

使用Microchip 組合語言編譯器

MPASM
&
MPLINK



認識 MPASM & MPLINK

www.microchip.com

自行下載



什麼是 MPASM

- ◆ MPASM 就是Microchip PICmicro Assembler
 - ◆ 單一原始檔案格式
 - 將組合語言直接翻譯成hex格式的機械碼
 - ◆ 多原始檔案格式
 - 將組合語言翻譯成object 格式的檔案，該檔案須經連結器(linker)再產生hex格式的機械碼



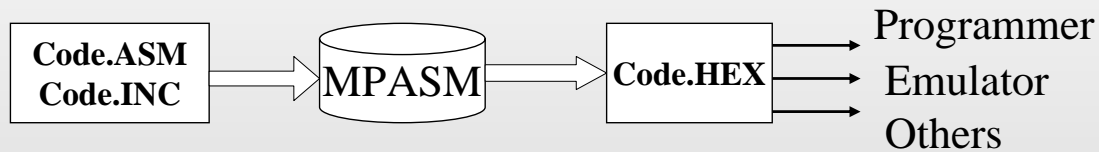
什麼是 MPLINK

- ◆ MPLINK 就是Microchip PICmicro Linker
- ◆ 依據連結描述檔的需求對所輸入的obj 檔案統一規劃處理
 - 聯結資料庫
 - 安排變數(RAM)位址
 - 安排程式(ROM)位址
 - 產生唯一的hex格式的機械碼

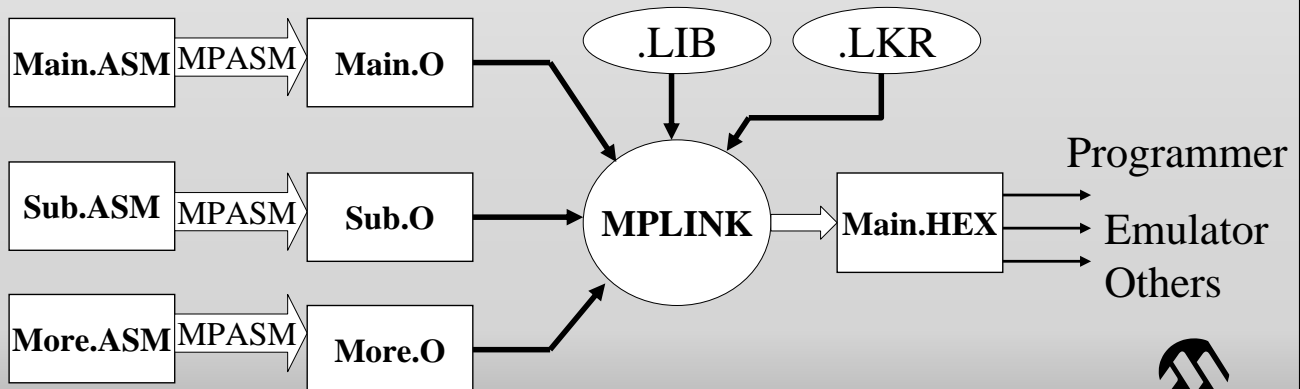


MPASM 組譯流程

● 單一原始組合語言檔 (.ASM → .HEX)



● 多原始組合語言檔 (.ASM → .OBJ → .HEX)



MPASM Input/Output Files

- ➔ .asm - 原始組合語言輸入檔
- ◆ .lst - 組譯完成後的列印輸出檔
- ◆ .err - 組譯錯誤提示輸出檔
- ◆ .hex - 16進制 ASCII 的輸出檔 (Intel Hex)
 - ✓ @ INHX8M - Intel 8-bit HEX Format (16C5x, 16C6x/7x)
 - ✓ @ INHX32 - Intel extended 32-bit address HEX format
- ◆ .cod - MPLAB IDE 除錯專用檔案
- ◆ .o - 可重新定址的目的檔

