HI-TECH PICC-16
C Compiler Workshop

課程說明

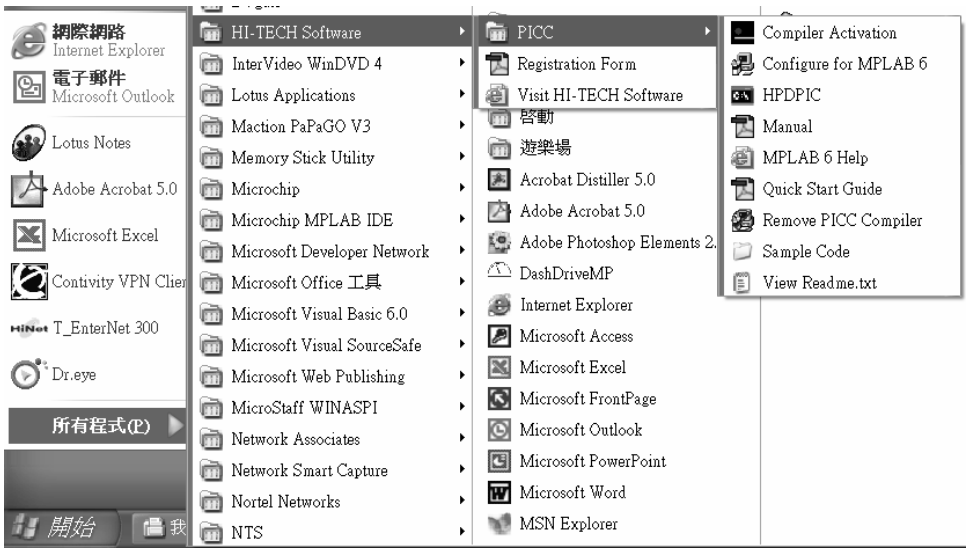
- 此教育訓練課程不是 C 的基礎課程
- 我們認定你已經有一定使用 C 的能力
  - 使用過 MPLAB C18
  - 使用過 ANSI C 或其它的 C 語言
- 本課程不在介紹如何撰寫 C 程式，而是介紹如何使用 Hi-Tech PICC
- 本教育訓練使用元件爲 PIC16F877A

Compiler Overview

MICROCHIP

Compiler Overview

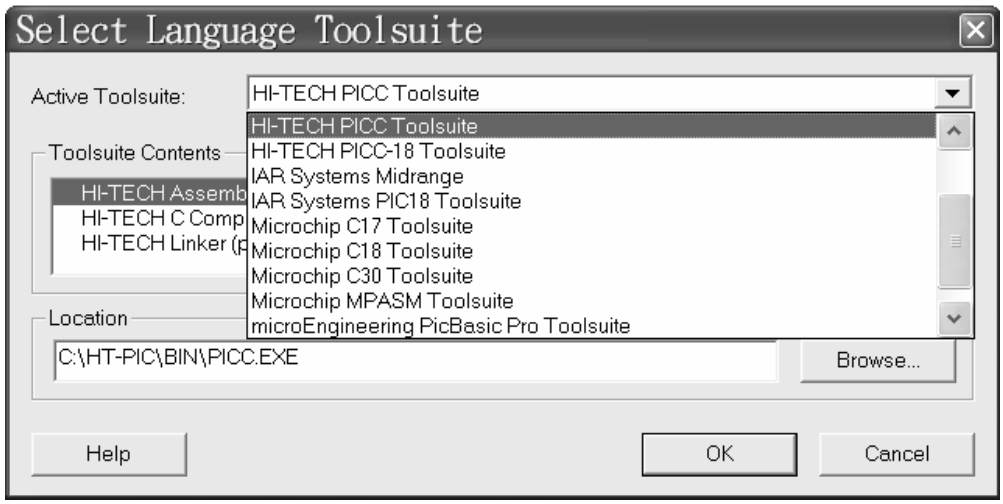
- 試用版的軟體並無功能上的限制
- 試用版的軟體有 21 天的使用時間限制
  - 使用 Compiler Activation 來作永久使用的設定



MICROCHIP

MPLAB IDE 使用 Hi-Tech PICC

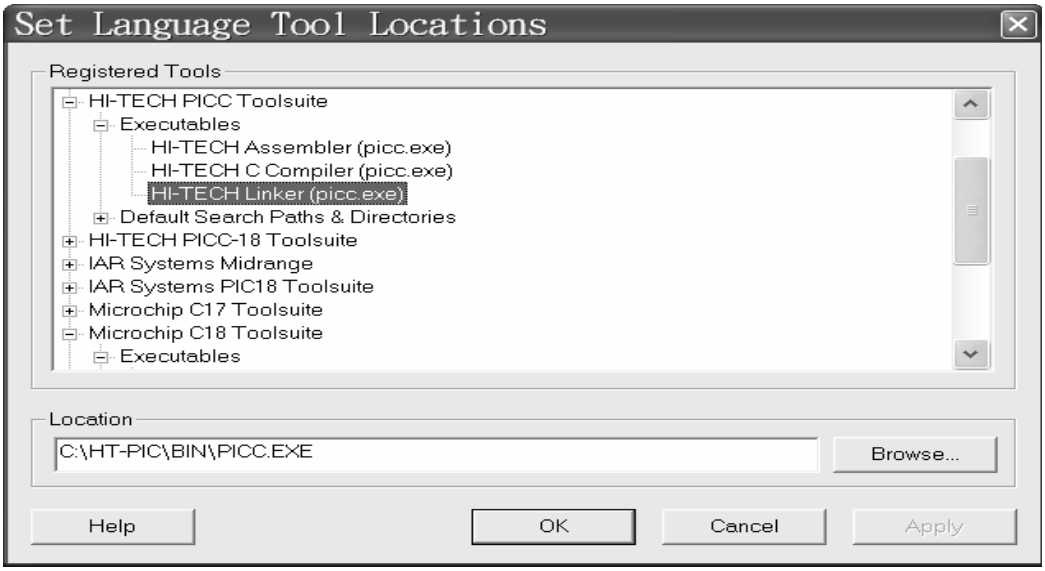
1. 使用 **“Project → Select Language Toolsuite”** 來設定語言工具爲 **HI-TECH PICC**
   - 最好安裝於預設的路徑 **C:\HT-PIC**



MICROCHIP

MPLAB IDE 使用 Hi-Tech PICC

2. 使用 **“Project → Set Language Tool Locations”** 設定 **HI-TECH PICC** 在 **MPLAB IDE** 的執行路徑
   - **C:\HT-PIC\BIN\PICC.exe**



