原 PICC 的巨集	XC 的巨集		功能說明
WDTEN /WDTDIS	WDTE=	ON/OFF	Watchdog timer
PWRTEN/PWRTDIS	PWRTE=	ON/OFF	Power up timer
BOREN /BORDIS	BOREN=	ON/OFF	Brown out reset
LVPEN/LVPDIS	LVP=	ON/OFF	Low voltage programming
DP / DPROT	CPD=	ON/OFF	Data code protected
PROTECT	CP=	ON	Program code protected
UNPROTECT		OFF	Program code unprotected
WRTEN	WRT=	OFF	Flash memory write DISABLED
WP1		256	Protect 00-FFh
WP2		1FOURTH	Protect 00-7FFh
RC	FOSC=	EXTRC	RC oscillator
HS		HS	High Speed crystal oscillator
XT		XT	Crystal oscillator
LP		LP	Low power crystal oscillator

如何查看 Configuration Bits
使用 MPLAB X IDE

1 打開 Configuration Bits（設定位元）視窗

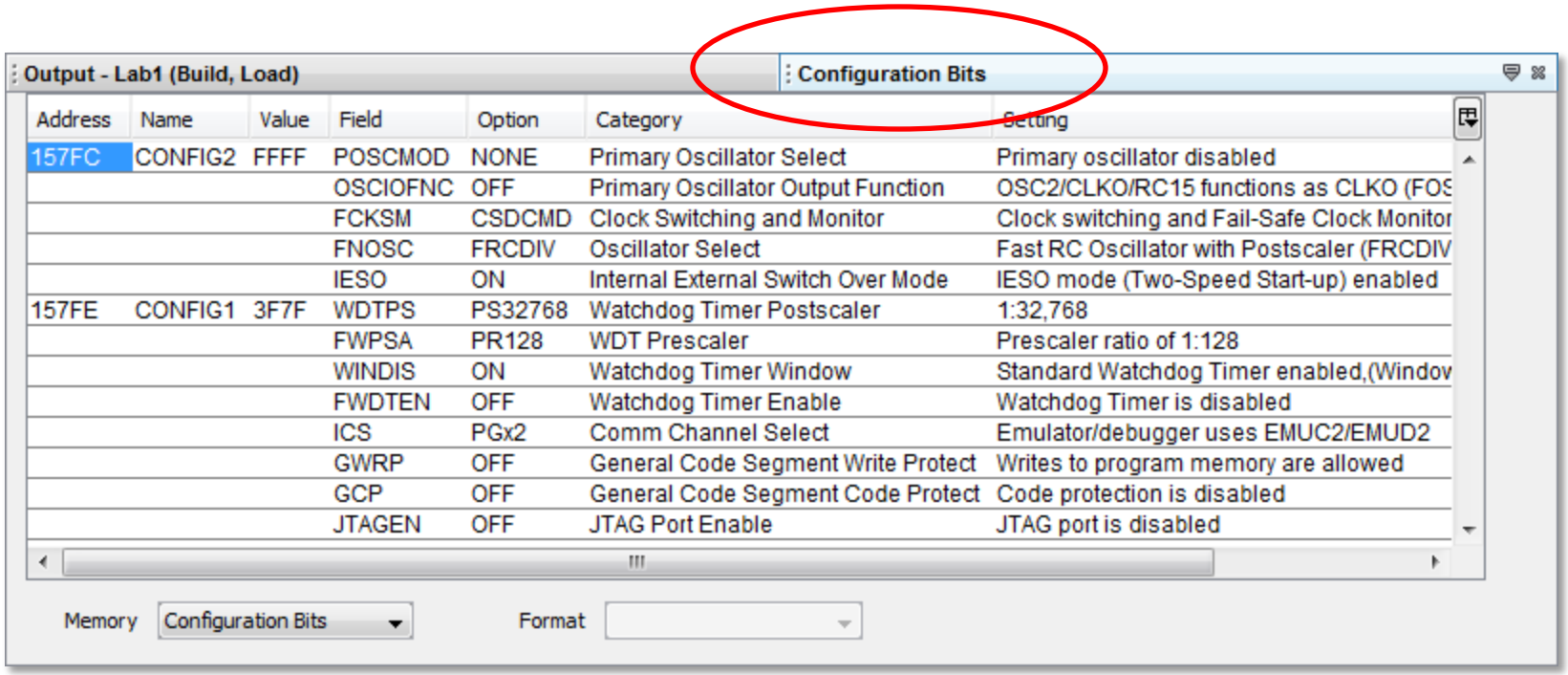
從主功能表中選擇：

- Window
- PIC Memory Views（PIC 記憶體視圖）
- Configuration Bits



如何查看 Configuration Bits

2 該視窗以選項卡的形式在 **Output** 視窗旁邊打開。



在程式裡設定 Configuration Bits

- **XC8 對於 Configuration Bits 的設定採用**
  - #pragma config <setting>=<named value>
  - 語法與 MPLAB C18 相同

- **XC8 對 Hi-Tech PICC (PIC16F) 所使用的 _ _config (x) 的語法也相容，希望改用新的語法**

XC8 的 Configuration Bits

- **PIC16F 系列**
  - 相關的定義在:
    - C:\Program Files\Microchip\xc8\vx.x\docs\pic_chipinfo.html

- **PIC18F 系列**
  - 相關的定義在:
    - C:\Program Files\Microchip\xc8\vx.x\docs\pic18_chipinfo.html

Config. Bits 元件的選擇

Microchip MPLAB XC8 C Compiler supported devices				
10F200	10F202	10F204	10F206	10F220
10F222	10F320	10F322	10LF320	10LF322
12C508	12C508A	12C509	12C509A	12C671
12C672	12CE518	12CE519	12CE673	12CE674
12CR509A	12F1501	12F1571	12F1572	12F1822
12F1840	12F508	12F509	12F510	12F519
12F520	12F529T39A	12F529T48A	12F609	12F615
12F617	12F629	12F635	12F675	12F683
12F752	12HV609	12HV615	12HV752	12LF1501
12LF1552	12LF1571	12LF1572	12LF1822	12LF1840
12LF1840T39A	12LF1840T48A	16C432	16C433	16C505
16C54	16C54A	16C54C	16C55	16C554
16C557	16C558	16C55A	16C56	16C56A
16C57	16C57C	16C58A	16C58B	16C620
16C620A	16C621	16C621A	16C622	16C622A
16C62A	16C62B	16C63	16C63A	16C642
16C64A	16C65A	16C65B	16C66	16C662

pic_chipinfo.html

Microchip MPLAB XC8 C Compiler supported devices				
18C242	18C252	18C442	18C452	18C601
18C658	18C801	18C858	18F1220	18F1230
18F1320	18F1330	18F13K22	18F13K50	18F14K22
18F14K22LIN	18F14K50	18F2220	18F2221	18F2320
18F2321	18F2331	18F23K20	18F23K22	18F2410
18F242	18F2420	18F2423	18F2431	18F2439
18F2450	18F2455	18F2458	18F248	18F2480
18F24J10	18F24J11	18F24J50	18F24K20	18F24K22
18F24K50	18F2510	18F2515	18F252	18F2520
18F2523	18F2525	18F2539	18F2550	18F2553
18F258	18F2580	18F2585	18F25J10	18F25J11
18F25J50	18F25K20	18F25K22	18F25K50	18F25K80
18F2610	18F2620	18F2680	18F2682	18F2685
18F26J11	18F26J13	18F26J50	18F26J53	18F26K20
18F26K22	18F26K80	18F27J13	18F27J53	18F4220
18F4221	18F4320	18F4321	18F4331	18F43K20
18F43K22	18F4410	18F442	18F4420	18F4423

pic18_chipinfo.html

pic18_chipinfo.html 裡的 PIC18F4520 Config.

#pragma config Settings

Register: CONFIG1H @ 0x300001

IESO =	**Internal/External Oscillator Switchover bit**	
	OFF	**Oscillator Switchover mode disabled**
	ON	**Oscillator Switchover mode enabled**
OSC =	**Oscillator Selection bits**	
	RCIO6	**External RC oscillator, port function on RA6**
	RC	**External RC oscillator, CLKO function on RA6**
	INTIO7	**Internal oscillator block, CLKO function on RA6, port function on RA7**
	XT	**XT oscillator**
	LP	**LP oscillator**
	HSPLL	**HS oscillator, PLL enabled (Clock Frequency = 4 x FOSC1)**
	ECIO6	**EC oscillator, port function on RA6**
	EC	**EC oscillator, CLKO function on RA6**
	INTIO67	**Internal oscillator block, port function on RA6 and RA7**
	HS	**HS oscillator**
FCMEN =	**Fail-Safe Clock Monitor Enable bit**	
	OFF	**Fail-Safe Clock Monitor disabled**
	ON	**Fail-Safe Clock Monitor enabled**

程式裡直接加入 Config. 的設定

- 如上頁所示的選擇項，使用 **#pragma config** 在程式裡做設定
- 範例：

```
#pragma config IESO = OFF, OSC = RCIO6, FCMEN = OFF
```

設定的範例說明:

IEOS = OFF： Internal/External Oscillator Switchover bit: Oscillator Switchover mode disabled

OSC = RCIO6： Oscillator Selection bits: External RC oscillator, port function on RA6

FCMEN = OFF : Fail-Safe Clock Monitor Enable bit: Fail-Safe Clock Monitor disabled

在 MPLAB X IDE 下使用 Generate Source Code to Output 功能自動產生 Config. 的設定程式

- 一般是直接在程式裡加入 **Configuration Bits** 的設定
- 在 **MPLAB X IDE** 下，可以先選好所需的 **Config.** 設定項後，使用 **“Generate Source Code to Output”** 的功能來產生設定所需的原始程式

如何自動產生 Config. 的設定程式
範例：PIC18F4520

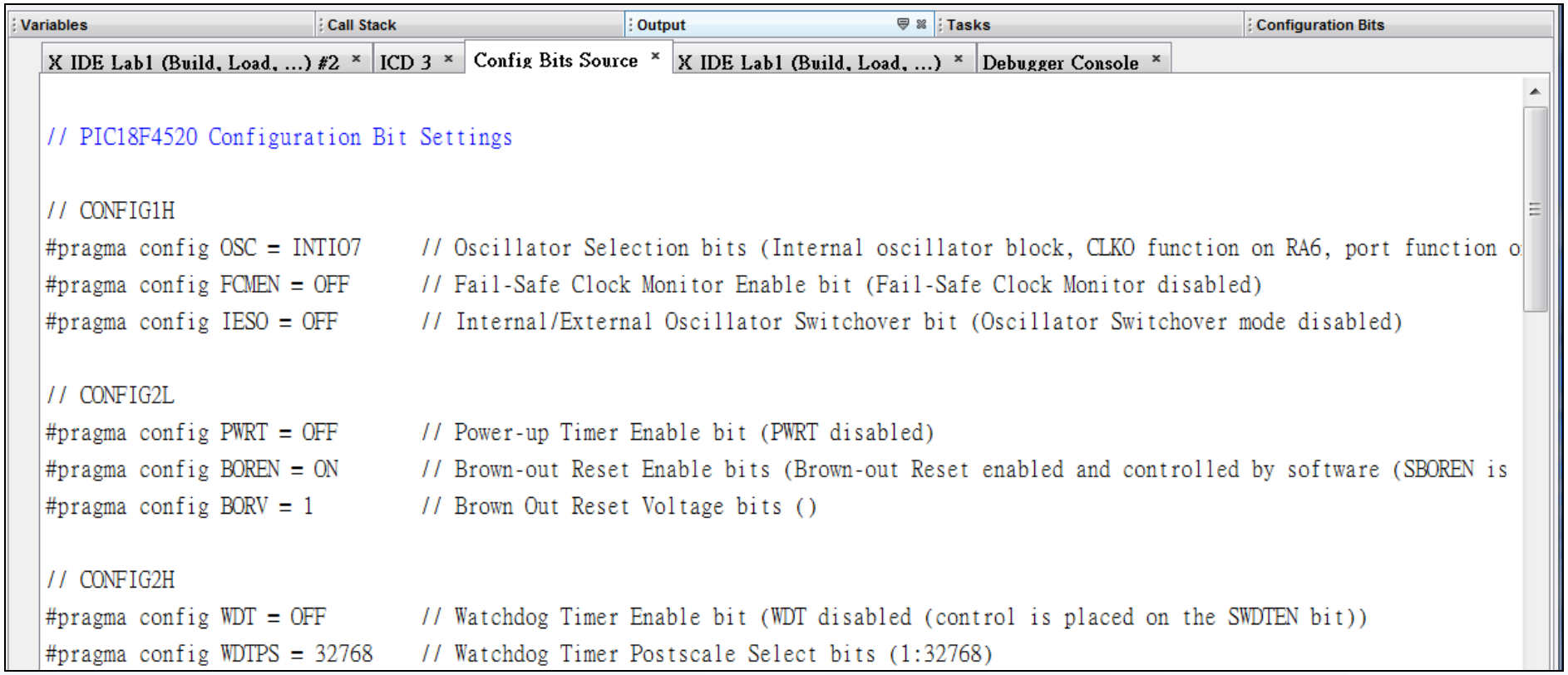
- 在 MPLAB® X IDE 下先手動修改設定
  - 在 Option 下，點選要更改的專案
  - 在下拉式功能表中選取所需的選擇項
  - 按一下按鈕 “Generate Source Code to Output”（產生設定 Config. 的 C 原始程式）