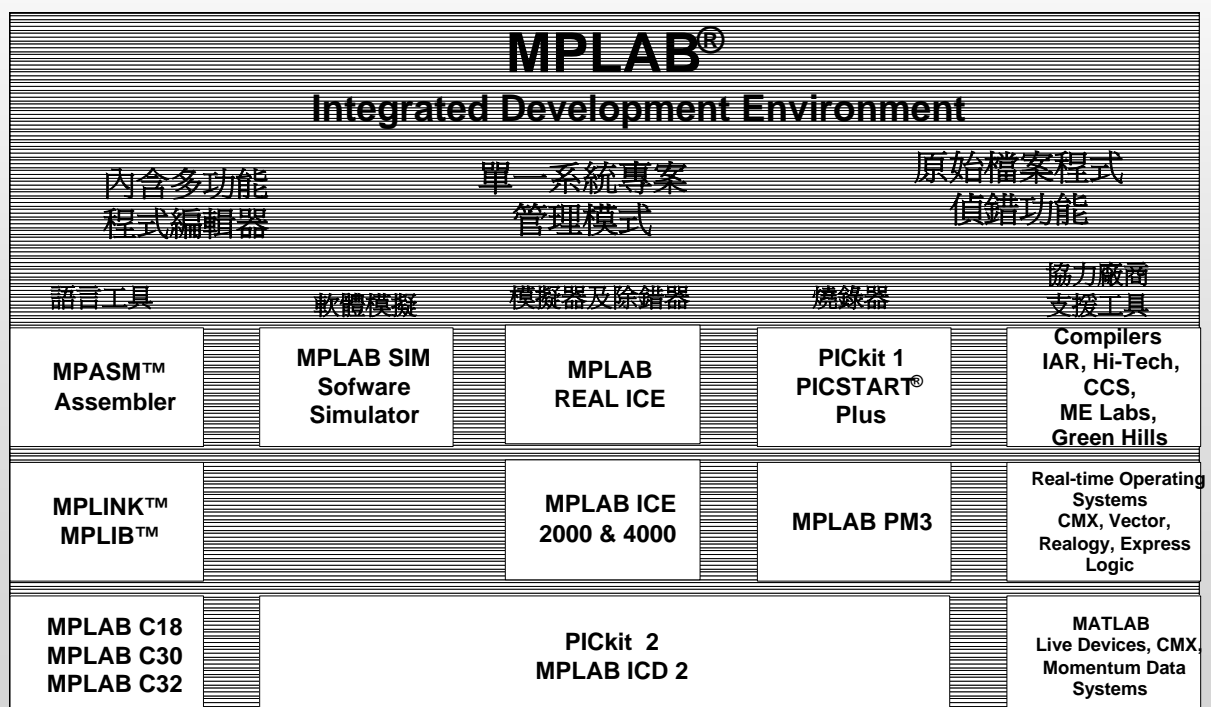
MPLINK Input/Output Files

- ➔ .o - 可重新定址的目的檔
- ➔ .lib - 資料庫檔，由 MPLIB 建立、MPLINK 來連結
- ➔ .lkr - 連結描述輸入檔
- ◆ .cof - 暫存檔，用來產生 .cod, .hex 及 .lst 等檔案
- ◆ .cod - MPLAB IDE 除錯專用檔案
- ◆ .map - 記憶體位址配置表
- ◆ .lst - 組譯完成後的列印輸出檔
- ◆ .hex - 16 進制 ASCII 的輸出檔 (Intel Hex)