MICROCHIP

PICC 介紹（一）

- 在 Hi-Tech 的開發環境下有兩種支援
  - PICC - command line driver (CLD)
    - CLD 可以被視窗的應用程式呼叫
      - 例如 : MPLAB IDE
  - HPDPIC - Hi-Tech IDE
- PICC 編譯器透過適當的命令列功能選項，可以接受原始程式的檔案進行編譯動作
- 有兩種原始程式可以被傳送
  - C 原始程式檔案 – 附加檔名爲 .c
  - 組合語言程式檔案 – 附加檔名爲 .as

MICROCHIP

PICC 介紹（二）

- PICC 也可將檔案加以連結
  - Relocatable object files – 附加檔名爲 .obj
  - Library files – 附加檔名爲 .lib
- Hi-Tech PICC 爲 ANSI C 相容語法，系統容易攜帶與程式轉移

MICROCHIP

PICC 執行檔的說明(一)

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

PICC 執行檔的說明(二)

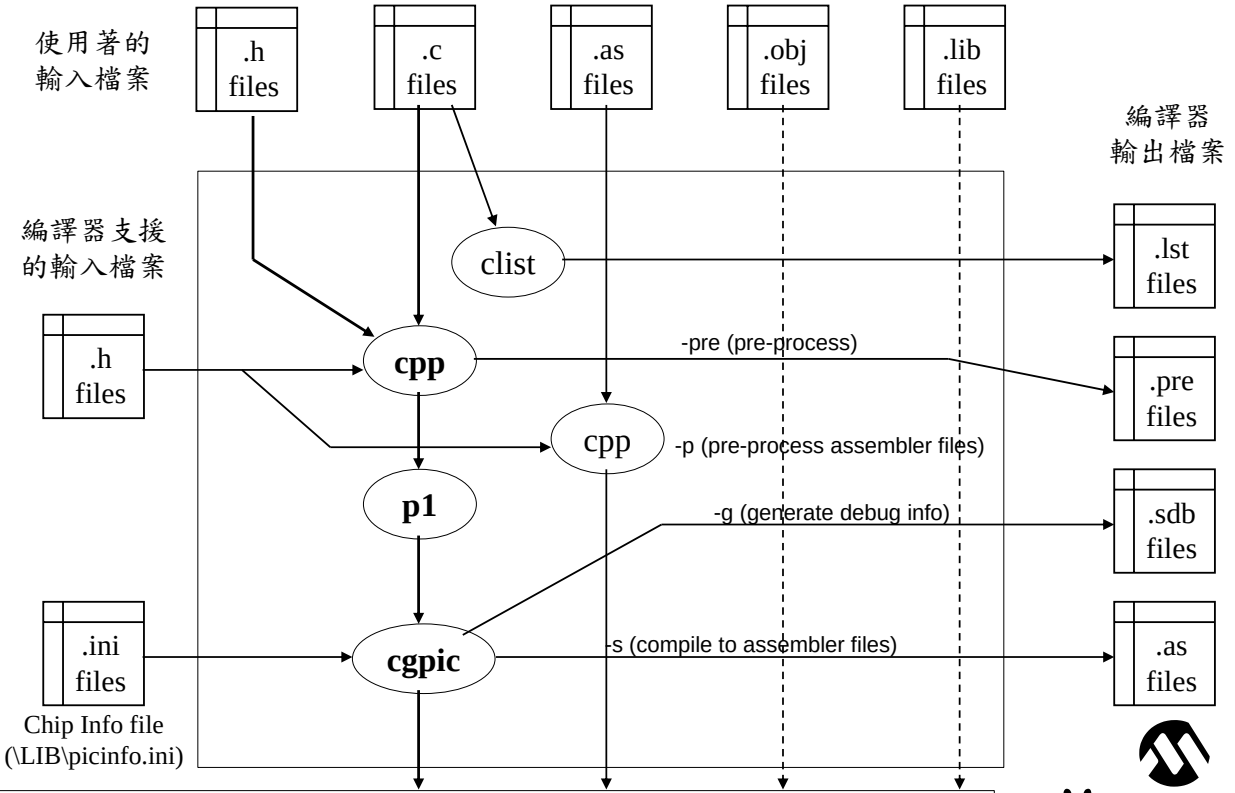
- CGPIC - The code generator
  - 將原始程式轉換成助憶符號的組合語言型態，其內容包含各個獨立的程式節區
- ASPIC - The assembler
  - 將組合語言轉換成可重新定位址的目的 (Relocatable object file)
- HLINK - The linker
  - 安排變數在 RAM 的實際位址
  - 聯結資料庫，並將各個獨立的程式節區的目的碼重新排定執行位址

PICC 執行檔的說明(三)

- OBJTOHEX - The output file converter
  - 產生一個可執行的 Hex 檔案
- CROMWELL - The re-formatter
  - 產生原始程式除錯的相關訊息檔案