Address	Name	Value	Field	Option	Category	Setting
157FC	CONFIG2	FFFF	POSCMOD	NONE	Primary Oscillator Select	Primary oscillator disabled
			OSCIOFNC	OFF	Primary Oscillator Output Function	OSC2/CLKO/RC15 functions as CLKO (FOSC/2)
			FCKSM	CSDCMD	Clock Switching and Monitor	Clock switching and Fail-Safe Clock Monitor are disabled
			FNOSC	FRCDIV	Oscillator Select	Fast RC Oscillator with Postscaler (FRCDIV)
			IESO	ON	Internal External Switch Over Mode	IESO mode (Two-Speed Start-up) enabled
157FE	CONFIG1	7FFF	WDTPS	PS32768	Watchdog Timer Postscaler	1:32,768
			FWPSA	PR128	WDT Prescaler	Prescaler ratio of 1:128
			WINDIS	ON	Watchdog Timer Window	Standard Watchdog Timer enabled,(Windowed-mode is disabled)
			FWDTEN	ON	Watchdog Timer Enable	Watchdog Timer is enabled
			ICS	PGx2	Comm Channel Select	Emulator/debugger uses EMUC2/EMUD2
			GWRP	OFF	General Code Segment Write Protect	Writes to program memory are allowed
			GCP	OFF	General Code Segment Code Protect	Code protection is disabled
			JTAGEN	ON	JTAG Port Enable	JTAG port is enabled

Memory: Configuration Bits　Format: Read/Write　Generate Source Code to Output

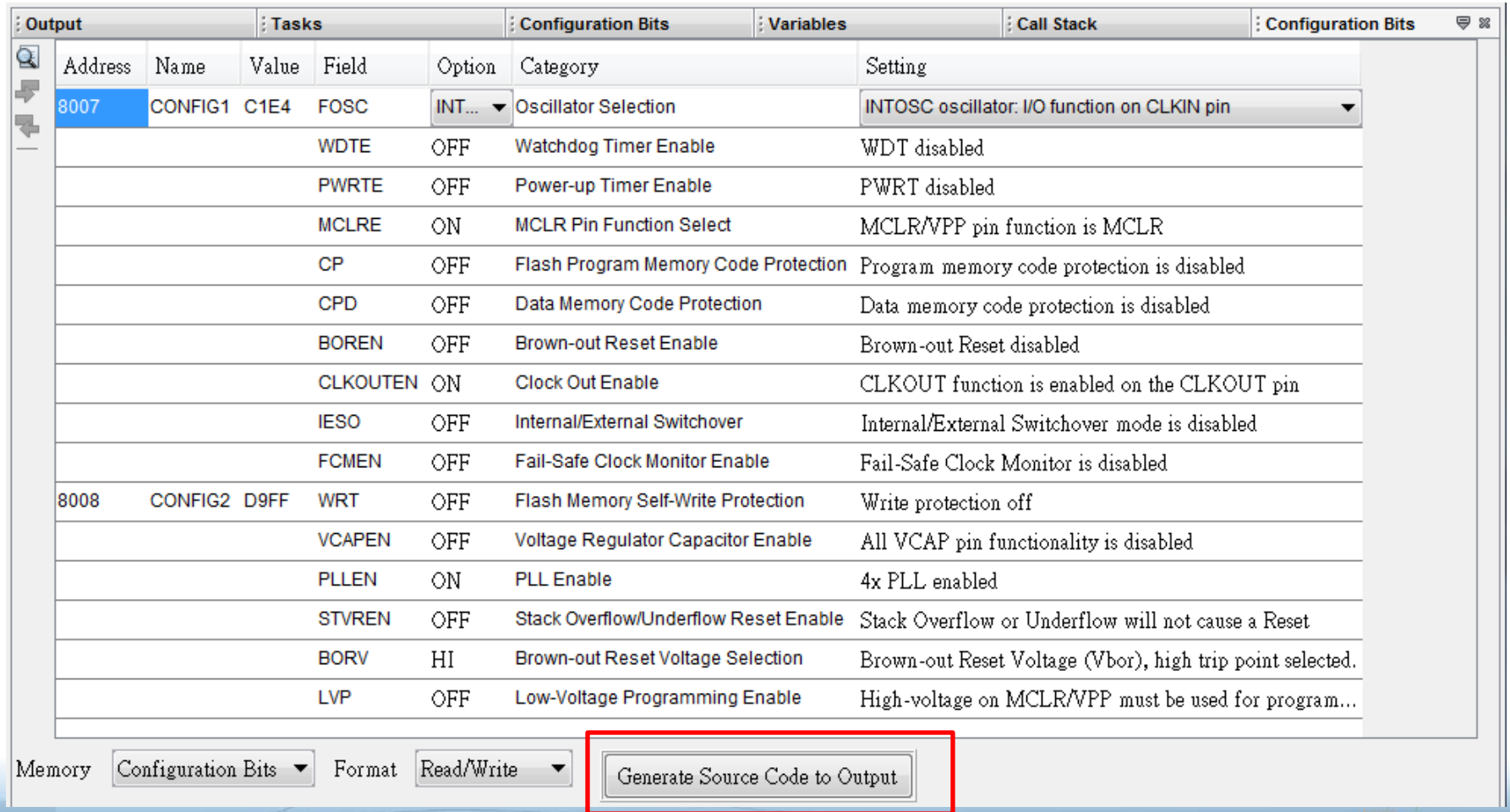
檢視與儲存 Config. 的原始程式
範例：PIC18F4520

- 在 Output 視窗下，開啟 “Config Bits Source” 的選項
- 按滑鼠右鍵，選擇 “Save As”（另存為）config.c
- 將此 C 檔加到專案中並編譯。



如何自動產生 Config. 的設定程式
範例：PIC16F1937

- 在 Option 下，點選要更改的專案
- 按一下按鈕 “Generate Source Code to Output”



自動產生的 Config. 原始程式範例：PIC16F1937

- 檢視一下 PIC16F1937 的 Config. Bits
- 按滑鼠右鍵儲存檔案後再將該檔案加入專案裡

PICkit 3 | Config Bits Source | Debugger Console | Drive LCD Glass (Build, Load, ...) #2

```
// PIC16F1937 Configuration Bit Settings

// CONFIG1
#pragma config FOSC = INTOSC    // Oscillator Selection (INTOSC oscillator: I/O function on CLKIN pin)
#pragma config WDTE = OFF       // Watchdog Timer Enable (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable (PWRT disabled)
#pragma config MCLRE = ON       // MCLR Pin Function Select (MCLR/VPP pin function is MCLR)
#pragma config CP = OFF         // Flash Program Memory Code Protection (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Memory Code Protection (Data memory code protection is disabled)
#pragma config BOREN = OFF      // Brown-out Reset Enable (Brown-out Reset disabled)
#pragma config CLKOUTEN = ON    // Clock Out Enable (CLKOUT function is enabled on the CLKOUT pin)
#pragma config IESO = OFF       // Internal/External Switchover (Internal/External Switchover mode is disabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable (Fail-Safe Clock Monitor is disabled)

// CONFIG2
#pragma config WRT = OFF        // Flash Memory Self-Write Protection (Write protection off)
#pragma config VCAPEN = OFF     // Voltage Regulator Capacitor Enable (All VCAP pin functionality is disabled)
#pragma config PLLEN = ON       // PLL Enable (4x PLL enabled)
```

查看 XC8 使用手冊

- 有關 **Configuration Bits** 的設定請參閱 **XC8** 的使用手冊
  - C:\Program Files\Microchip \xc8\v1.12\docs\manual.pdf
  - 5.3.5 Configuration Bit Access 的章節裡有更多的說明

ID Location 設定

- 使用的語法
  - #pragma config IDLOC*x* = *value*
  - _ _ *IDLOC (nnnn);*

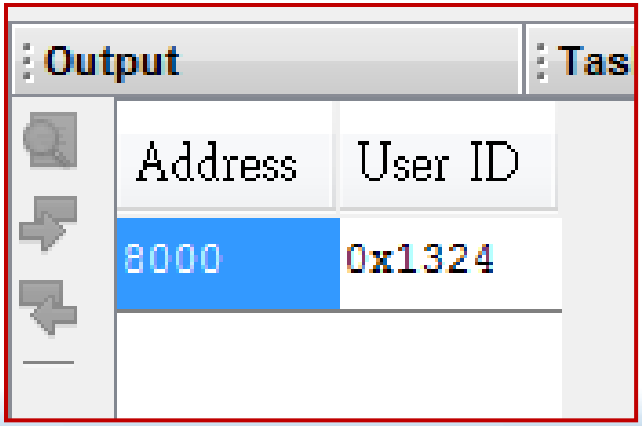
範例:

```
#include    <XC8.h>
#pragma config IDLOC0 = 1
#pragma config IDLOC1 = 3
#pragma config IDLOC2 = 2
#pragma config IDLOC3 = 4
```

或

```
/* will store 1, 5, F and 0 in the ID registers */
__IDLOC(15F0);
```

主選單下，點選 Windiw
→PC Memory Views
→ User ID Memory 視窗觀察



EEPROM 初始燒錄值設定

- 在 **XC8** 底下，**PIC16F** 及 **PIC18F** 的設定語法是一樣的
  - 使用 __EEPROM_DATA( ) 的巨集
  - 擺放位址從 EEPROM 0x00 的位址開始

範例:

```
#include  <xc8.h>
__EEPROM_DATA(0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07);
__EEPROM_DATA(0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F);
```

在 Window 下 → PC Memory Views → EE Data Memory 視窗

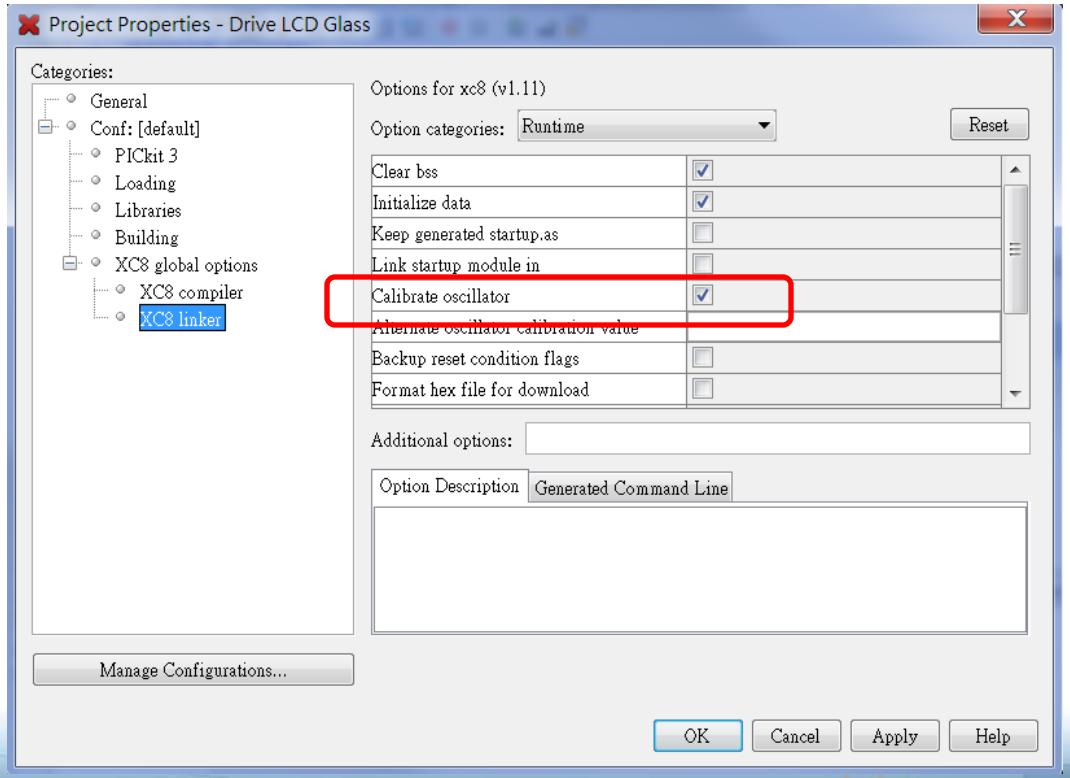
Output | Tasks | Configuration Bits | Variables | Call Stack | Configuration Bits

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	....
10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....