MICROCHIP

MPLAB IDE - 整合式的發展環境



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



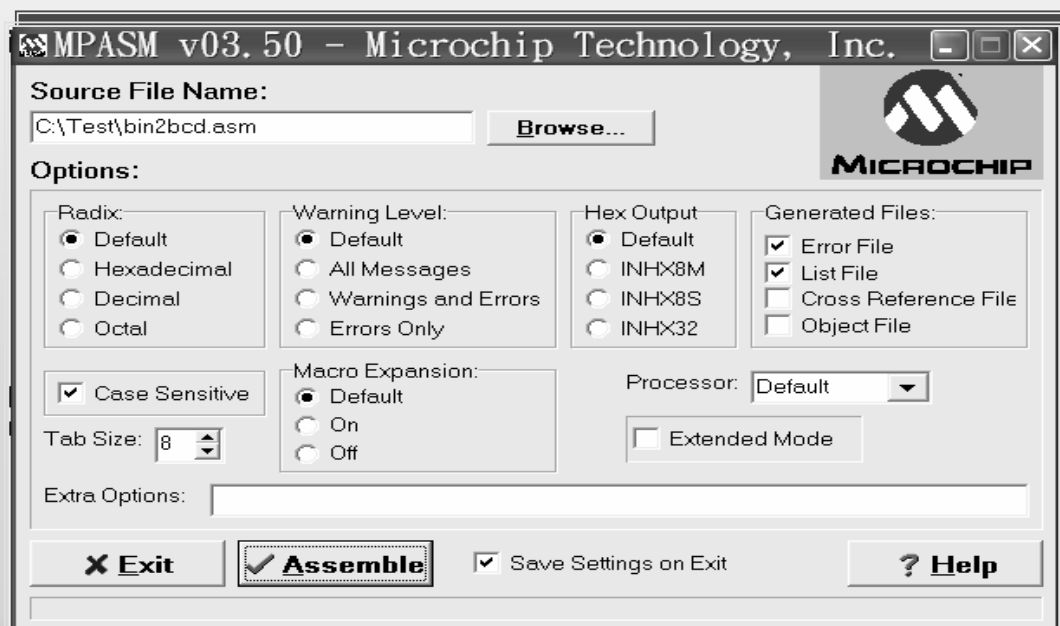
MICROCHIP

執行 MPASM 的方式

- ◆ 直接在 Windows 下執行 mpasmwin.exe
 - C:\Program Files\MPLAB IDE\MCHIP_Tools
- ◆ 在 MPLAB IDE 開發平台下叫用 MPASM
 - 建立 Project
 - 撰寫組合語言
 - 利用 Build 功能組譯組合語言

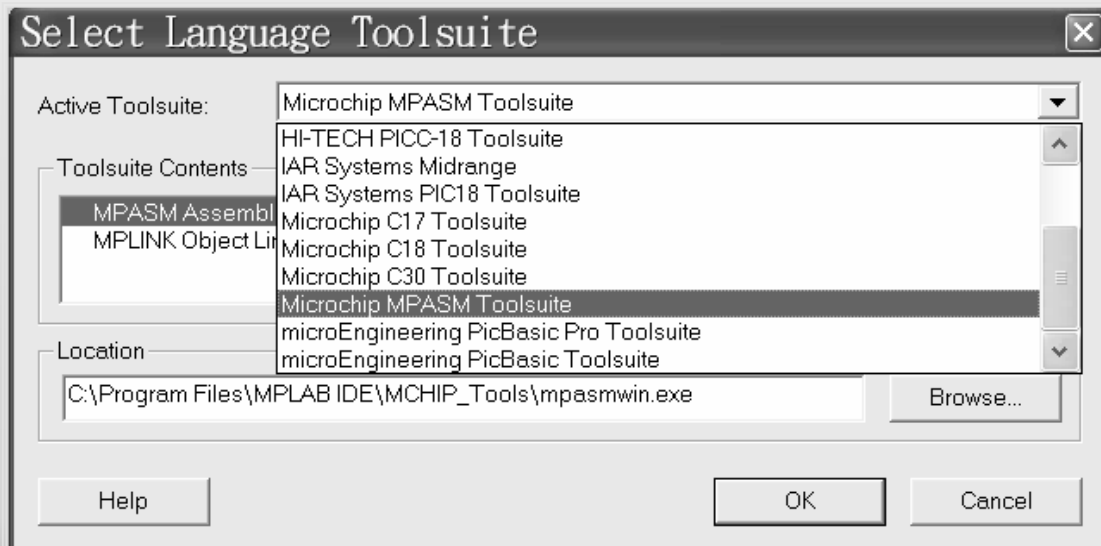
直接在Windows下執行MPASM

- ◆ C:\Program Files\MPLAB IDE\MCHIP_Tools\mpasmwin.exe



MPLAB IDE 選用 MPASM 組譯工具

- ◆ 在MPLAB IDE 主目錄下選擇
 - ◆ Project → Set Language Toolsuite...