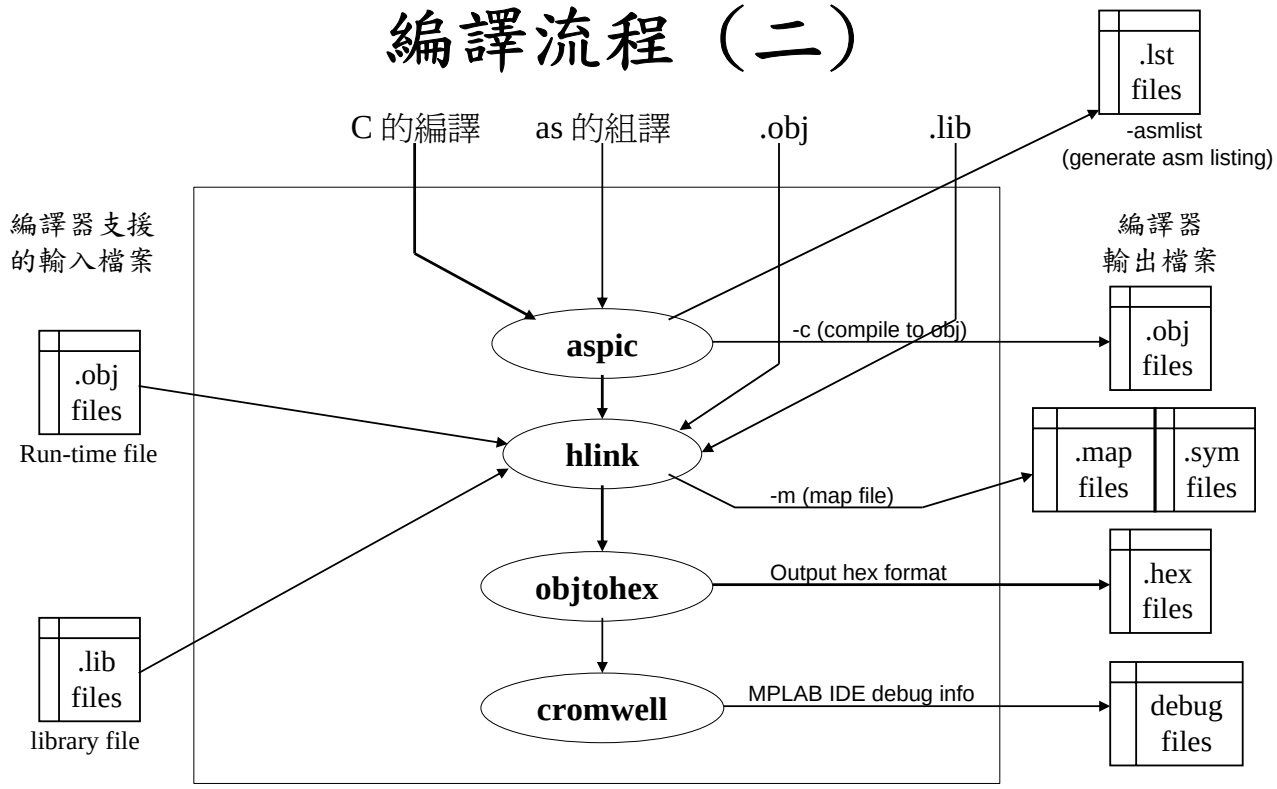
MICROCHIP

編譯流程（一）



MICROCHIP

編譯流程（二）



MICROCHIP

Hi-Tech PICC 輸入檔案

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

MICROCHIP

其它檔案說明

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

MICROCHIP

特殊功能暫存器 (SFR)

MICROCHIP

特殊功能暫存器

- C 可以直接定址到所有的 SFR
  - C 可以使用 @ 的絕對定址符號定義SFR的位址
  - 使用此種宣告方式，該位址並不會被保留
  - 一但使用上述的定義後，在 C 程式中就可以直接存取 SFR 暫存器
  - 有關 PICmicro 的特殊功能暫存器的定義內容已包含在編譯器所提供的 header files 裡
    - 使用時需將<pic.h>加入程式中
    - 它會依據你所選擇的 PICmicro 元件使用該元件的 Header File

MICROCHIP

PIC.h

- pic.h 的檔案在哪裡? (C:\HT-PIC\include\pic.h)

```
#ifndef     _PIC_H
#define     _PIC_H
#if  defined(_12C508)    || defined(_12C509)    ||\
     defined(_12C508A)   || defined(_12C509A)   ||\
     defined(_12CE518)   || defined(_12CE519)   ||\
     defined(_12C509AG)  || defined(_12C509AF)  ||\
     defined(_12CR509A)  || defined(_RF509AG)   ||\
     defined(_RF509AF)
     #include  <pic125xx.h>
#endif
#if  defined(_16C432)    || defined(_16C433)
     #include  <pic1643x.h>
#endif
#if  defined(_16C52)     || defined(_16C54)     || defined(_16C54A)    ||\
     defined(_16C54B)    || defined(_16C54C)    || defined(_16CR54A)   ||\
     defined(_16CR54B)   || defined(_16CR54C)   || defined(_16C55)     ||\
     defined(_16C55A)    || defined(_16C56)     || defined(_16C56A)    ||\
     defined(_16CR56A)   || defined(_16C57)     || defined(_16C57C)    ||\
     defined(_16CR57B)   || defined(_16CR57C)   || defined(_16C58A)    ||\
     defined(_16C58B)    || defined(_16CR58A)   || defined(_16CR58B)   ||\
     defined(_16C58)     || defined(_16HV540)
     #include  <pic165x.h>
#endif
```

PIC.h 依據所選定的元件來決定使用那一個元件的定義檔案

MICROCHIP

元件的 PIC168xA.h

```
/*
 *      Header file for the Microchip
 *      PIC 16F873A chip
 *      PIC 16F874A chip
 *      PIC 16F876A chip
 *      PIC 16F877A chip
 *      Midrange Microcontroller
 */
#if defined(_16F874A) || defined(_16F877A)
#define     __PINS_40
#endif

static volatile unsigned char       INDF    @ 0x00;
static volatile unsigned char       TMR0    @ 0x01;
static volatile unsigned char       PCL     @ 0x02;
static volatile unsigned char       STATUS  @ 0x03;
static       unsigned char          FSR     @ 0x04;
static volatile unsigned char       PORTA   @ 0x05;
static volatile unsigned char       PORTB   @ 0x06;
static volatile unsigned char       PORTC   @ 0x07;
#ifdef __PINS_40
static volatile unsigned char       PORTD   @ 0x08;
static volatile unsigned char       PORTE   @ 0x09;
#endif
```

選定PIC16F877A的元件後，暫存器的定義將會使用PIC168xA.h的宣告

MICROCHIP

練習一
Hi-Tech PICC 工作環境設定

- 設定 Hi-TECH PICC 在 MPLAB IDE 的工作環境
  - 建立一個 Hi-Tech PICC 工作項目(Project)
    - 程式 : Lab1.c
    - Head files : pic.h , config877a.h
  - 使用元件爲 PIC17F877A
  - 語言工具是否設定正確
  - MPLAB ICD2 是否能正常執行

MICROCHIP

練習一
Hi-Tech PICC思考問題

- 了解 pic.h 是如何找出正確的 PIC168xA.h
- pic.h 是否還有其它的巨集定義(macro) ?
- 函數是否需要做原型宣告 ?
- Delay_1mS 是如何得知爲 1mS Delay ?

MICROCHIP

資料型態
變數種類
I/O 的定址

MICROCHIP

資料型態

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

A. A char is unsigned by default.
B. A double defaults to 24-bit. Easily changed to 32 bits within the HPDPIC, PICC command line interface or within MPLAB-IDE.

Floating point is implemented using the **IEEE 754** 32-bit.