PIC16F 內部RC振盪器校正值的設定

- **PIC10,12,16 (Baseline：PIC12F509)**
  - 啟動模組在開機時會自動將校正值直接存入 OSCCAL 暫存器裡

設定 XC8 Linker
勾選 Calibrate Oscillator



PIC16F 內部RC振盪器校正值的設定

- **PIC10,12,16 (Mid-Range：PIC12F675)**
  - 啟動模組在開機時會自動將校正值載入 OSCCAL 暫存器
  - 詳細說明參考使用手冊: 5.3.9.1 OSCILLATOR CALIBRATION CONSTANTS

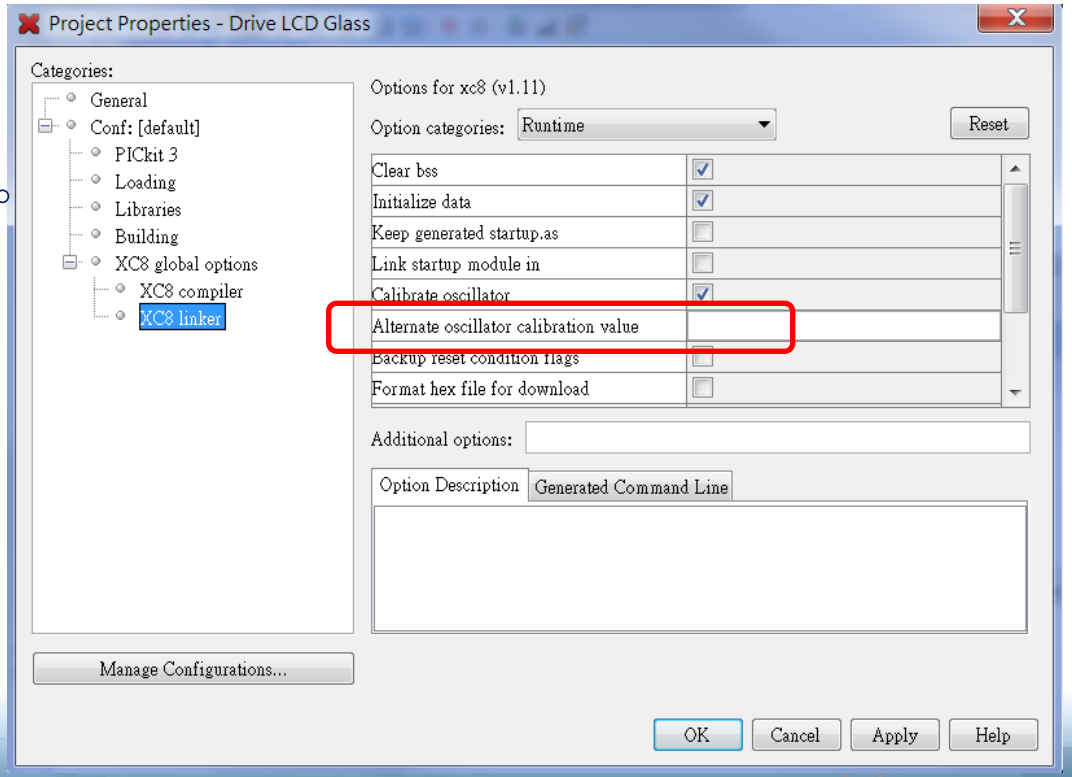
Mid-Range 的振盪校正值可以使用 XC8 提供的巨集。

讀取校正值:

```
calVal = __osccal_val( );
```

設定校正值:

```
OSCCAL = _READ_OSCCAL_DATA( );
```



休息一下、執行 X IDE Lab1.X

- 使用 **MPLAB X IDE** 開啟此專案
- 測試 **XC8** 的環境設定
- 測試中文字型是否可以正常顯示
- 利用此基本程式加入先前所講的
  - **EEPROM** 資料設定
  - **ID** 資料設定
  - 使用 **X IDE** 的 **Config. Code** 產生器建立一個 **Config.c** 的設定程式並加到 **Project** 裡

MICROCHIP

Regional Training Centers

存取特殊功能暫存器

SFR and Bit Access

使用特殊功能暫存器

- 特殊功能暫存器 (SFR) 的存取就像一般的全域變數一樣的簡易
  - 名稱的宣告與資料手冊相同
  - 同時也定義了位元名稱
  - <xc.h> 檔會自動找出所使用元件的定義檔

使用 SFR 的範例：

```
#include <xc.h>

ADCON0 = (channel << 3) + 0xC1;
GODONE = 1;          // ADC 開始轉換
while(GODONE);       // 轉換完成?
```

SFR 暫存器的定義

- 暫存器使用絕對定址方式直接映射到所指定的位址

xc8 所採用的絕對定址方式

```
type FSR_name @ address;
```

- SFR 將置放在特定的 address
- 變數如需擺放在特定 RAM 的位址， 可以使用此種絕對位址的宣告方式
- 也適用於位元的宣告

```
volatile unsigned char ADCON0 @ 0x01F;
volatile bit GODONE @ (unsigned)&ADCON0*8+1;
```

暫存器的限制

- 避免去修改 **CPU** 內部的暫存器 避免中間資料改變的錯誤，例如:
  - **WREG**
  - **FSR0，FSR1, FSR2**
  - **STATUS (Z & C bits) …**

```
read();
WREG = 0x12;   // 避免此寫法
Z = 0;         // 避免此寫法
process();
```

XC8 位元定址方式

- 使用位元的絕對位址方式
  - 源自於 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
  - 源自於 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

使用絕對位元
PORTA 的位元定義 (pic16f887.h)

- 可以直接使用 **RA0** 或 **RA0_bit** 的名稱

```
extern volatile __bit          RA0     @ (((unsigned) &PORTA)*8) + 0;
#define                     RA0_bit    BANKMASK(PORTA), 0
extern volatile __bit          RA1     @ (((unsigned) &PORTA)*8) + 1;
#define                     RA1_bit    BANKMASK(PORTA), 1
extern volatile __bit          RA2     @ (((unsigned) &PORTA)*8) + 2;
#define                     RA2_bit    BANKMASK(PORTA), 2
extern volatile __bit          RA3     @ (((unsigned) &PORTA)*8) + 3;
#define                     RA3_bit    BANKMASK(PORTA), 3
extern volatile __bit          RA4     @ (((unsigned) &PORTA)*8) + 4;
#define                     RA4_bit    BANKMASK(PORTA), 4
extern volatile __bit          RA5     @ (((unsigned) &PORTA)*8) + 5;
#define                     RA5_bit    BANKMASK(PORTA), 5
extern volatile __bit          RA6     @ (((unsigned) &PORTA)*8) + 6;
#define                     RA6_bit    BANKMASK(PORTA), 6
extern volatile __bit          RA7     @ (((unsigned) &PORTA)*8) + 7;
#define                     RA7_bit    BANKMASK(PORTA), 7
```

使用絕對位元
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為 0
  - 公式 : (8 bits * SFR 的位址) + 偏移位元

範例: 存取 PORTA 的 RA5

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

```
RA5 = 1;          // RA5 輸出 High
Nop( );           //  避免 Read-Modify-Write 現象
RA3= 0;           // RA3 輸出 Low
If (RB0) ….       // 判斷RB0
```

使用 bit 型別

- **bit 型別使用布林數 (boolean values)**
  - 使用 bit 的定義

```
bit     motor_is_on;     // motor state flag
```

  - 會將零散 bit 位元變數整合在一個 Byte 的 RAM
  - Bit 變數無法為 auto 變數型態
    - 可以宣告成 local 變數，但必須加上 static 的宣告
  - Bit 變數無法透過指標存取
  - 使用起來非常簡單

bit 型別範例

- 透過 **bit** 的簡易的整數位元轉移

範例:

```
char value = 0x34; // 0b00111000
bit flag;

flag = value;      // flag will be assigned 0
```

- 簡易使用的 **bit** 型別

```
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
pic16f887.h

```
// Register: PORTA
extern volatile unsigned char       PORTA          @ 0x005;
#ifndef _LIB_BUILD
asm("PORTA equ 05h");
#endif
// bitfield definitions
typedef 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_t;
extern volatile  PORTAbits_t  PORTAbits  @ 0x005;
```

ANSI C 標準語法，
使用位元結構方式來宣告。
此方式與 C18 相容

使用位元結構方式
(與 C18 & C30 相容)

將 PORTD 的 bit3 & bit5 設定為輸出腳並設定輸出為 High：

```
TRISDbits.TRISD3 = 0;          // 設定 RD3 為輸出腳功能
TRISDbits.TRISD5 = 0;          // 設定 RD5 為輸出腳功能
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 為輸出腳功能
TRISAbits.TRISA0 = 1;          // 設定 RA0 為輸入腳功能
:
If (!SW1) LED1=1;              // SW1 被按下，點亮 LED1
else LED1=0;                   // SW1 放開，LED1 熄滅
```

浮點數格式

- 支援兩種浮點數格式
  - 32-bit IEEE format

sign	exp	mantissa
x	xxxx xxxx	xxx xxxx xxxx xxxx xxxx xxxx

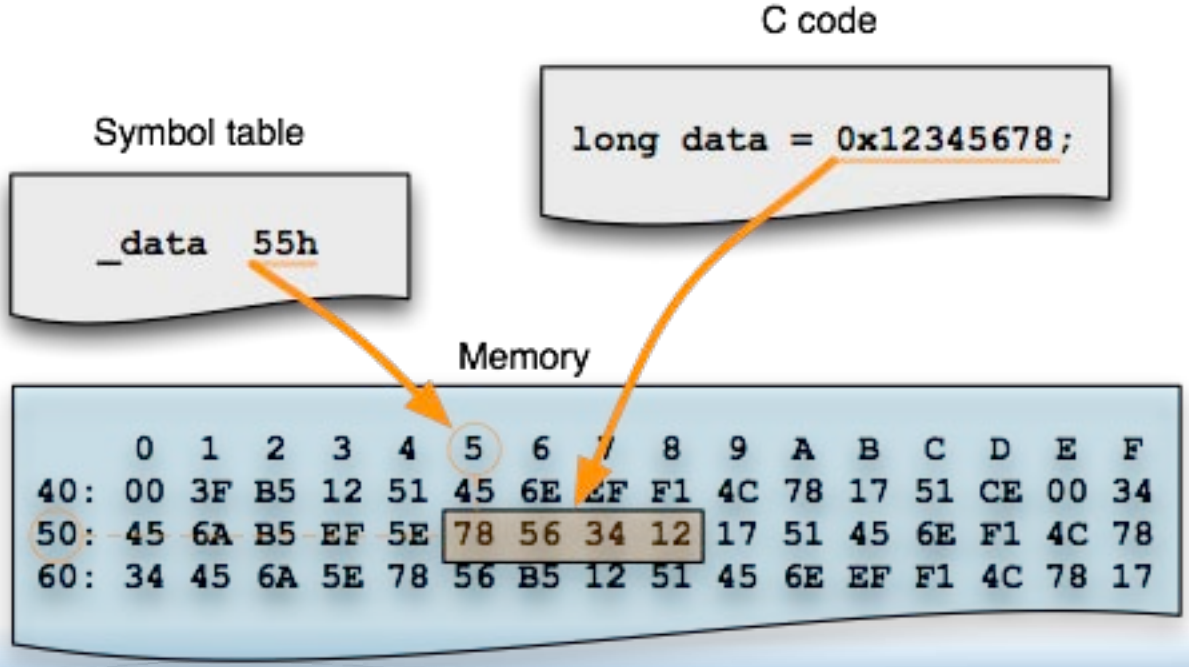
  - 24-bit truncated IEEE format

sign	exp	mantissa
x	xxxx xxxx	xxx xxxx xxxx xxxx

- **The larger format offers greater precision**

資料的擺放方式

- 超過 **Byte** 以上的數值採用 “***little endian***” 的格式
  - **LSB** 放在最低的位址
  - 結構變數的第一個成員(或位元成員) 儲存在最低的位址



XC8 指定變數擺放位址

- XC8 使用 @ address 的方式設定變數的絕對位址
- 範例：
  - 在 MPLAB C18 變數絕對位址的定義方式

```
#pragma udata myUdata=0x100
int foobar;
```

  - 在 MPLAB XC8 變數絕對位址的定義方式

```
int foobar @ 0x100;
```

函數的絕對位址

- 函數呼叫:
  - 注意 :Baseline 元件只有兩層或四層堆疊可用
  - 函數的呼叫將使用內建的硬體堆疊
  - 參數的傳遞無法使用硬體堆疊，需畫出一塊 RAM 來做參數的傳遞
- 使用絕對定址方式來設定函數的位址
  - "*函數原形宣告*" 上使用絕對定址法 ***@ address***
  - 函數進入點將會被設定在 address

設定函數在特定的絕對位址

```
void special(int a) @ 0x100;
```

函數絕對位址的設定

- **XC8** 使用 **@ address** 的方式設定函數的絕對位址
- 範例：
  - 在 MPLAB C18 函數絕對位址的定義方式

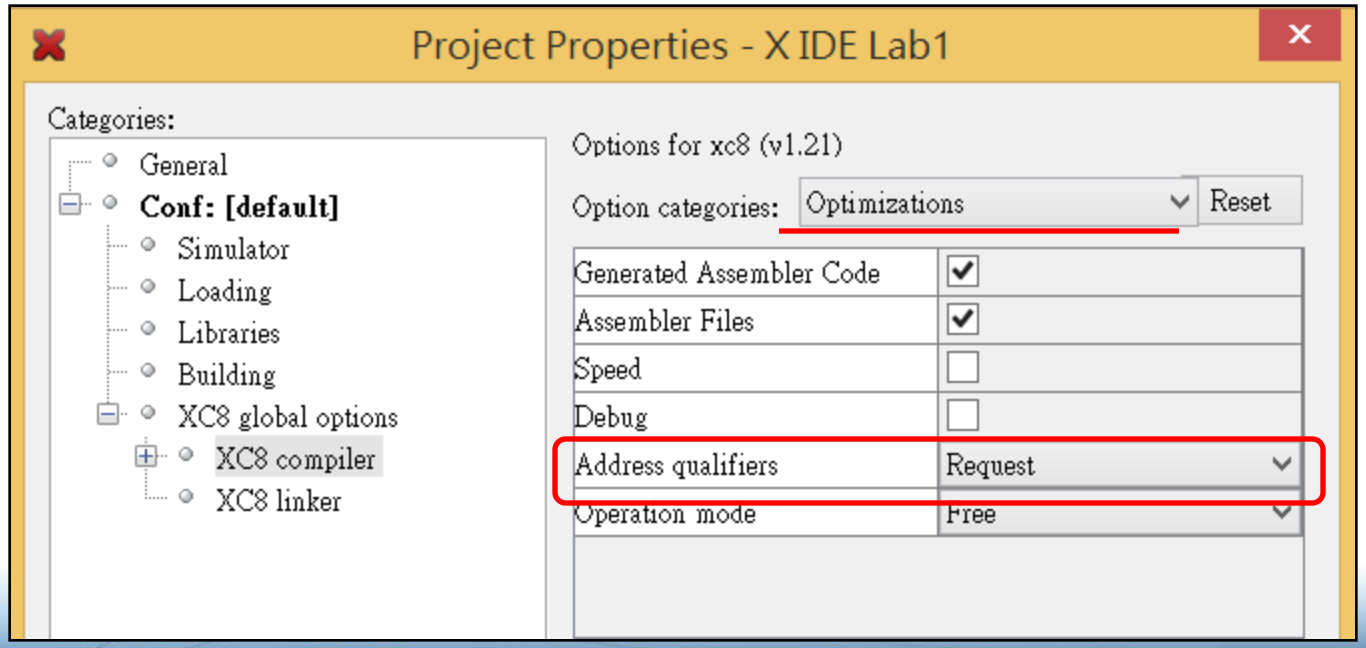
```
#pragma code myCode=0x2000
int calcOffset(int radius) { ... }
```

- 在 MPLAB XC8 函數絕對位址的定義方式