組合語言的格式

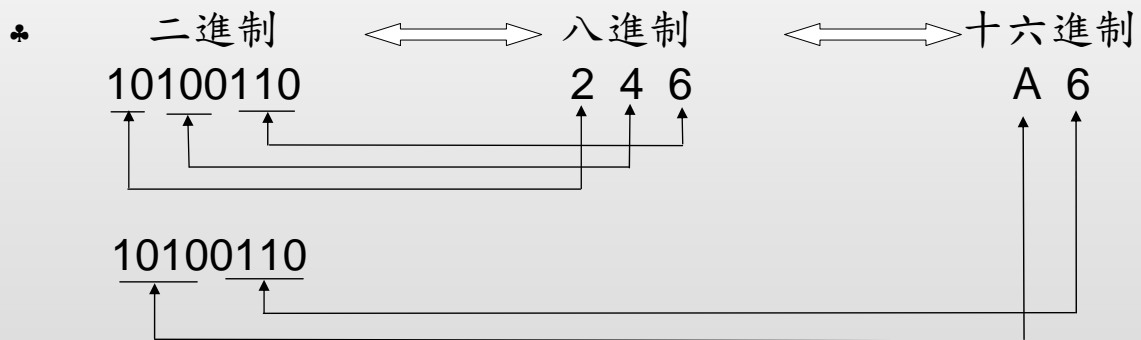
數值的表示

指令集

常數值的載入

數值、數字 及 字元 的表示

◆ 進制的互換



◆ 十六進制 ↔ 十進制

A 6 ($A \times 16 + 6 = 166$) 166

♥ 十六進制 ↔ 字元

41 A

61 a

(數值、數字) 表示法

◆ 十進制表示(Decimal): D'<十進制數目>' , .<十進制數目>

- MOVLW D'100' ; 載入常數 100₍₁₀₎ to W Reg.
- MOVLW .100 ; 載入常數 100₍₁₀₎ to W Reg.
- Const1 EQU D'200' ; Const1 = 200 (十進制)

◆ 十六進制表示(Hexadecimal): H'<十六進制數目>' , 0x<十六進制數目> , <十六進制數目>h

- MOVLW H'3F' ; 載入常數 3F₍₁₆₎ to W Reg.
- MOVLW 0x3F ; 載入常數 3F₍₁₆₎ to W Reg.
- MOVLW 0FEh ; 載入常數 FE₍₁₆₎ to W Reg.
- Const1 EQU H'5A' ; Const1 = 5A (十六進制)

◆ 八進制表示(Octal): O'<八進制數目>'

- MOVLW O'133' ; 載入常數 133₍₈₎ to W Reg.
- Const1 EQU O'300' ; Const1 = 300 (八進制)

(字元、字串) 表示法

◆ 二進制表示(Binary): B'<二進制數目>'

- MOVLW B'11110000' ; 載入常數 0xF0 to W Reg.
- Const1 EQU B'01010101' ; Const1 = 01010101 (二進制)

◆ 字元(ASCII): A'<字元>' , '<字元>'

- MOVLW A'R' ; 載入字元 "R" to W Reg.
- MOVLW 'c' ; 載入字元 "c" to W Reg.
- Const1 EQU 'a' ; Const1 = 小寫的 字元 A

◆ 字串(STRING): "字串"

- DT "Baud Rate = " ; 定訂查表字串為 "Baud Rate"



MICROCHIP

組合語言的語法格式

- ◆ 標記欄 (Labels)
- ◆ 指令助憶欄 (mnemonics)
- ◆ 運算欄 (operands)
- ◆ 註解欄 (comments)

```
;
;
; Sample MPASM Source Code. For illustration only
;
list      p=16f54
Dest      equ      h'0B'
           org      h'1FF'
           goto     Start
           org      h'000'
Start      movlw    h'0A'           ; Load 0x0a to WREG
           movwf    Dest           ; Store 0x0a to Dest
           bcf      Dest,3
           goto     Start
           end
```



MICROCHIP

標記欄 (Labels)

◆ 標記欄 (Label) - 程式位址的助憶標記

- 必須放在該行的第一個位置
- 標記開頭必須是英文字母或 “_”
- 標記文字不可超過 32 個字
- 字母大、小寫為不同的定義 (Default)

port_set	clrf	PORTC
	movlw	B'00001111'
_input:	movwf	PORTB
output		
	clrf	PORTA
	goto	port_set



指令與運算欄 (operands)

◆ 指令 (虛擬指令) 和 運算元

MOVF f, d ; Move RAM/Register to Destination

f 暫存器 或 RAM 的位址

d 選擇資料移送的目的暫存器

d= 0 時，放置在 w 暫存器，亦可寫成 “W”

d= 1 時，放回在 f 暫存器，亦可寫成 “F” 或忽略不寫

範例: var_count equ 0x3f
 ;