MICROCHIP

變數的等級

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

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

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

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

MICROCHIP

變數的型態

1. *Const* - const 的保留字是用來告訴編譯器該內容的資料型態爲固定不變的常數值。任何企圖修改這個常數值的動作編譯器將會產生警告訊息。

2. *Volatile* - volatile 的保留字是用來告訴編譯器該變數的內容不一定要經過程式的執行後才會改變。它可以告訴編譯器在做最佳化處理的時候無需將該變數簡化。所有的輸入 / 輸出暫存器及中斷所使用到的變數均需宣告成非揮發性變數型態（volatile)。

MICROCHIP

特殊的變數型態

1. *Persistent* - 爲一個特殊的保留字，用來告訴編譯器該變數在重新啓動時(startup)無須被清除爲零。一般而言，使用此方式所宣告的變數會被存放在另外的區域 (for example, the *nvram* or *nvram_1* psects)

2. 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。

MICROCHIP

絕對位址變數

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

範例 :

```
volatile unsigned char PORTB @ 0x06
```


MICROCHIP

絕對定址使用範例

- 定義特殊暫存器(FSR)的絕對位址的範例 (PIC168xA.H)

```
/*  bank 0 registers */
    static volatile unsigned char INDF          @ 0x00;
    static volatile unsigned char TMR0          @ 0x01;
    static volatile unsigned char PCL           @ 0x02;
    static volatile unsigned char STATUS        @ 0x03;
    static          unsigned char FSR           @ 0x04;
                    :
/*  bank 1 registers */
    static          unsigned char bank1  OPTION @ 0x81;
    static volatile unsigned char bank1  TRISA  @ 0x85;
    static volatile unsigned char bank1  TRISB  @ 0x86;
    static volatile unsigned char bank1  TRISC  @ 0x87;
                    :
```


MICROCHIP

位元（bit）變數定址能力

- 與絕對位址變數共用
  - Hi-Tech C 的專用語法
  - 一般用來定義特殊暫存器(SFR)的獨立位元
  - 特殊暫存器(SFR)內的位元定義 – PIC168xA.h
- 使用 bit 定義位元變數
  - Hi-Tech C 的專用語法
  - 最簡單方便的使用方式
- 位元結構方式
  - ANSI C 標準使用方式

MICROCHIP

位元（bit）變數定址 – 與絕對位址變數共用

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

範例 : 存取 Z 旗號

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

存取 PORTA 的 RA5

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


MICROCHIP

位元（bit）變數定址
－與絕對位址變數共用

- HI-TECH C 已將各相關的位元名稱及相關位置定義完成 (PIC168xA.h)
  - **/* STATUS bits */**
  - **static volatile bit IRP @ (unsigned)&STATUS*8+7;**
  - **static volatile bit RP1 @ (unsigned)&STATUS*8+6;**
  - **static volatile bit RP0 @ (unsigned)&STATUS*8+5;**
  - **static volatile bit TO @ (unsigned)&STATUS*8+4;**
  - **static volatile bit PD @ (unsigned)&STATUS*8+3;**
  - **static volatile bit ZERO @ (unsigned)&STATUS*8+2;**
  - **static volatile bit DC @ (unsigned)&STATUS*8+1;**
  - **static volatile bit CARRY @ (unsigned)&STATUS*8+0;**
- 若要判斷 Z 旗標，只要
  - If (ZERO) ……

MICROCHIP

位元（bit）變數定址
－ 使用 bit 定義位元變數

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

範例 ： 利用 bit 定義位元變數

```
static bit Count_Flag ;
static bit Buzzer_1_Flag ;

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


MICROCHIP

準備練習二
關於 LCD 顯示函數

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

MICROCHIP

練習二
基本I/O的設定

- 本練習請開啓 lab2.mcp
  - 請使用 PIC168xA.h 所提供的FSR暫存器定義名稱，修改 I/O Port 以符合 Hi-Tech PICC 的語法
    - lab2.c : 主程式
    - mid_lcd.c , mid_lcd.h : LCD顯示函數
    - cnfig877a.h : 16F877A configuration setting
  - 按下 SW2 或 SW3時， LED會顯示加一或減一
  - LCD模組也會顯示如下的字元

```
Hi-Tech PICC Ex2
Up: 01   Down: FE
```


MICROCHIP

結構變數

共用變數

位元結構

MICROCHIP

結構型態（Structures）

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

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

結構名稱

變數成員

結構變數

MICROCHIP

使用基本的結構變數

- 欲使用結構內的成員，可使用“.”來組成:
  variable-name.member*x* （結構變數.成員)

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

Comm_protocol 在 **RAM** 的排列

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

MICROCHIP

存取結構中的陣列（一）

範例 1 :

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

成員的存取使用小數點的運算符號“ . “

例如：

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

“***Fbandgap”*** *陣列的第二個元素會被存入* **0x34** 的值.

MICROCHIP

存取結構中的陣列（二）
使用結構指標

範例 2 ：

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

使用範例 **1** 所定義過的結構名稱 **“filtered_data”**

宣告: **struct filtered_data *ptr;**

ptr = &Fcount;

ptr->Frefhi[0] = 0x87;

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

MICROCHIP

在宣告同時指定結構變數的初始值

範例 3 ：

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


MICROCHIP

以結構為元素的陣列

範例 4 :

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

存取結構陣列內成員爲 **"*mode*"** 的第一個陣列值:

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

存取結構陣列內成員爲 **"*sign*"** 的第三個陣列值:

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


MICROCHIP

共用型態（union）

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

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

共用型態名稱

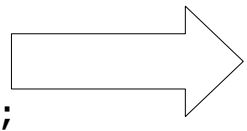
共用元素成員

共用型態變數

MICROCHIP

共用型態的資料架構

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



Word	
Byte[0]	Byte[1]
RAM0	RAM1

EE_tag 佔 2 個 **Bytes**

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

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

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

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

MICROCHIP

共用型態和成員的存取（一）

範例 1 :

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

成員的存取使用 **‘.’** 運算元

- ***utemp.abc*** **= ‘A’;**
- ***utemp.value*** **= 0x3456;**

範例 2 :

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

成員的存取使用 **‘->’** 運算元

- ***uptr->value*** **= 0x5678;**

MICROCHIP

共用型態和成員的存取（二）

- 一個結構型態或共用型態的宣告內，也可含有其它的共用型態或結構型態

範例 : 在整數(16-bit)中單獨存取其中一個 8-bit 值

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

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


MICROCHIP

合併使用 結構型態 & 共用型態

- 在浮點數中單獨取出指數部分

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

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


MICROCHIP

位元結構

範例 **1 :** 位元結構型態 **– Byte** 大小

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

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

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

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

MICROCHIP

位元結構

範例 **2 :** 位元結構型態 **– Word** 大小

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

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

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

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

MICROCHIP

位元的使用範例(一)

範例 1:

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

現在你可以看到：

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

MICROCHIP

位元的使用範例(二)

範例 2:

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

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