```
int calcOffset(int radius) @ 0x2000
{ ... }
```

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。
  - XC8 編譯器對 bankx 的安排內定是忽略的 (Ignore)，要啟用此項指定需在編譯時加以設定為 “Request”。



MICROCHIP

Regional Training Centers

Qualifiers and Memory Allocation

Standard Qualifiers

- **Const 修飾 (qualifier)**
  - XC8 使用 const 作為常數的宣告，取代 C18 使用 rom 作為常數的設定
  - 宣告時需要有常數值的存在
  - 常數值只能讀取(不能被寫入)
  - 所宣告的常數會被擺在程式記憶空間裡
  - C18 所使用的 **rom** 在程式記憶體的宣告將不在使用， XC8 將用 **const** 取而代之。

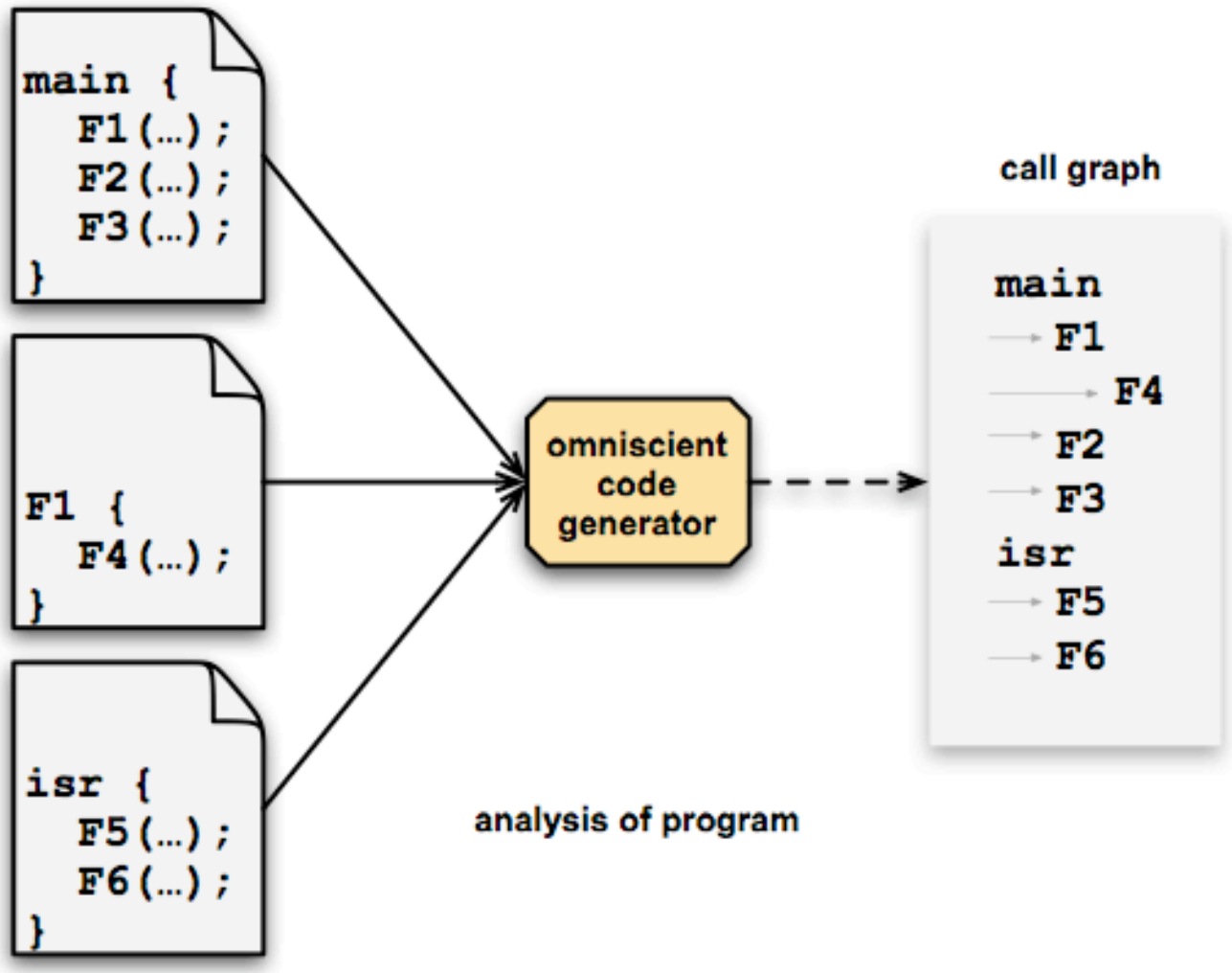
Standard Qualifiers

- **`volatile` qualifier**
  - 該變數值可被外部周邊修改，其內容值不一定要經過程式的執行才會改變
  - 中斷也屬於外部的修改，所以使用於中斷函數的變數一定要宣告成 volatile 的型態
  - 一般使用 volatile 型態:
    - 硬體周邊的暫存器宣告 (Timer, UART…)
    - 暫存器會因外部輸入而改變的 (PORT)
    - 在中斷函數裡所使用到的變數
    - 不想被最佳化修改的變數

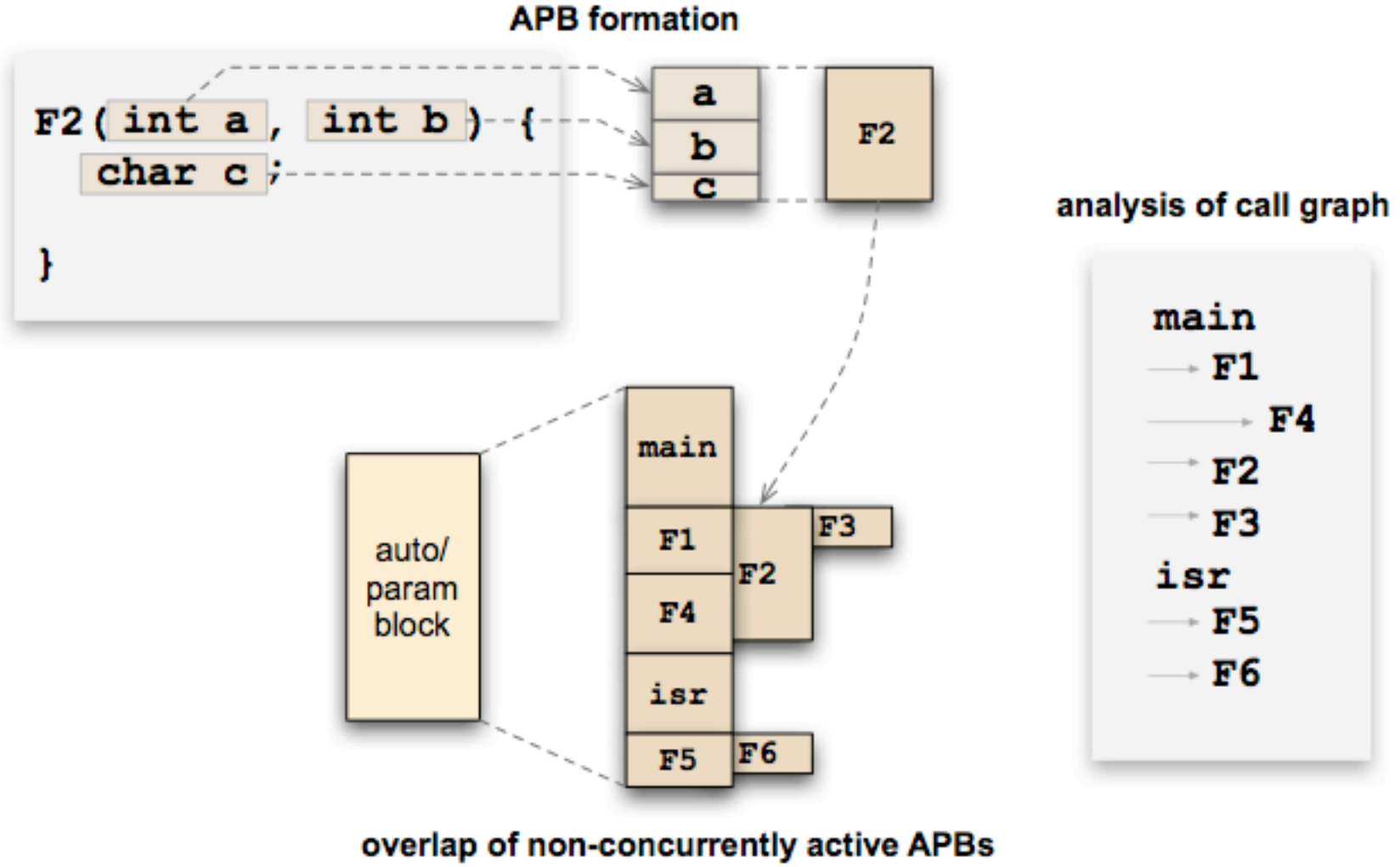
Microchip XC Qualifiers

- **`persistent` qualifier**
  - 在系統能供電的情況下，當元件重置 (reset) 時，所宣告的資料變數將不會被 C 的啟動模組初始化或歸零。
  - 冷、熱開機的判斷，喚醒時的判斷
- **底下傳統的修飾字將被認定是內定設定且被忽略**
  - `near`
  - `bankx`

記憶體分配 (函數)



記憶體分配 (參數)


APB formation
F2(int a, int b) {
char c;
}
a
b
c
F2
analysis of call graph
main
F1
F4
F2
F3
isr
F5
F6
auto/
param
block
main
F1
F4
isr
F5
F2
F3
F6
overlap of non-concurrently active APBs

MICROCHIP

Regional Training Centers

Pointers

一般指標 宣告

- 標準 C 的指標支援，可指向資料區及程式記憶空間
- 一般指標的定義及範例:

資料型別 & 修飾	*	指標的修飾	指標的名稱 & 初始值設定;
`char`	*		`cp ;`
`const char`	*		`cp ;`
`char`	*	`const`	`cp = address;`
`const char`	*	`const`	`cp = address;`

cp 是一個指標其內容值為一個常數，指標型別為 char 的指向程式記憶空間

XC8 的指標

- 指標有兩種
  - 一是指向 Data RAM 的指標
  - 二是指向 Program 的指標
- **PIC16 的指標依架構有三種型態**
  - Base-Line (PIC16F5x)
  - Mid-Range (PIC16Fxxx)
  - Enhanced Mid-Range (PIC16F1xx)
- **PIC18 的指標**
  - FSR 使用於 RAM
  - TABLE Read/Write 使用於ROM

PIC16 指標 Base-Line

- 以 **PIC12F508** 為例:
  - SFR 0x00 ~ 0x06 ; RAM 0x07 ~ 0x1F
  - ROM 0x000 ~ 0x1FF (0.5KW)
- **RAM** 指標，使用 **FSR**，區間為 **0x07 ~ 0x1F**
- **ROM** 指標，採先用 **FSR** 當索引再使用 查表方式完成

```
MOVF    FSR,W   ; 將 ROM 指標加一
INCF    FSR,W
ANDLW   0x1F    ; 限定查表的範圍
ADDWF   PCL,F   ; 開始查表
RETLW   0xn0
RETLW   0xn1
:
```

- 使用時要特別注意 **RAM** 的空間
- **Const** 的查表不可超過 **250**個查表資料，最好置放在 **PAGE0** 的上半區域 **(0x000 ~ 0x0FF)**

12F508_Pointer Lab1

PIC12F508 指標練習

- 開啟本練習專案
  - **\Pointer Labs\12F508_Pointer Lab1.X**
- 本練習使用 **BIG-5** 的中文編碼方式
  - 如需變更請到專案下的內容的：
    - **“Categories” → “General” 的 “Encoding” 改成 BIG-5**

Lab1 - PIC12F508 指標範例