建議語法

 addwf var_count, 0
 addwf var_count, 1
 { addwf var_count, F
 addwf var_count, w



PICmicro MCU 指令集

(14-bit core 共35個指令)

位元組操作指令集		
NOP	-	No Operation
MOVWF	f	Move W to f
CLRW	-	Clear W
CLRF	f	Clear f
SUBWF	f,d	Subtract W from f
DECF	f,d	Decrement f
IORWF	f,d	Inclusive OR W and f
ANDWF	f,d	AND W and f
XORWF	f,d	Exclusive OR W and f
ADDWF	f,d	Add W and f
MOVF	f,d	Move f
COMF	f,d	Complement f
INCF	f,d	Increment f
DECFSZ	f,d	Decrement f, skip if zero
RRF	f,d	Rotate right f through carry
RLF	f,d	Rotate left f through carry
SWAPF	f,d	Swap nibbles of f
INCFSZ	f,d	Increment f, skip if zero

位元操作指令集		
BCF	f,b	Bit clear f
BSF	f,b	Bit set f
BTFSC	f,b	Bit test f, skip if clear
BTFSS	f,b	Bit test f, skip if set
立即數及控制指令集		
SLEEP	-	Go into standby mode
CLRWDI	-	Clear watchdog timer
RETLW	k	Return, place literal in W
RETFIE	-	Return from interrupt
RETURN	-	Return from subroutine
CALL	k	Call subroutine
GOTO	k	Go to address (k is 9-bit)
MOVLW	k	Move literal to W
IORLW	k	Inclusive OR literal with W
ADDLW	k	Add literal with W
SUBLW	k	Subtract W from literal
ANDLW	k	AND literal with W
XORLW	k	Exclusive OR literal with W

f = 暫存器, k = 8 位元的常數, b = 位元定址位置 <0,7>, d = 儲存的位置 (0 = W, 1 = F(預設值))

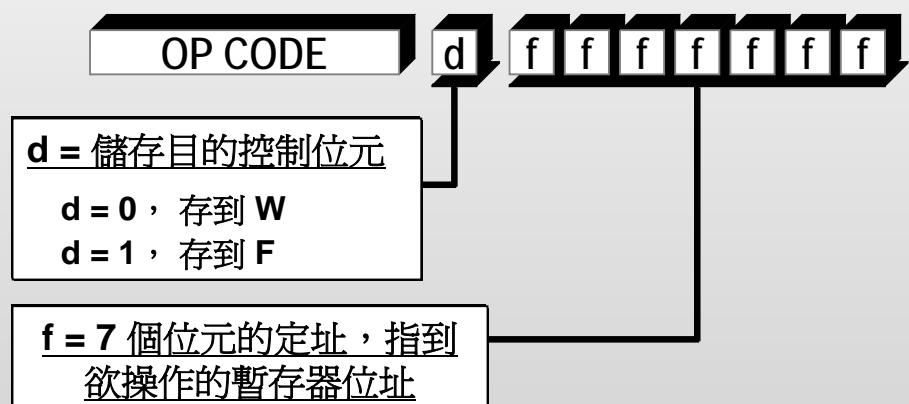


位元組操作指令集

位元組操作指令集

NOP	-
MOVWF	f
CLRW	-
CLRF	f
SUBWF	f,d
DECF	f,d
IORWF	f,d
ANDWF	f,d
XORWF	f,d
ADDWF	f,d
MOVF	f,d
COMF	f,d
INCF	f,d
DECFSZ	f,d
RRF	f,d
RRL	f,d
SWAPF	f,d
INCFSZ	f,d

14 位元指令 – 位元組操作指令說明



Example:

ADDWF REG, W
ADDWF REG



位元操作指令集

位元操作指令集

BCF f,b
BSF f,b
BTFSC f,b
BTFSS f,b

14 位元指令 – 位元操作指令說明



b = 3 個位元的定址
， 指到欲運算的位元

f = 7 個位元的定址，指到欲
操作的暫存器位址

Example:

BTFSC STATUS, Z
BTFSC PORTC, 3

立即數運算指令集

立即數運算指令集

MOVLW k
IORLW k
ADDLW k
SUBLW k
ANDLW k
XORLW k

14 位元指令 – 位元操作指令說明



k = 8 個位元，立即常數

Example:

MOVLW 0x2F
MOVLW k

控制指令集

控制指令集

SLEEP	-
CLRWDT	-
RETLW	k
RETFIE	-
RETURN	-
CALL	k
GOTO	k

14 位元指令 – 控制指令說明 RETLW



k = 8 個位元，立即常數

14 位元指令 – 控制指令說明 CALL K、GOTO K



k = 11 個位元，立即位址

CALL K，GOTO K 只能指到 11 個有效位址，故直接定址能力只有 2K 的範圍
故需配合 PCLATH 的 bit [3:4] 以擴展程式位址到 8K 的範圍

應注意的指令 減法指令

◆ 減法指令 -- SUBLW k，SUBWF f, d

- 減法是採 2 的補數運算
- SUBLW k 的運算方式為 (K - W --> W)
- SUBWF f, d 的運算方式為 (f - W --> W)
- 減法運算後，若 C=1 表示沒有借位 (即 f >= w, C 為 1)
- 減法運算後，若 C=0 表示有借位 (即 f < w, C 為 0)
- 減法運算後，若 Z=1 表示兩數相等 (即 f = w, Z 為 1)

C 旗號	Z 旗號	比較結果
1	0	F >= W
0	0	F < W
1	1	F = W

- 若需判斷 F > W，測試 C = 1, Z=0 兩條件同時成立
- 若要判斷 F <= W，測試 C = 0 或 Z=1 只要有一條件成立

應注意的指令

CALL , GOTO 指令

◆ CALL k , GOTO k

- k 為 11-bit 的直接位址值，它會直接以指令的方式直接載入到 PC 的 bit<10:0> 的位置 (範圍為 2K 內)
- 每個連續 2K 位址範圍的程式記憶區塊稱為一個 Page
- 以 PIC16F877 為例，最高定址能力為 8K，須切分為 4 個 Page
- PC 的最高位元位址 bit<11:12> 直接從暫存器 PCLATH<4:3> 載入以擴展程式位址至 8K 範圍 (0000h:1FFFh)。
- K --> PC<10:0> , PCLATH<4:3> --> PC<12:11>
.....
這種方式稱為跳頁選擇 (Page Select)

◆ 建議使用需指令 PAGESEL 做跳頁的切換



MICROCHIP

常數值的計算(一)

◆ 在 MPASM 的語法中，常數值是可以利用一些計算符號來計算其值

- ❖ + : 加法運算
- ❖ - : 減法運算
- ❖ * : 乘法運算
- ❖ / : 除法運算後，取商數
 - movlw (a * b - c) / d - a
- ❖ % : 除法運算後，取餘數
 - movlw .65536 % .1134
- ❖ - : 將數值取 2 的補數 (負數)
 - addlw -2 ; 將 W 減 2 運算
- ❖ ~ : 將數值取 1 的補數 (位元反向)



MICROCHIP

常數值的計算(二)

- ❖ << : 向左旋轉n個位元
- ❖ >> : 向右旋轉n個位元
 - `movlw h'2A' >> 3`
- ❖ && : 邏輯 AND 判斷
 - `if (a == 255) && (b == c)`
- ❖ || : 邏輯 OR 判斷
- ❖ ! : 邏輯反向判斷
 - `if !(a == b)`
- ❖ & : 邏輯 AND 運算
 - `movlw (b'10101010'>>3) & 0x0F`
- ❖ ^ : 邏輯 XOR 運算
- ❖ | : 邏輯 OR 運算

常數值的計算(三)

- ❖ \$: 目前的位址 (適用 PIC16Fxxx 的組合語言, 如果使用此語法在 PIC18Fxxxx 的話後面所帶的位址必須為偶數位址)
 - `goto $; 永久迴圈`
 - `goto $-3 ; 往回跳三個指令`
 - `goto $+5 ; 往下跳五個指令`
- ❖ LOW : 取位址的低位元組
- ❖ HIGH : 取位址的高位元組
- ❖ UPPER : 取位址的最高位元組
 - `movlw (low) CTR_Table`
 - `movlw (high) CTR_Table`

常數值載入 – 範例

假設 Timer1 接一個外部 32768Hz 的石英振盪器，欲設定的中斷時間為一秒，其所需的常數值可以用下列方式來設定給 Timer1 的計數器

```
#define TMR1_VAL .32768 ; Define Timer1 time period are 1 SEC
;
movlw (.65536 -TMR1_VAL) /.256 ; Calculate the MSB value for the Timer1
movwf TMR1H
movlw (.65536 -TMR1_VAL)%.256 ; Calculate the LSB value for the Timer1
movwf TMR1L
```