```
Flow = 1;
```

範例 3:

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

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

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


MICROCHIP

位元的使用範例(三)

範例 **4:**

你也可以採用巨集的方式宣告**:**

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

然後就可以使用下列的宣告方式指定任何暫存器的任一位元定址：

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

一般較常用的方式是用 **#define**

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


MICROCHIP

位元的使用範例(四)

範例 **5:**

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

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

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


MICROCHIP

準備練習三
重要函數說名

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

MICROCHIP

練習三

- 練習三是利用 10-bit A/D 轉換結果利用 union 和 struct 來拆解資料
  - 10-bit 的數值會轉換成 10 進制並做居先零的抑制後顯示在 LCD Module
  - 另將 A/D 拆成兩個 8-bit 的資料，分別顯示在LCD Module 的第二行
- 有關 A/D 供能請參考後面有關 A/D Module 的說明或參考 Data Sheet
- 練習試著使用 bit 來設定單一位元旗號

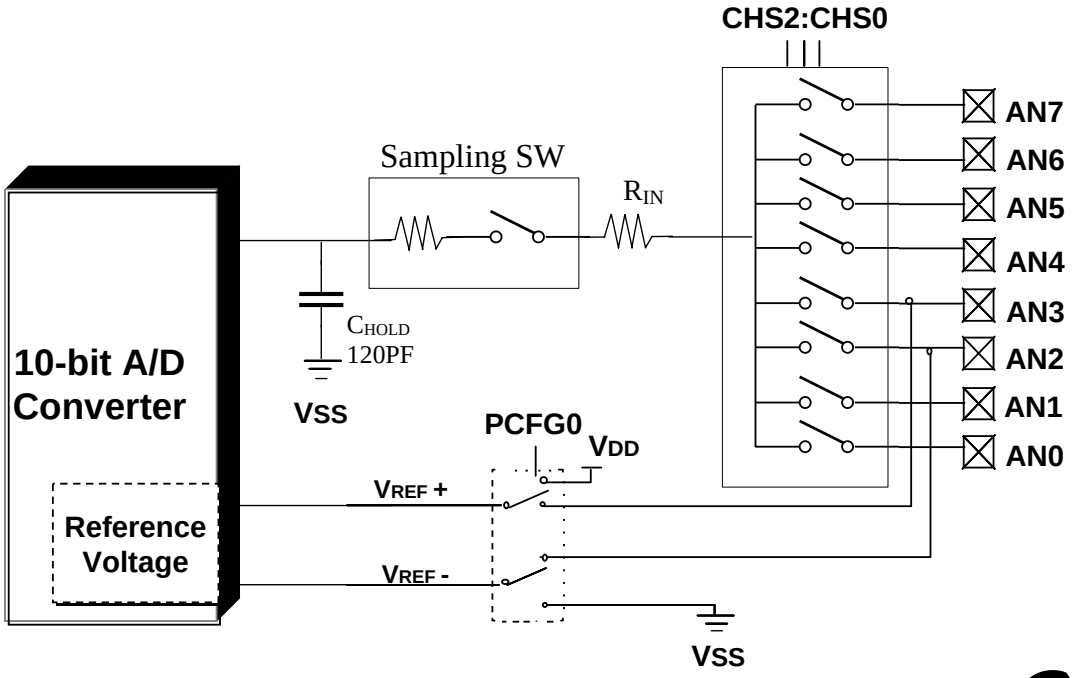
MICROCHIP

10-bit A/D 轉換器

- 8組類比轉換多工輸入選擇，10 bits 解析度
- 類比輸入取樣時間：20 µS (輸入阻抗<10K)
- 類比輸入轉換時間：19.2 µS (12 T_{AD})
- 10-bit 解析度時，只有一位元的誤差
- 允許使用外部參考電壓： VREF+ & VREF-
- 轉換的結果允許自動向左、向右對齊修正
- 完整的轉換時間共須 39.2 µs
  - 如輸入腳位固定，其轉換時間只需：29.2 µs

MICROCHIP

10-bit A/D 方塊圖



MICROCHIP

A/D 控制暫存器

See Data Book

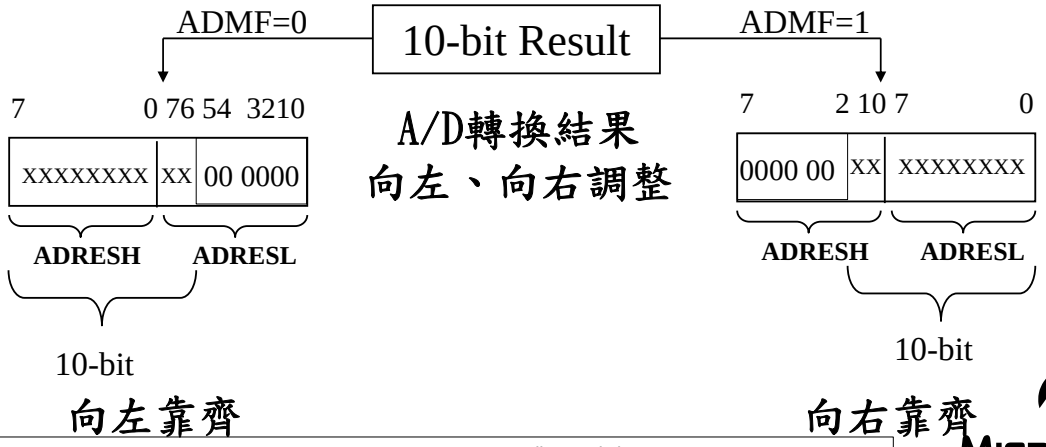
ADCON0 Register

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

bit7 bit0

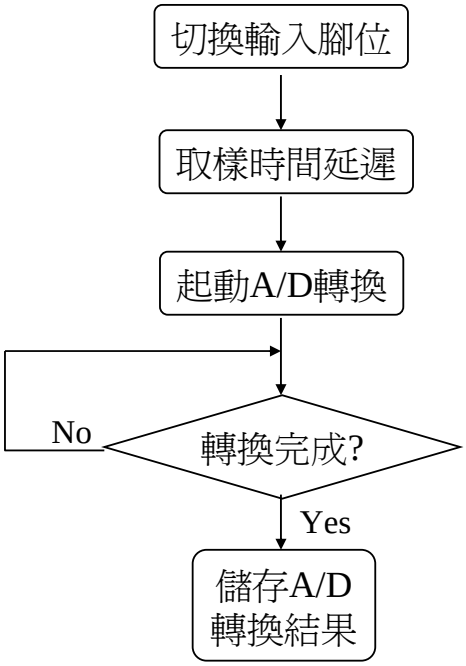
ADCON1 Register

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



MICROCHIP

A/D 轉換基本流程



MICROCHIP

RAM & ROM 形態的指標和陣列

MICROCHIP

使用指標

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

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

MICROCHIP

ROM & RAM 指標

- 各種型態的指標保留字可擴展指標對不同的記憶體存取
  - const 保留字允許指標對 ROM 作業
  - Bankn 保留字允許指標對不同的 RAM banks 存取 (n 爲數字 1 – 2)
  - Pointers to volatile-qualified objects may use different sequences of code to access objects

MICROCHIP

常見錯誤指標用法（一）

- 將位址指定給不合型態的指標將帶來無法預估的災難

範例 1：錯誤的 ROM 指標例