```
const unsigned char ROM_Buffer[  ]= { 0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80, \
                                     0x90,0xa0,0xb0,0xc0,0xd0,0xe0,0xf0 };
unsigned char Input_Buffer[10]    ;            // 宣告 RAM 陣列
const unsigned char * ROM_PTR;
unsigned char *RAM_PTR;
near unsigned char j ;

void main(void)
{
    ROM_PTR = ROM_Buffer;
    RAM_PTR = Input_Buffer ;

    for (j=0; j<=10; j++) *RAM_PTR++  =  *ROM_PTR++;
    NOP();

    while(1);
}
```

File Registers

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
00	00	00	21	1F	F2	00	00	0B	10	20	30	40	50	60	70	80	..!..... . 0
10	90	A0	B1	0C	00	00	00	00	00	00	00	00	00	00	00	01	....

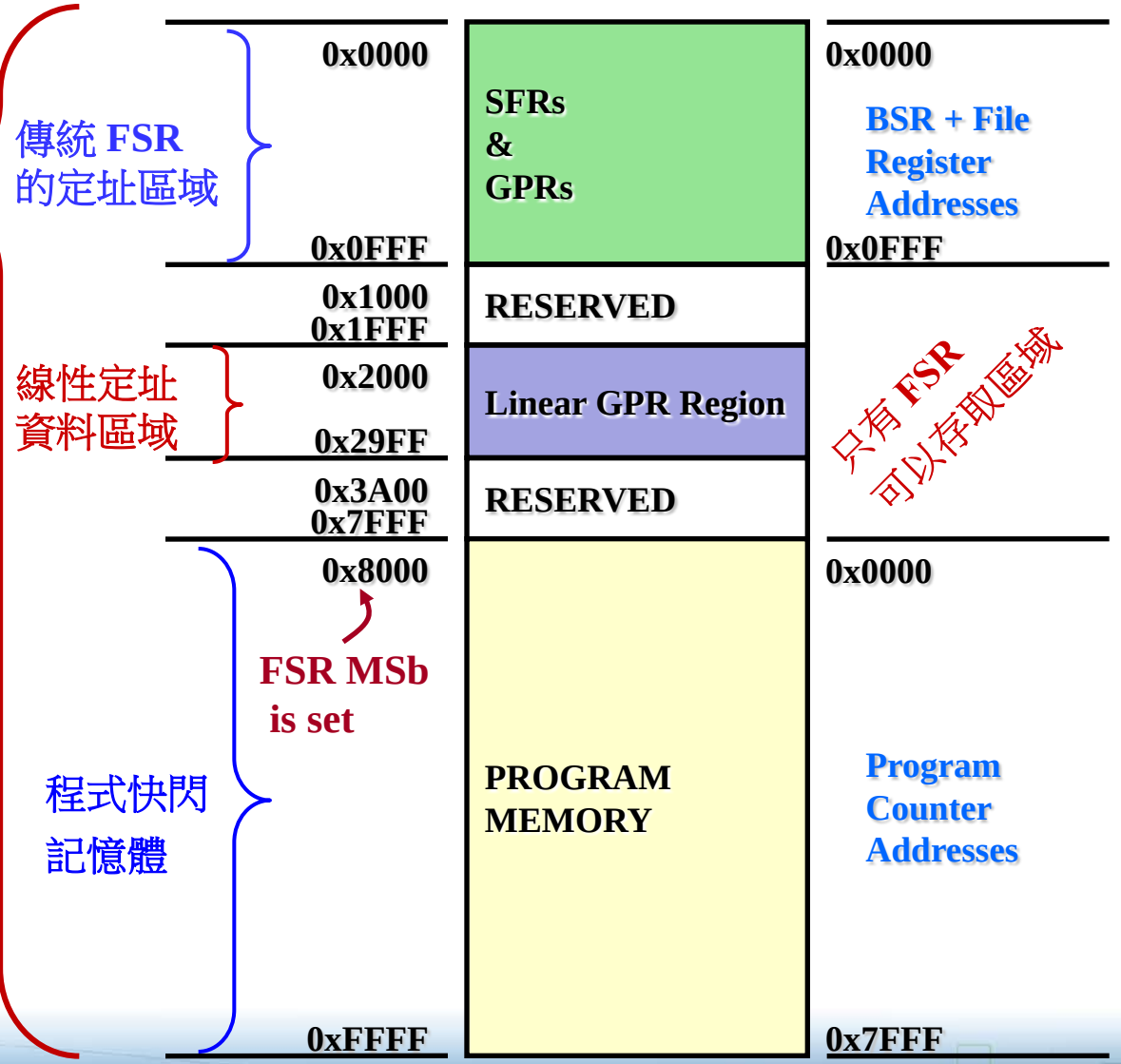
PIC16F1xxx 指標
Enhanced Mid-Range

- **PIC16F1xxx 的 RAM 指標有兩種方式**
  - 傳統直接索引方式 (陣列小於 80 Bytes)
  - 線性索引方式 (新功能，可索引到所有的 RAM 區域) (Lab2)
    - FSRH: FSRL 的 b15 = 0

- **PIC16F1xxx 的 ROM 指標**
  - 傳統查表方式 (查表資料小於 255 Bytes)
  - 改良式的 ROM 指標查表法，視野範圍 32KB
    - FSRH: FSRL 的 b15 = 1
    - ROM 位址 : 0x0000 ~ 0x7FFF

PIC16F1xxx 新架構下的 FSRs

- 兩組 16-bit FSRs
- 視野可達 64K 全範圍
  - 傳統定址資料區域
  - 線性定址資料區域
  - 程式快閃記憶體
- FSR 最高位元
  - = 0，RAM 空間
  - = 1，程式空間
- FSRs 有更多的功能支援以方便查表的應用



16F1939_Pointer Lab2

PIC16F1939 線性指標練習

- PIC16F1939 有 1024 Bytes RAM 分散在 Bank0 ~ Bank12
  - 本練習使用 960 Bytes RAM + Access RAM
- 開啟本練習專案
  - \Pointer Labs\ 16F1939_Pointer_Lab2.X
- 本練習使用 BIG-5 的中文編碼方式
  - 如需變更請到專案下的內容的：
    - “Categories” → “General” 的 “Encoding” 改成 BIG-5

Lab2 - PIC16F1939 指標範例

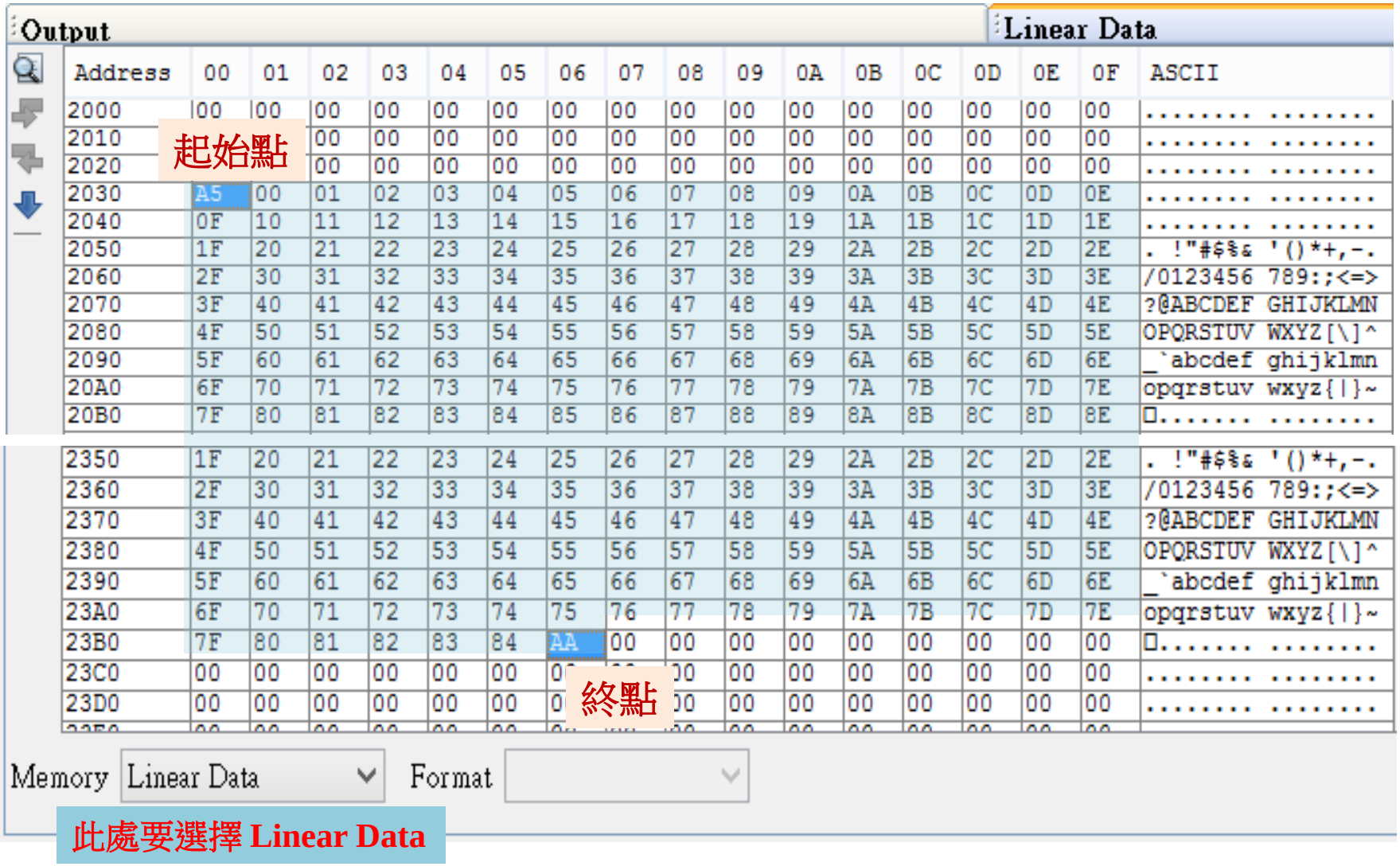
```
unsigned char InputBuffer[960]    ;     // 宣告陣列
unsigned char *near PTR ;               // PTR 指標位址設在 Access Memory
near unsigned int j ;                   // 變數 j 放在 Access Memory (0x70~0x7F)

void main(void)
{
    PTR = InputBuffer ;

    *PTR++ = 0xa5;                      // 第一個陣列值先填入 0xa5
    for (j=0; j<=900; j++) *PTR ++ = (unsigned char) j;
    *PTR = 0xaa;                        // 最後填入 0xaa

    NOP( );
    while(1);
}
```

Lab2 線性指標結果



16F1939_Pointer Lab3

PIC16F1939 FSRH:FSRL ROM指標練習

- 本實驗使用 **PIC16F1xxx** 系列新架構的 **FSR** 對 **ROM** 的查表
  - 利用 **const** 建立一個 **303 Bytes** 的 **ROM** 陣列
  - 使用線性指標方式在位址 **0x2100** 建立一 **RAM** 的陣列，陣列大小 **320 Bytes**
  - **..\XC8\Pointer Labs\ 16F1939_ROM_Pointer_Lab3.X**
- 本練習使用 **BIG-5** 的中文編碼方式
  - 如需變更請到專案下的內容的：
    - **“Categories” → “General” 的 “Encoding” 改成 BIG-5**

Lab3 的建立在 ROM 的查表資料

```
const unsigned char Lookup_Table[ ]  =
    {0xA0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
     18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
     34,35,36,37,38,39,40,41,42,43,44,45,56,57,48,49,50,
     51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,
           :
     1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
     18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
     34,35,36,37,38,39,40,41,42,43,44,45,56,57,48,49,50,
     51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,
     68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,
     84,85,86,87,88,89,80,91,92,93,94,95,96,97,98,99,100,
    0xAA,0xAB};
```

LAB3 利用 FSR 查表到 RAM

```
unsigned char Input_Buffer[320] @0x2100;
const unsigned char *near ROMPTR;
unsigned char *near RAMPTR ;   // PTR 指標位址設在 Access Memory
near unsigned int j ;          // 變數 j 放在 Access Memory

void main(void)
{
    ROMPTR = Lookup_Table ;
    RAMPTR = Input_Buffer ;
    for (j=0; j<=302; j++) *RAMPTR++ = *ROMPTR++;