```
const char array[ ] = {0, 2, 4, 6};
char * cp, c;
cp =  array;
c = * cp;                    // *cp will access RAM not ROM
```

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


MICROCHIP

常見錯誤指標用法（二）

範例 2： 錯誤的 bank 宣告例

```
void process( int * ip );  // 錯誤的宣告

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

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


MICROCHIP

指標存放資料格式

- 對資料存放的方式採用低位址存放低數值格式 (little endian format)
  - Char 指標可以用來存取 multi-byte 中的任何一個 byte 資料

範例：

```
long input;
char * cp;
*((char *)&input + 3) = 0x55;  // write character to MSB of input
```


MICROCHIP

RAM 的指標 - Midrange PICmicro

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

MICROCHIP

RAM 的指標 - Midrange PICmicro

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

MICROCHIP

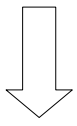
RAM 的指標 - Midrange PICmicro

- RAM 指標 – 我該使用何種型態的指標
  - 取決於使用何種 PICmicro (PIC16C5x, 16Fxx)
  - 取決於你的應用程式
  - 是否有提供 *far* 型態的指標？
    - 沒有，不過 ....

MICROCHIP

RAM 的指標 - Midrange PICmicro

- Use Bank2 or Bank3 Pointers
  - bank2 char *ramptr;
  - bank3 char *ramptr;
  - bank3 char * bank2 ramptr;
    - 這種宣告會有何結果?



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

MICROCHIP

RAM 用指標方式存取

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

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

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


MICROCHIP

RAM 用陣列方式存取

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

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

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

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


MICROCHIP

練習四
RAM 指標

- 修改 lab4.c 的程式，讓指標能夠正確指到對應的 RAM Bank (0 – 3)
  - 程式執行前，先清除所有的 RAM
    - Debugger → Clear Memory → File Register
  - 執行程式後，檢查指標變數存放的內容爲
    - aptr = IRP=0，0x20 , 內容值爲 0x61 (‘a’)
    - bptr = IRP=0，0xA0 , 內容值爲 0x62 (‘b’)
    - cptr = IRP:1， 0x10 , 內容值爲 0x63 (‘c’)
    - dptr = IRP:1， 0x90 , 內容值爲 0x64 (‘d’)

MICROCHIP

常數型態的指標
– Midrange PICmicro

- 常數型態的指標爲 16-bits 的位址範圍
  - 可以用來讀取所有的 RAM 資料及所有的 ROM 資料
  - 無法寫入資料到 RAM
    - 爲什麼?

MICROCHIP

ROM 指標的查表法

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

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

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

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


MICROCHIP

ROM 陣列的查表法

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

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

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

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


MICROCHIP

練習五
取得 ROM 陣列資料

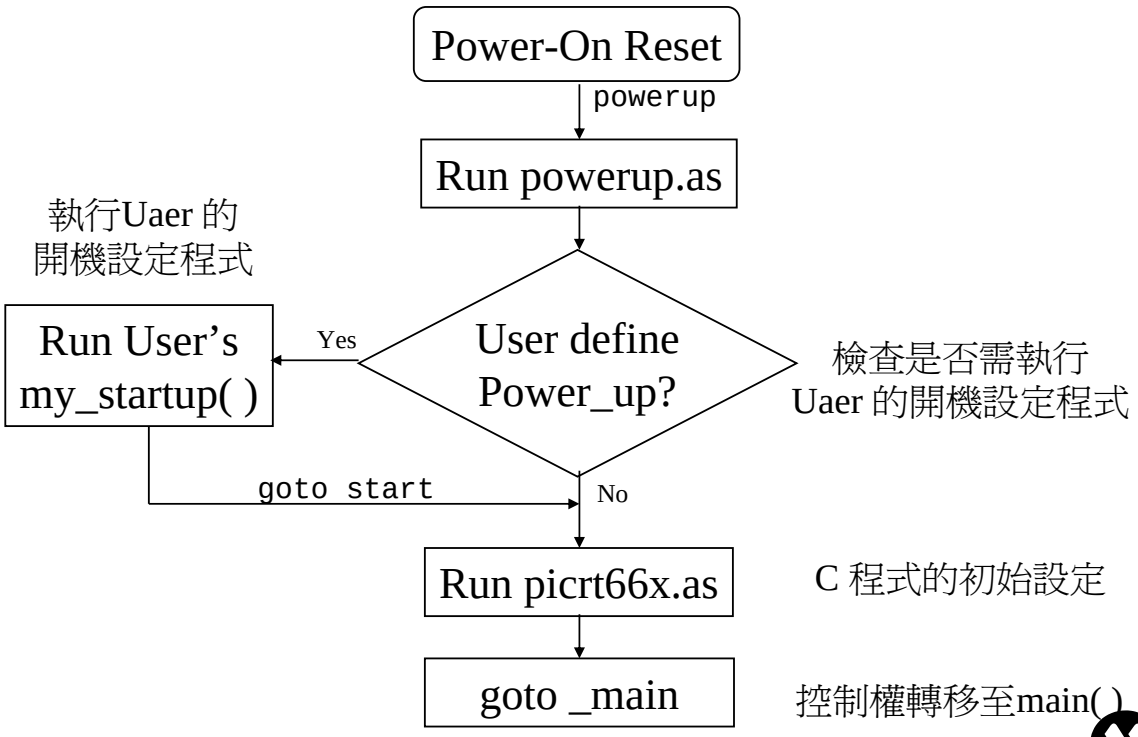
- 修改 lab5.c 的程式，讓程式能正確讀取 ROM 的陣列並填入 RAM 中
  - 程式執行前，先清除所有的 RAM
    - Debugger → Clear Memory → File Register
  - 執行程式後，檢查陣列資料是否被複製到
    - RAMarray1[ ] : 起始位址在 0x23
    - RAMarray2[ ] : 起始位址在 0x110 (bank2)

MICROCHIP

Hi-Tech PICC
重置後的初始設定

MICROCHIP

Reset 動作流程



MICROCHIP

啟動程式

- 重置位址在 0x00，函數名稱爲 ***powerup***
  - MCU Reset 後立即執行此程式
  - C:\HT-PIC\SOURCE\powerup.as
  - 可藉修改此程式，強制先執行I/O設定動作
- 啓動模組工作，函數名稱爲 ***reset***
  - 載入RC振盪器的校正值(有內建RC振盪器者)
  - 清除 bss psects (uninitialised data) 區域
  - 設定 data psects 的初始資料
  - 將程式控制權轉移至 main( )
  - 原始程式 : C:\HT-PIC\SOURCES\picrt66x.as

MICROCHIP

自行設定的啟動工作

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

MICROCHIP

I/O Initialization on Powerup - PIC16CXXX

C:

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

     movlw   high start
     movwf   _PCLATH
     goto    ( start &
0x7FF )
; asm (  "ljmp start " );
#endasm
}
```

Assembly:

```
     #include      "sfr.h"
extrn  _mystartup
      global  powerup, start
      psect
         powerup,class=CODE,delta=2

powerup
#if  defined (_12C508) || defined(_12C509)
       movwf  5
#endif
#if  defined(_PIC14)
       movlw   high _mystartup
       movwf   PCLATH
       goto     ( _mystartup & 0x7FF )
#endif
#if defined(_PIC16) . . . .
#endif
                    end powerup
```


MICROCHIP

中斷管理

MICROCHIP

中斷處理 – PIC16Fxxx

- 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
- 只能允許一個中斷函數
  - 無法傳遞參數到中斷函數
  - C 程式可在任何地方呼叫此函數

```
void interrupt isr_rec ( void )
{
        if ( RCIF )
          byte = RCREG;
}
```

中斷函數名稱

MICROCHIP

多個中斷處理

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

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


MICROCHIP

中斷參數的存取

- 中斷函數會自動存/取的值
  - 最重要的 W , STATUS , FSR , PCLATH
  - 中斷會自動儲存使用的暫存器
  - 嵌入式組合語言使用的 RAM 無法自動被儲存

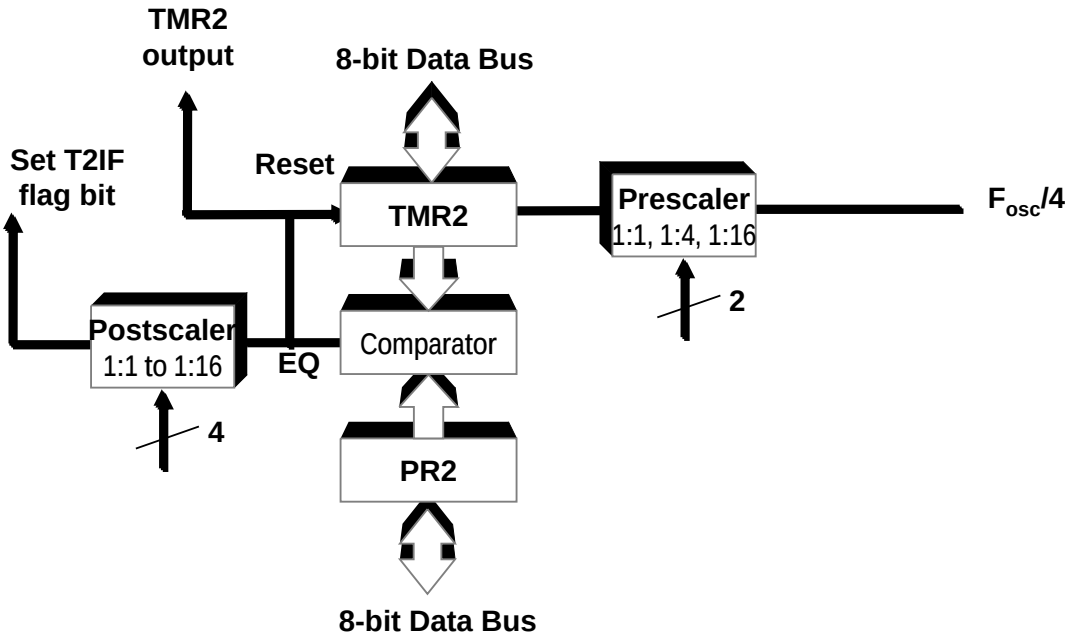
MICROCHIP

準備練習六
處理 Timer2 中斷

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

MICROCHIP

準備練習六
Timer2 方塊說名



MICROCHIP

準備練習六
Timer2 暫存器說明

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

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

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

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

- 中斷設定 : GIE , PEIE , TMR2IE
- 中斷旗號 : TMR2IF
- 中斷時間 : 0.25us *16 *16 * (155+1) = 9.984mS

MICROCHIP

練習六

- lab6.c 程式加入 Timer2 中斷處理功能，讓程式能夠每 10mS 中斷一次，每 100mS 執行跑馬燈的動作

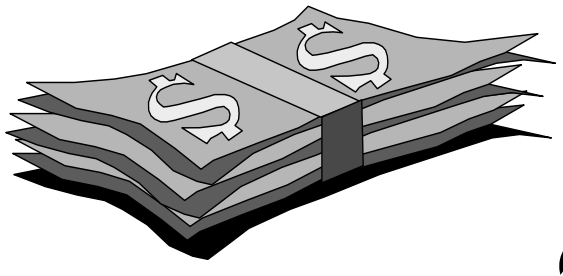
MICROCHIP

合併 組合語言與 C 語言

MICROCHIP

合併組合語言與 C 語言

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



MICROCHIP

合併組合語言與 C 語言

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

MICROCHIP

嵌入式的組合語言

- 嵌入式的組合語言可以直接以 C 語言的程序 (statements) 的方式直接加入程式中

```
Examples:
asm(“movlw  0x20”);      // C statement style
asm(“movwf  0x30”);
```

- 或使用 #asm , #endasm 的宣告

```
#asm
  movlw  0x10              // Assembler directive style
  movwf  0x20
#endasm
```


MICROCHIP

獨立式組合語言

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

MICROCHIP

獨立式組合語言

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

MICROCHIP

獨立式組合語言

- 組合語言與 C 語言可以同時使用函數或物件

在組合語言使用 C 的物件

```
int        response;              // a global C object

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

在 C 程式下使用組合語言的物件

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

extern    unsigned int answer;    // reference linkage
answer = 10;                      // accessing assembly object from C
```


MICROCHIP

獨立式組合語言

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

MICROCHIP

獨立式組合語言

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

MICROCHIP

獨立式組合語言

- 變數的存取

Assembly:

```
    processor 16C74A
    #include  "p16c74a.hti"

    psect text,global,class=CODE,delta=2
    global _Foo
    signat _Foo,88

    global _Var1,_Var2

_Foo
    movf _Var1,W
    addwf _Var2,W
    movwf (_Var2+1)
    return
    end
```

C:

```
#include <pic1674.h>

extern void Foo( void );

char Var1;
int Var2;

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


MICROCHIP

Hi-Tech PICC
所提供的資料庫

MICROCHIP

函數庫

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

MICROCHIP

函數庫

- 超過 60 個標準 C 函數庫
  - 字串及記憶體處理函數庫
    - strcpy
    - strcmp
    - strlen
    - strcat
    - memset
    - memcpy
    - memcmp
    - and more . . .