    NOP( );
    while(1);
}
```

Lab3 ROM 到 RAM 結果

查表開始

20F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2100	A0	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2110	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2120	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	38	39
2130	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
2140	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
2150	50	51	52	53	54	55	56	57	58	59	50	5B	5C	5D	5E	5F
2160	60	61	62	63	64	01	02	03	04	05	06	07	08	09	0A	0B
2170	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
2180	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B
2190	2C	2D	38	39	30	31	32	33	34	35	36	37	38	39	3A	3B
21A0	3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A	4B
21B0	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	50	5B
21C0	5C	5D	5E	5F	60	61	62	63	64	01	02	03	04	05	06	07
21D0	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
21E0	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
21F0	28	29	2A	2B	2C	2D	38	39	30	31	32	33	34	35	36	37
2200	38	39	3A	3B	3C	3D	3E	3F	40	41	42	43	44	45	46	47
2210	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57
2220	58	59	50	5B	5C	5D	5E	5F	60	61	62	63	64	AA	AB	00
2230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

查表結束

MICROCHIP

Regional Training Centers

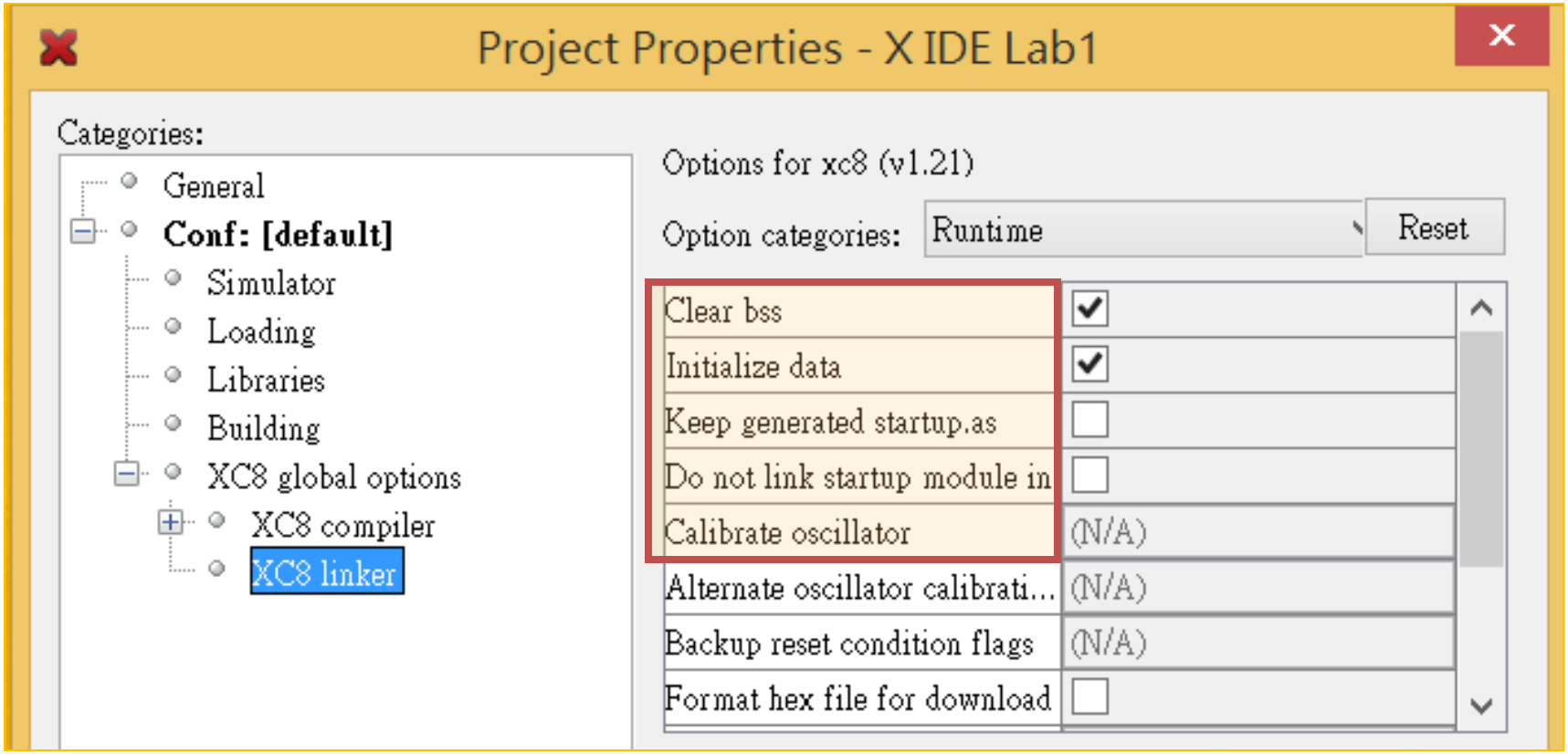
開機初始執行環境

重置後的動作

- **立即去執行啟動模組 (Startup Code) @0x0000**
  - 啟動程式以組合語言架構撰寫，編譯時自動產生並加入到程式裡
  - 主要的任務
    - 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( )

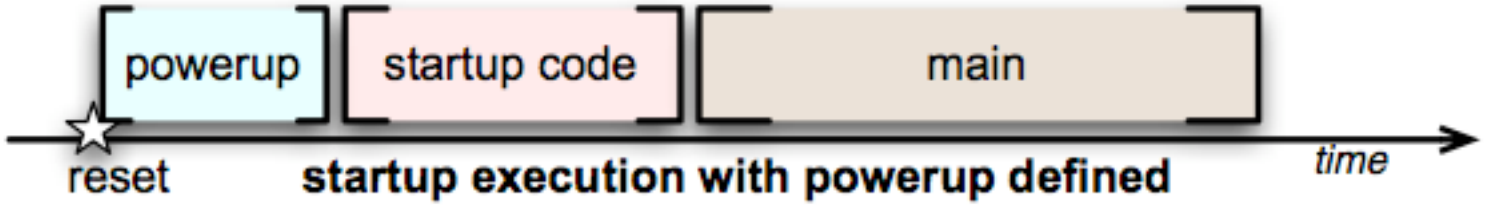
啟動程式模組控制

- 啟動模組控制設定選項



自訂開機立即執行程式 powerup.as

- 一般開機後會立即執行 C 的啟動模組程式
- 可自訂開機立即執行程式 **(powerup routine)** ，它可以比啟動模組先執行，做一些開機的初始設定。例如: 一開機時 **I/O** 腳位的立即設定
  - 最好使用組合語言撰寫



powerup.as 撰寫準則

- 設定的用法是 XC8 自動提供的:
  - 組語寫在 “psect” 後，副程式名稱為 **powerup**
  - 程式設定完成後將跳到 **start** 的函數
  - 程式長度不限
  - 程式的安排不受中斷向量影響
- 原始參考範例提供
  - 參考組語範本: **powerup.as**
  - 組合語言檔案也許要做預處理步驟 (參考使用手冊的 C and assembly 的章節)

C:\Program Files (x86)\Microchip\xc8\v1.21\sources\pic\powerup.as
C:\Program Files (x86)\Microchip\xc8\v1.21\sources\pic18\powerup.as

powerup.as 範例

使用時須將 **powerup.as** 的檔案加到 **Project** 裡一同編譯。

```
#include          "aspic.h"
    GLOBAL   powerup,start
    PSECT    powerup
powerup:
;   將“立即開機方式”寫在這裡
        clrf     STATUS
        movlw    start>>8
        movwf    PCLATH
        goto     start ;跳到啟動模組
```

MICROCHIP

Regional Training Centers

XC8 中斷處理

PIC16Fxxx

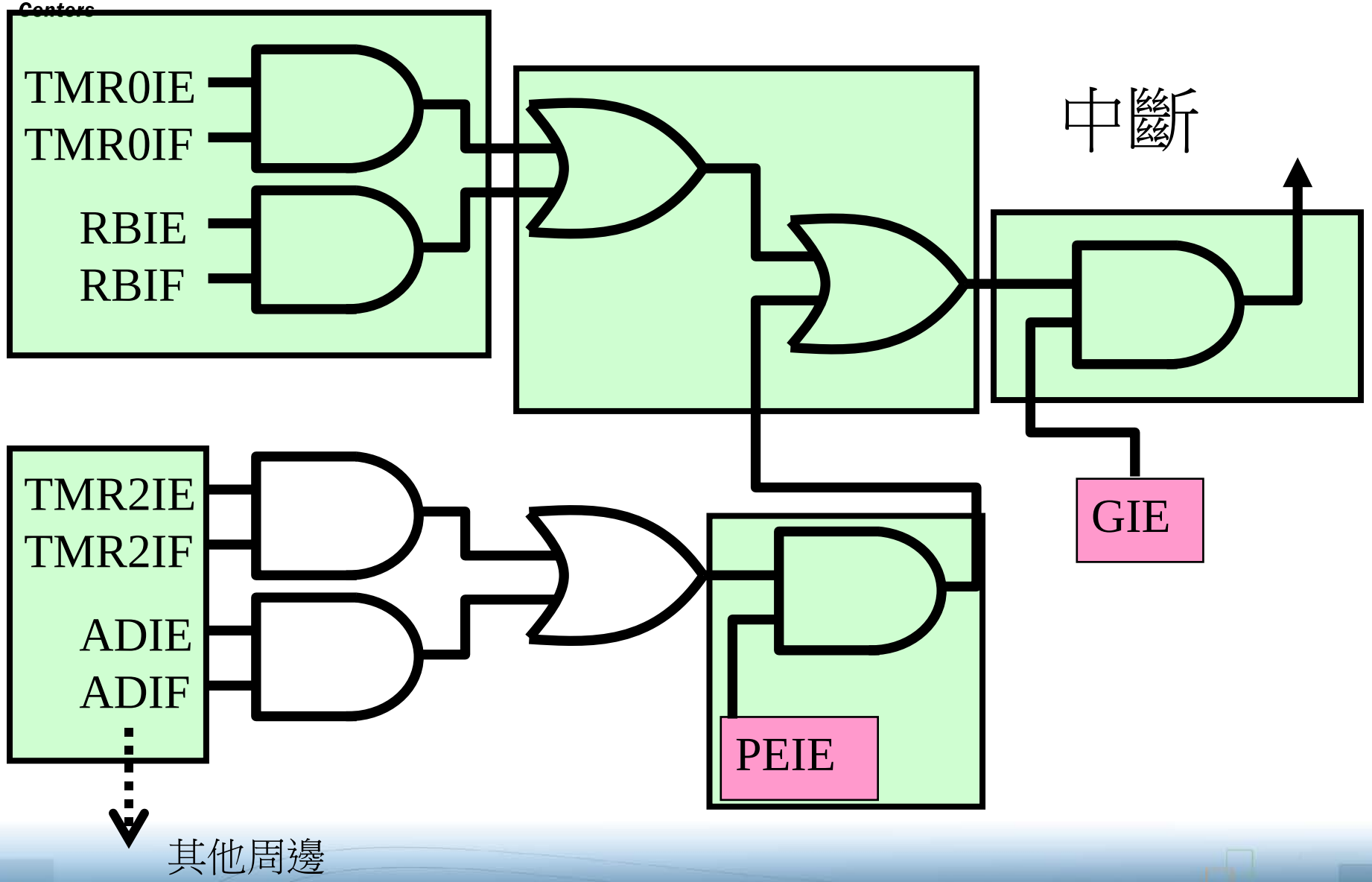
PIC16F1xxx

PIC18Fxxx

中斷注意事項

- 中斷函數無參數傳遞功能
- 中斷所使用的變數需加入 volatile 的宣告
- 中斷函數儘量不要使用 Local 變數，影響斷響應時間
- 中斷越短越好，不要做太多的處理，可以設定 Flag 後交給主程式處理
- 中斷函數最好不要做額外的數學的運算
- 中斷裡不要再呼叫主程式有在使用的函數

PIC16F 中斷邏輯



PIC16F Interrupts

- 一般的中階 PIC16F 有支援一個中斷
  - 唯一的中斷進入點，執行中斷程式
    - 中斷位址為 0x0004
  - 可支援多重中斷事件處理
    - 因為沒有優先權功能，所以中斷事件須詢問(Polling)中斷旗號以決定是誰發生中斷
  - 背景儲存中斷程式自動處理
  - 堆疊只有 8 層，主程式使用至七層，保留一層給中斷使用

PIC16F1xxx Interrupts

- 加強型的中階 **PIC16F1** 支援一個中斷
  - 唯一的中斷進入點，執行中斷程式
    - 中斷位址為 0x0004
  - 可支援多重中斷事件處理
    - 因為沒有優先權功能，所以中斷優先權，由輪詢(Pollong)先後決定
  - PIC16F1 背景儲存採用 Shadow 方式
    - Shadow Register 在 BANK31
  - 堆疊擴充至 16 層

撰寫 PIC16F 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 Access Memory(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

PIC16F 中斷函式範例

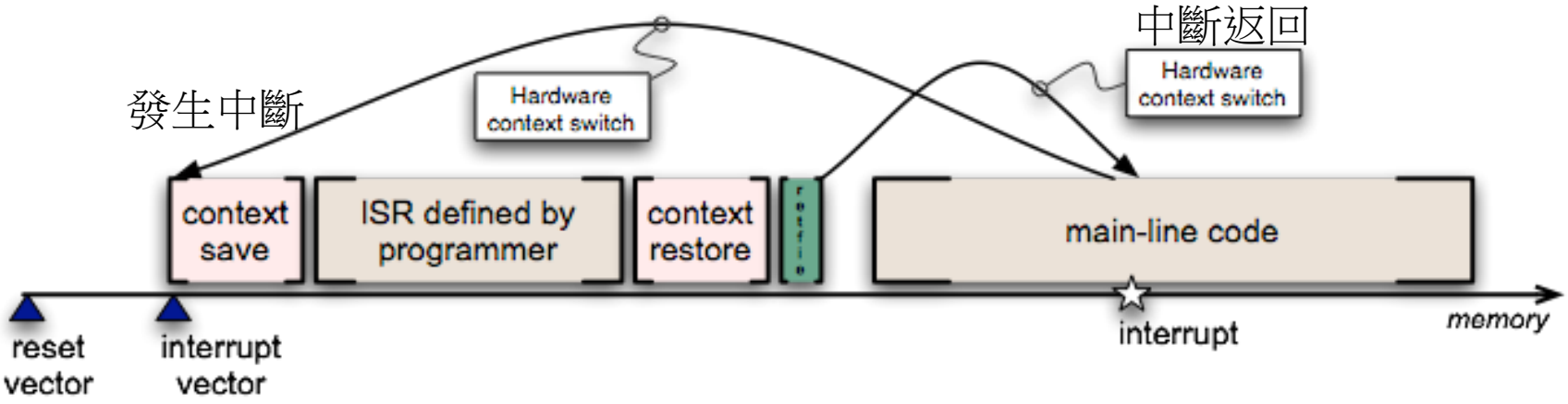
- C 函數可以被連結到中斷向量的位置，只要在函數原型宣告使用保留字 *“interrupt”* 即可
- 啟動中斷的設定
  - GIE =1 ;
  - PEIE = 1 ;
  - RCIE = 1 ;
- 同時檢查週邊的中斷旗號與周邊的中斷致能位元

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

中斷函數名稱

中斷處理動做

- 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



PIC16F 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)

中斷函式裡勿使用一般函式

- 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
  // ...
```

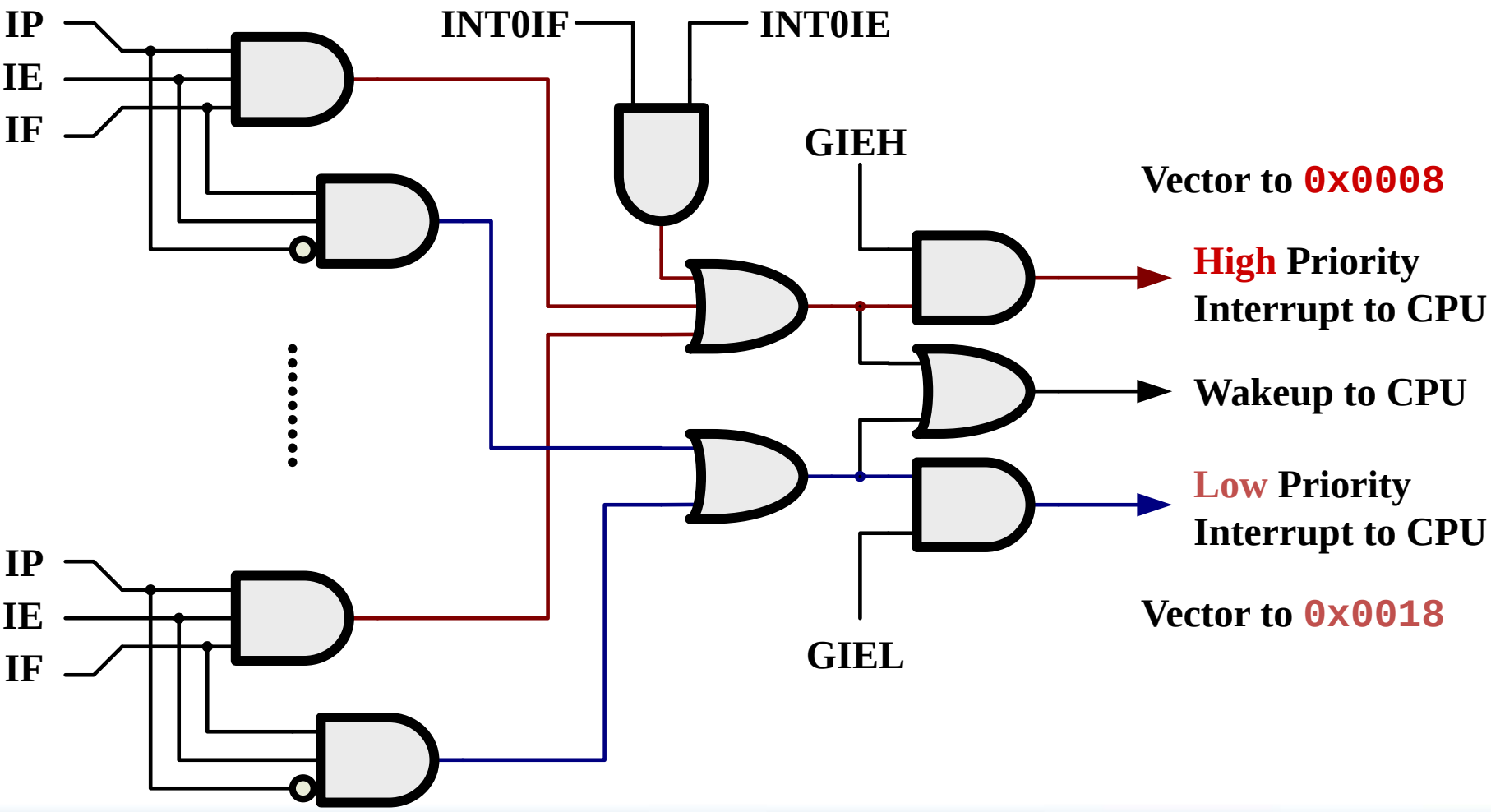
PIC16F 多個中斷處理

```
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
        }
}
```

PIC18Fxxxx 中斷邏輯電路

優先權模式



PIC18F 中斷處理

- **PIC18Fxxxx 有兩個中斷進入點**
  - 高優先權==>中斷位址0x0008
  - 低優先權==>中斷位址0x0018
  - 每個中斷源均可選擇其中斷優先權（二選一，INT0 除外）
  - 每個中斷源均有獨立的中斷旗標 (Interrupt Flag)
  - 每個中斷源均可單獨 Enable 或 Disable
  - 中斷旗標的清除 ==> 需自行用軟體方式清除
    - USART 產生的 TXIF 及 RCIF 旗標， 無法直接用軟體清除 Flag
    - 清除 TXIF → 寫入 TXREG
    - 清除 RCIF → 讀取 RCREG

PIC18F 的 Shadow 暫存器

- **Shadow Register**
  - 提高中斷程式對事件的反應速度
- 高優先權中斷
  - W，BSR，STATUS 自動存入Shadow Register
  - RETFIE FAST：自 Shadow Register 取回暫存值
- 低優先權中斷
  - W，BSR，STATUS 的存入、取出需透過軟體堆疊
  - 程式的返回：RETFIE 0

C18 對高優先中斷設定

```
#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函式(isr_high)

中斷服務函式(isr_high)

XC8 對 PIC18F 中斷設定

- 原先的 C18 對中斷的設定較直接
- XC8 對中斷的設定較為簡單
- 同樣 XC8 對 PIC18F 也支援高、低優先權的中斷設定
- 一樣的，高優先權會使用 Shadow
- 低優先權使用堆疊做背景儲存，響應速度較高優先權中斷慢

PIC18Fxxxx 的中斷設定

- 高優先權中斷函數
  - void interrupt HighISR(void)
  - 無法傳入及回傳參數
- 低優先權中斷函數
  - void interrupt low_priority LowISR(void)

注意: 除了中斷函數的設定，其他相關的中斷啟用位元及優先權控制位元也要設定到

LAB3 中斷範例程式 (PIC18F)

```
void interrupt HighISR(void)                    // 高優先權中斷函數
{
   if (TMR0IE && TMR0IF)
   {
      WriteTimer0(65536-488);     // 500mS Period, 500mS/4uS/256 = 488
      LATDbits.LATD0 = !LATDbits.LATD0;
      TMR0IF=0;
   }
}

void interrupt low_priority LowISR(void)      //  低優先權中斷函數
{
   if (TMR1IF && TMR1IE)
   {
      WriteTimer1(65536-7812);    // 250mS Period, 250mS/4uS/8 = 7812
      LATDbits.LATD7 = !LATDbits.LATD7;
      TMR1IF = 0;
   }
}
```

XC8 編譯後的中斷程式碼

Line	Address	Opcode	Label	DisAssy
1	0000	EF6F		GOTO 0xDE
2	0002	F000		NOP
3	0004	FFFF		NOP
4	0006	FFFF		NOP
5	0008	CFFA	HighISR	MOVFF PCLATH, 0x1C
6	000A	F01C		NOP
7	000C	CFFB		MOVFF PCLATU, 0x1D
8	000E	F01D		NOP
9	0010	CFE9		MOVFF FSR0L, 0x1E
10	0012	F01E		NOP
11	0014	ED12		CALL 0x624, 1
12	0016	F003		NOP
13	0018	CFD8	LowISR	MOVFF STATUS, 0x6
14	001A	F006		NOP
15	001C	CFE8		MOVFF WREG, 0x7
16	001E	F007		NOP
17	0020	CFE0		MOVFF BSR, 0x8
18	0022	F008		NOP
19	0024	CFFA		MOVFF PCLATH, 0x9
20	0026	F009		NOP
21	0028	CFFB		MOVFF PCLATU, 0xA
22	002A	F00A		NOP
23	002C	CFE9		MOVFF FSR0L, 0xB
24	002E	F00B		NOP
25	0030	CFEA		MOVFF FSR0H, 0xC
26	0032	F00C		NOP
27	0034	CFE1		MOVFF FSR1L, 0xD
28	0036	F00D		NOP
29	0038	CFE2		MOVFF FSR1H, 0xE
30	003A	F00E		NOP
31	003C	CFD9		MOVFF FSR2L, 0xF
32	003E	F00F		NOP
33	0040	CFDA		MOVFF FSR2H, 0x10
34	0042	F010		NOP

重置位址

高優先權中斷

低優先權中斷

LAB3 裡相關的中斷位元

- 單單只有中斷函數設定中斷還是無法啟動的
- 有那些中斷位元還要做設定
  - 要使用 Timer0 & Timer1 的中斷前置設定

```
INTCON2bits.TMR0IP = 1;          // 設定 Timer0 為高優先權中斷
INTCONbits.T0IE = 1;             // 啟用 Timer0 中斷

IPR1bits.TMR1IP = 0;             // 設定 Timer1 為低優先權中斷
PIE1bits.TMR1IE = 1;             // 啟用 Timer1 中斷