MICROCHIP

函數庫

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

MICROCHIP

函數庫

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

MICROCHIP

函數庫

- 其它功能函數庫
  - atof
  - atoi
  - atol
  - ldiv
  - rand
  - and more . . .

MICROCHIP

Hi-Tech PICC
其它功能

MICROCHIP

有用的巨集定義
PIC168xA.h

- 如何定義 …
  - #define sleep( ) asm(“sleep”)
  - #define nop( ) asm(“nop”)
  - #define clrwdt( ) asm(“clrwdt”)
    - 所以要進入睡眠省電模式只要輸入巨集指令
      - Sleep( );
- 另有兩個巨集指令可以開啓或關閉中斷
  - 在 C 程式下只要輸入
    - ei( );
    - di( );

MICROCHIP

C 程式下設定 Config.

- Configuration Fuses Set in C Code
  - 爲什麼？
    - 設定簡單
    - 程式的可讀性提高
    - 設定訊息存放在 .hex 檔案中
    - Hi-Tech 提供 “_ _CONFIG” 的巨集
    - 可以 #define 的宣告建立成 .h 的檔案
    - Hi-Tech PICC 已將 Config. Fuse 的定義

MICROCHIP

PIC16F877A.INC（組合語言）

```
_CP_ALL           EQU     H'1FFF'
_CP_OFF           EQU     H'3FFF'
_DEBUG_OFF        EQU     H'3FFF'
_DEBUG_ON         EQU     H'37FF'
_WRT_OFF          EQU     H'3FFF'    ; No prog memmory write protection
_WRT_256          EQU     H'3DFF'    ; First 256 prog memmory write
_WRT_1FOURTH      EQU     H'3BFF'    ; First quarter prog memmory write
_WRT_HALF         EQU     H'39FF'    ; First half memmory write protected
_CPD_OFF          EQU     H'3FFF'
_CPD_ON           EQU     H'3EFF'
_LVP_ON           EQU     H'3FFF'
_LVP_OFF          EQU     H'3F7F'
_BODEN_ON         EQU     H'3FFF'
_BODEN_OFF        EQU     H'3FBF'
_PWRTE_OFF        EQU     H'3FFF'
_PWRTE_ON         EQU     H'3FF7'
_WDT_ON           EQU     H'3FFF'
_WDT_OFF          EQU     H'3FFB'
_RC_OSC           EQU     H'3FFF'
_HS_OSC           EQU     H'3FFE'
_XT_OSC           EQU     H'3FFD'
_LP_OSC           EQU     H'3FFC'
```


MICROCHIP

Config 的設定 (pic168xa.h)

```
#define         CONFIG_ADDR     0x2007

#define         RC              0x3FFF          // resistor/capacitor
#define         HS              0x3FFE          // high speed crystal/resonator
#define         XT              0x3FFD          // crystal/resonator
#define         LP              0x3FFC          // low power crystal/resonator
#define         WDTEN           0x3FFF          // enable watchdog timer
#define         WDTDIS          0x3FFB          // disable watchdog timer
#define         PWRTEN          0x3FF7          // enable power up timer
#define         PWRTDIS         0x3FFF          // disable power up timer
#define         BOREN           0x3FFF          // enable brown out reset
#define         BORDIS          0x3FBF          // disable brown out reset
#define         LVPEN           0x3FFF          // LVP enabled
#define         LVPDIS          0x3F7F          // LVP disabled
#define         WRTEN           0x3FFF          // flash write enabled
#define         WP1             0x3DFF          // protect 0000 - 00FF
#define         WP2             0x3BFF          // protect 0000 - 07FF
#define         WP3             0x39FF          // protect 0000 - 1FFF
#define         PROTECT         0x1FFF          // protect program code
#define         UNPROTECT       0x3FFF          // do not protect the code
```


MICROCHIP

設定 Config. Word

- CONFIG 的巨集宣告 (pic.h)

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

  - 位元定義的巨集名稱不可弄錯

範例：**PIC16F877A**

```
#include <pic.h>

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

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


MICROCHIP

設定 ID Locations

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

範例：**PIC16F877A**

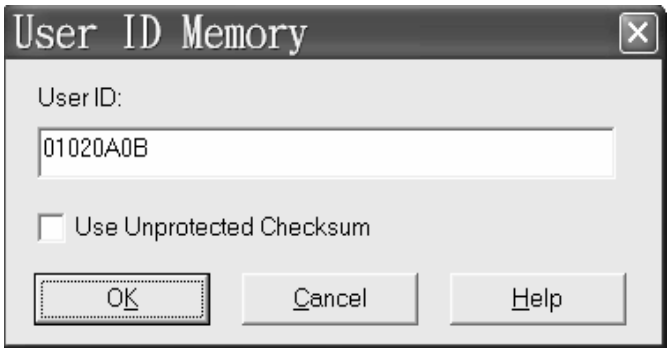
```
#include  <pic.h>

      __IDLOC  ( 12AB );

void main(void)
{

        //      ... your code

}
```



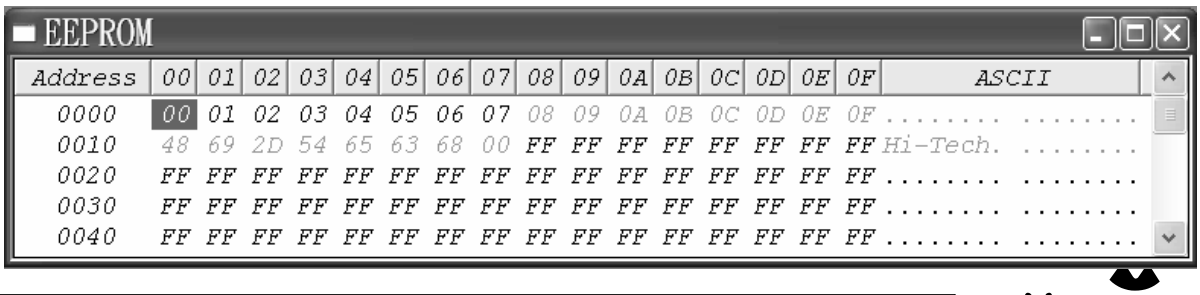
MICROCHIP

設定 EEPROM

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

PIC16F877A 範例：

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



MICROCHIP

課程結束!

感謝您對Microchip的支持
如有任何問題請電

0800-717-718

或

e-Mail: Taiwan.Techhelp@Microchip.com