RCONbits.IPEN = 1;               // 啟用高、低優先權控制機制
INTCONbits.GIEH = 1;             // 開啟高優先權中斷總控制位元
INTCONbits.GIEL = 1;             // 開啟低優先權中斷總控制位元
```

PIC18F 高、低優先權中斷練習

- 在 **MPALB X IDE**下開啟底下的專案
  - **\PIC18F Labs\Lab3.X**
- 程式執行後將可以看到
  - **LED0** 使用 **Timer0** 並設成高優先權中斷，每 **500mS** 閃爍一次
  - **LED7** 使用 **Timer1** 並設成低優先權中斷，每 **250mS** 閃爍一次

MICROCHIP

Regional Training Centers

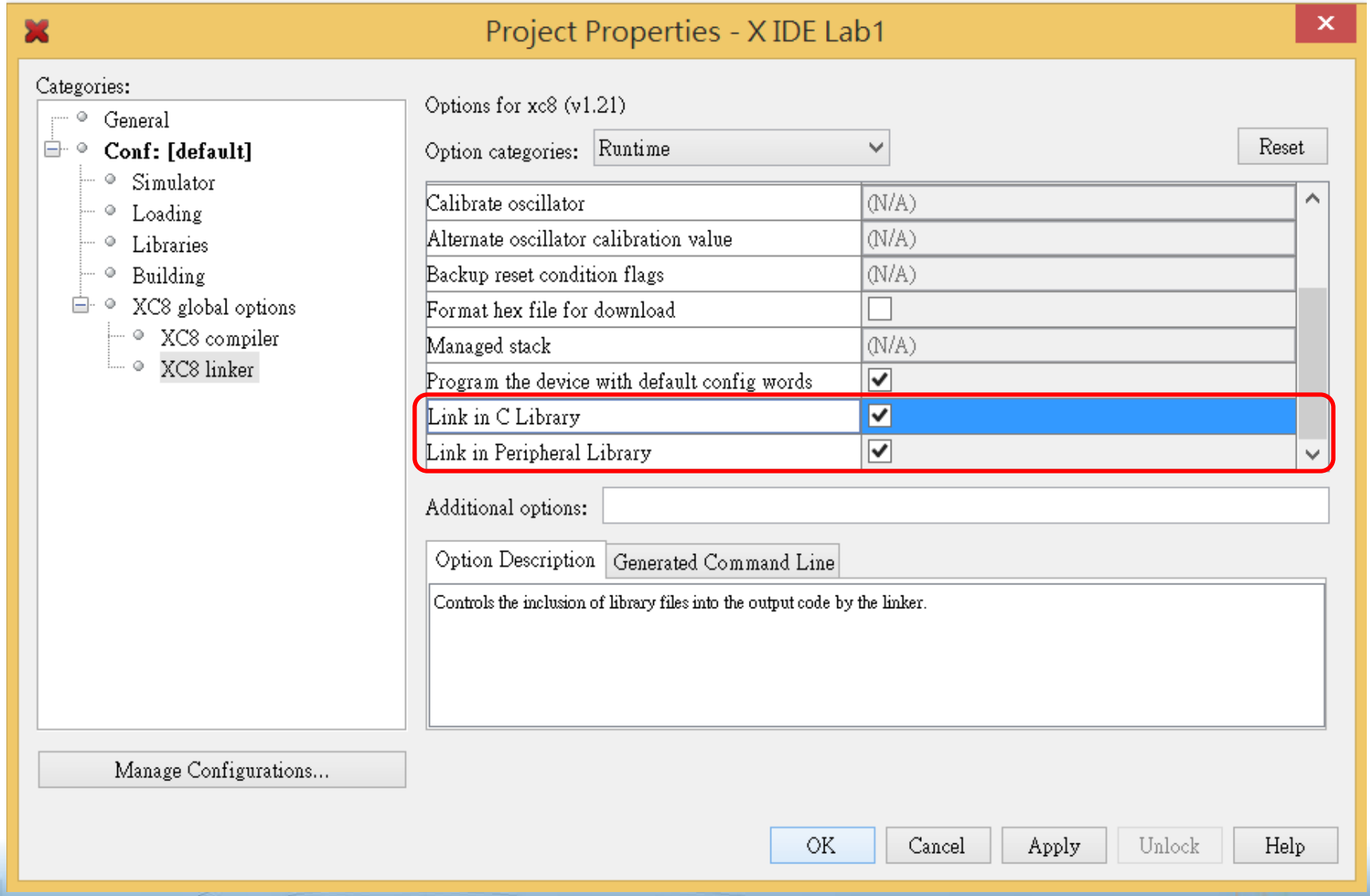
XC8 函數庫

了解 XC8 裡的對函數庫的支援

C Library

Peripheral Library

Linker 加入 CLIB & PLIB

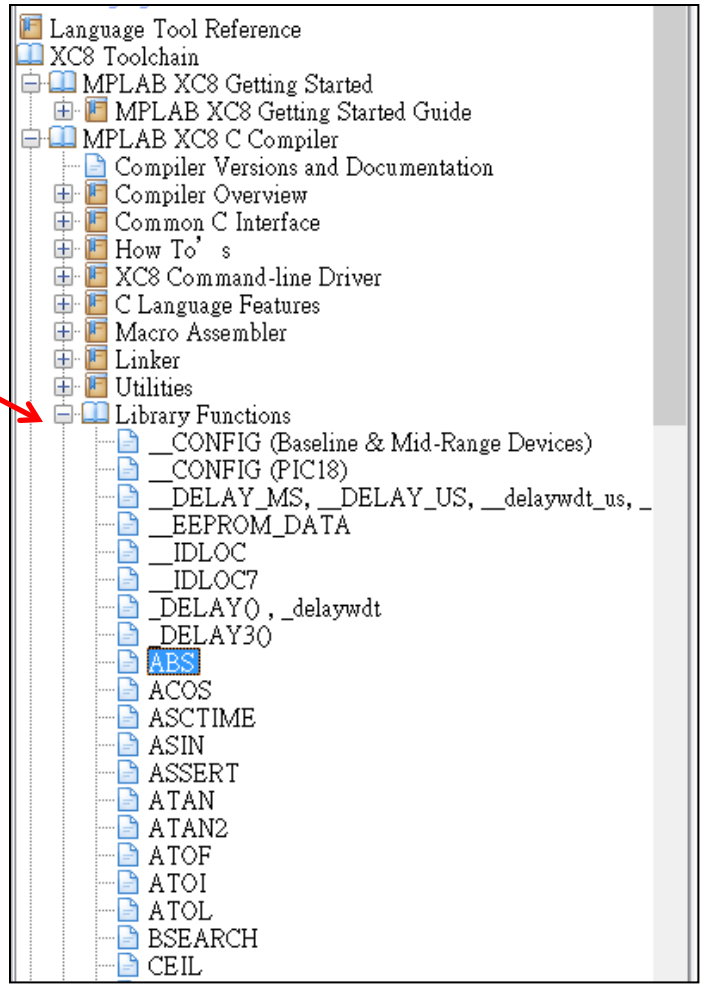


C Library

- **CLIB 提供 ANSI C 標準函數庫**
  - Delay Function
    - __DELAY_MS, __DELAY_US, __DELAYWDT_US, __DELAYWDT_MS
  - 數學運算 : #include <math.h>
    - sin( ), cos( ), sqrt( ), exp( )
  - 轉換函數 : #include <stdlib.h>
    - ltoa( ), itoa( ), ftoa( ),atof( ), atoi( )
  - 標準輸入\ 輸出 : #include <stdio.h>
    - printf( ), fgets( ), freopen( ), puts( )
  - 還有 ….

查詢 CLIB 的函數

- 在 **MPLAB X IDE** 下，點選 **Help** 選向下的 **“Help Contents” (建議使用)**
  - 開啟 **MPLAB XC8 C Compiler** 下的 **Library Functions (如右圖)**



- 使用 **pdf** 查詢，開啟 **XC8** 的使用手冊 **MPLAB_XC8_C_Compiler_User_Guide .pdf**，在 **Appendix A. Library Functions** 的章節有個別函數的使用方法

XC8 周邊函數庫 (PLIB)

- 目前 **XC8** 只支援 **PIC18F**系列的周邊函數庫，使用方式與 **C18** 幾乎雷同
- **PIC18** 函數庫參考手冊
  - C:\Program Files (x86)\ Microchip\xc8\v1.21 \docs\MPLAB_XC8_Peripheral_Libraries.pdf

使用 PIC18F 的周邊函數庫

- 開啟 **MPLAB_XC8_Peripheral_Libraries.pdf**
- 搜尋 **PIC18F4520** 這顆元件
  - ***CLICK HERE (see page 699) for the Peripheral Library Support Details for this Device***

右圖為 **18F4520** 的所有支援 **ADC** 的函數，點選所需的函數就可以看到所需的參數項及定義

Functions
OpenADC (see page 848)
SetChanADC (see page 882)
SelChanConvADC (see page 889)
ConvertADC (see page 896)
BusyADC (see page 897)
ReadADC (see page 897)
CloseADC (see page 897)

PIC16F 周邊函數庫

- **PIC16F 周邊函數庫？**
  - 目並未提供
  - 可以利用 PIC18F的原始碼來修改
  - C:\Program Files (x86)\Microchip\xc8\v1.21\sources\pic18\plib 目錄下
    - 例如：\ADC\adcopen.c

adcopen.c 原始程式

```
#include <p18cxxx.h>
#include <adc.h>

#if defined (ADC_V1)
void OpenADC( unsigned char config, unsigned char config2)
:
#elif defined (ADC_V2)
void OpenADC( unsigned char config, unsigned char config2)
:
#elif defined (ADC_V4) || defined (ADC_V5) || defined (ADC_V6)
void OpenADC( unsigned char config, unsigned char config2, unsigned char portconfig)
:
:
#elif defined (ADC_V15) || defined (ADC_V15_1)
void OpenADC( unsigned char config, unsigned char config1, unsigned char config2,
        unsigned char config3, unsigned char config4, unsigned char config5,
        unsigned char config6, unsigned char config7)
```

我怎樣知道 PIC 周邊的版本

- 要使用 **PIC18** 的周邊函數庫的原始程式必須知道該周邊的版本
  - 你知道 PIC18F4520 的 ADC 在 adccopen.c 是屬於哪一個版本?
    - 答: v5 版本的 ADC
  - 那 PIC18F4520 的 USART ?
    - 答: v4 版本

讓 pconfig.h 告訴您

- 開啟 \include\pconfig.h 檔案
  - 搜尋 “PIC18F4520”

```
#ifdef   __18F4520
/*############################################################*/
/*        Configuration for device =  'PIC18F4520'          */
/*############################################################*/
/* ADC */
#define ADC_V5
/* USART */
#define EAUSART_V4
/* SPI */
#define SPI_V1
```

MICROCHIP

Regional Training Centers

Mixing C and Assembly

XC8 專案下使用組合語言

- 組合語言可以被使用 **XC8** 的專案裡：
  - 獨立式組合語言
    - 使用**.as** 或 **.asm** 附檔名稱
  - **C** 的嵌入式組合語言
    - **asm(“*instruction*”);** 用於單行的嵌入式組語
    - **#asm … #endasm** ；區塊式的嵌入式組語

底下範例為兩種嵌入式組合語言撰寫的方式

```
asm(“BCF 0,3”);
asm(“BANKSEL _var”);
asm(“RLF (_var)&07fh”);
asm(“RLF (_var+1)&07fh”);
```

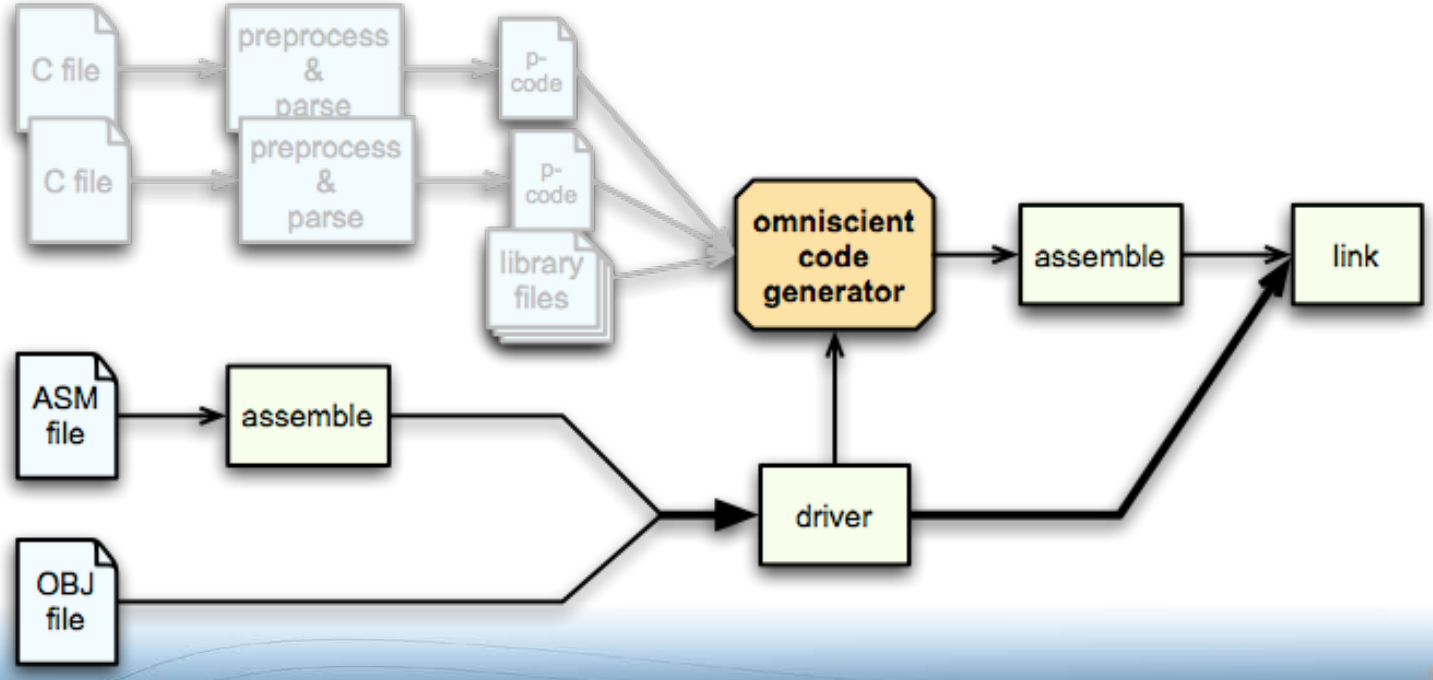
```
#asm
        BCF 0,3
        BANKSEL(_var)
        RLF (_var)&07fh
        RLF (_var+1)&07fh
#endasm
```

C 專案下的組合語言

- `#asm` … **#endasm** 這個組語的區塊將不視為 **C** 程式的一部分
  - 組語裡不要再使用 loops 或 conditionals, e.g. `if`, `do`, `for`, `while`.
- 使用嵌入式組語有些暫存器的狀態必須給予保存:
  - **WREG**
  - **FSR**
  - **STATUS**
  - **PCLATH;** or
  - scratch variables (**btemp**)

C-assembly Interaction

- **Prescan of assembly/obj modules detects:**
  - Symbols referenced
  - Memory requirements
  - Function calls



嵌入式巨集

- 嵌入式巨集定義在 **pic.h** 或 **pic18.h**
  - NOP()
  - SLEEP()
  - CLRWDT()
- **Base-Line** 的特殊指令會自動使用 **TRIS** 和 **OPTION** 暫存器

Example of C code accessing the OPTION register and assembly output

```
OPTION = 99     | 75   1EC   C63      movlw   99
                | 76   1ED   002      option
```

C 的變數使用在組合語言

- **確定將使用何種變數型態：**
  - 這變數是 global, auto 或 parameter?
- **變數的資料型別有多少 bytes?**
  - 編譯後的資料擺放方式是 little endian
    - LSB 位元組永遠被擺在最低的位址
- **如果變數不是存放在 bank0 ?**
  - 我將如何正確的切換 bank?

組合語言去看 C 的變數

- C 的變數型態必須是 Global(變數 x)

```
int x;

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

- 組合語言使用 C 的變數 x 時:
  - 看到的變數名稱需多加“_”，如：_x
  - 需在組語下使用 GLOBAL 虛指令來宣告
  - 考慮使用 bank 選擇的巨集 (banksel & bankmask)
  - 加入組語型態的暫存器定義檔<xc.inc>
    - pic.inc 或 pic18.inc
      - ✓ pic_chip_select.inc 或 pic18_chip_select.inc

組合語言範例

這個組語副程式會將 16-bit 變數 _x 的值加一後返回

```
#include <xc.inc>

GLOBAL  _x, _inc_word // 變數 x 及副程式名稱
                      // inc_word 為全域性

_inc_word:
    BANKSEL _x              ;select bank of _x
    incf  BANKMASK(_x)      ;mask address
    btfsc ZERO              ;or use STATUS,2
    incf  BANKMASK(_x+1)    ;access MSB
    return
```

潛在的問題

- 不正確的 **bank** 的切換或未設定 **bank** 的切換，編譯器是無法偵測到 其錯誤的 **(banksel ???)**
- 對於未使用 **masked address** 其結果也許會有錯誤
  - 如果變數 _x 的 MSB & LSB 不在同一 bank 的話

```
incf _x     ; no address masking!!!
```

C 呼叫組合語言的函數

- 確定有正確的雛型宣告 (prototype)
- 確定組合語言的語法符合 C 的用法

C 程式去呼叫組合語言的副程式 的前置作業範例

```
void inc_word(void);

int x;

void main(void){
  inc_word();       // this will increment x
}
```

MICROCHIP
Regional Training Centers

Questions?

MICROCHIP

Regional Training Centers

Thank You!

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, chipKIT, chipKIT logo, 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, 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.