認識最基本的虛指令

LIST
#INCLUDE
EQU
ORG
END

必需要使用的五個虛擬指令

- ◆ LIST - 目錄控制 (Listing Control)
 - list p=16f877a, f=INHX8M
- ◆ #INCLUDE - 加入一原始檔、定義檔或敘述檔
 - #include <C:\MPLAB\16f877a.INC>
- ◆ EQU - 宣告常數、變數
 - memory equ 0x3f
 - count equ .100
 - io_set equ B'11000011'
- ◆ ORG - 設定程式組譯的起始位址
 - org 0x00 ; 組譯位址從“00h”開始
 - org 0x30 ; 組譯位址從“30h”開始
- ◆ END - 程式結束



MICROCHIP

虛指令 - list

- ◆ list – 顯示項目的選擇
- ◆ 主要是用來設定使用何種 PIC 元件
- ◆ 次要用來選擇 hex 檔案的輸出格式

使用範例:

```
list p=16f877a, f=INH8S
```

設定使用 PIC16F877A，Hex 的輸出格式為標準的 8-bit 格式



MICROCHIP

虛指令 - #include

- ◆ #include – 加入額外的原始程式或定義檔案
 - “18f452.inc” 與 <18f452.inc> 磁碟路徑不同如下範例之說明

使用範例:

```
#include "c:\workshop\wap002\18f452.inc"
```

(使用 指定路徑 將 18f452.inc 的定義檔加入)

```
#include <18f452.inc>
```

(使用 MPLAB IDE 內定路徑 將 18f452.inc 的定義檔加入)

虛指令 - equ

- ◆ equ – 定義一個常數值或位址給一個代表文字
- ◆ 該代表文字只能唯一不可重複再定義

使用範例:

```
Count_us    equ    h'20'
Count_ms    equ    Count_us + 1
DLY_VAL     equ    h'20'
;
movlw       DLY_VAL
movwf       Count_us
movlw       DLY_VAL + .100
movwf       Count_ms
; ***** 你看的出兩個 h'20' 有何差異? *****
```

虛指令 - org

- ◆ org – 設定底下程式執行的起始位址
- ◆ org 虛指令只能用在獨立的程式設計，它無須藉由 Linker 來安排位址，就可以產生特定執行位址的 hex
- ◆ 不指定 org 時，程式自動從位址 0x00 開始

使用範例:

```
ORG    0x000           ; 設定程式執行位址從“0”開始
                        ; (Reset Vector)

clrf   PCLATH
goto   main
ORG    0x004           ; 中斷向量進入位址
;***** 中斷處理副程式區 *****
retfie           ; 中斷返回
```

虛指令 - end

- ◆ end – 結束組合語言的組譯動作
- ◆ 在 end 以後的組合語言將不會被組譯

虛擬指令的使用 - 範例一

```
list      p=16f877a      ; 定義使用的 MCU 為 PIC16F877A
#include <p16f877a.inc> ; 使用 16F877A 的標準定義檔
;***** 定義變數、常數、參數區 *****
t_delay   EQU      D'100'
dly_count EQU      H'7'
;*****
;
ORG       0x000          ; 設定程式執行位址從"0"開始
                        ; (Reset Vector)

clrf      PCLATH
goto     main
ORG       0x004          ; 中斷向量進入位址
;***** 中斷處理副程式區 *****
retfie    ; 中斷返回
;
main      movlw      t_delay      ; 主程式開始
          movwf      dly_count
          :
; remaining code goes here
END       ; 程式結束
```

練習一

- ◆ 建立一個 MPLAB IDE 的 Project
 - 目錄在 ..\Exercise\Exer1
 - 原始程式為 Exer1.asm
 - 建立的Project名稱為 Exer1.mcp
- ◆ 組譯成功後，觀察 Exer1.lst 的記憶分配
- ◆ 這是組合語言最基本的寫法

MPLINK



MICROCHIP

介紹 MPLINK

◆ 什麼是MPLINK?

- ◆ MPLINK 是將組合語言或 C 編譯器所產生的obj 檔加以連結並指定變數及程式執行的位址後，輸出一可執行的 hex 檔。

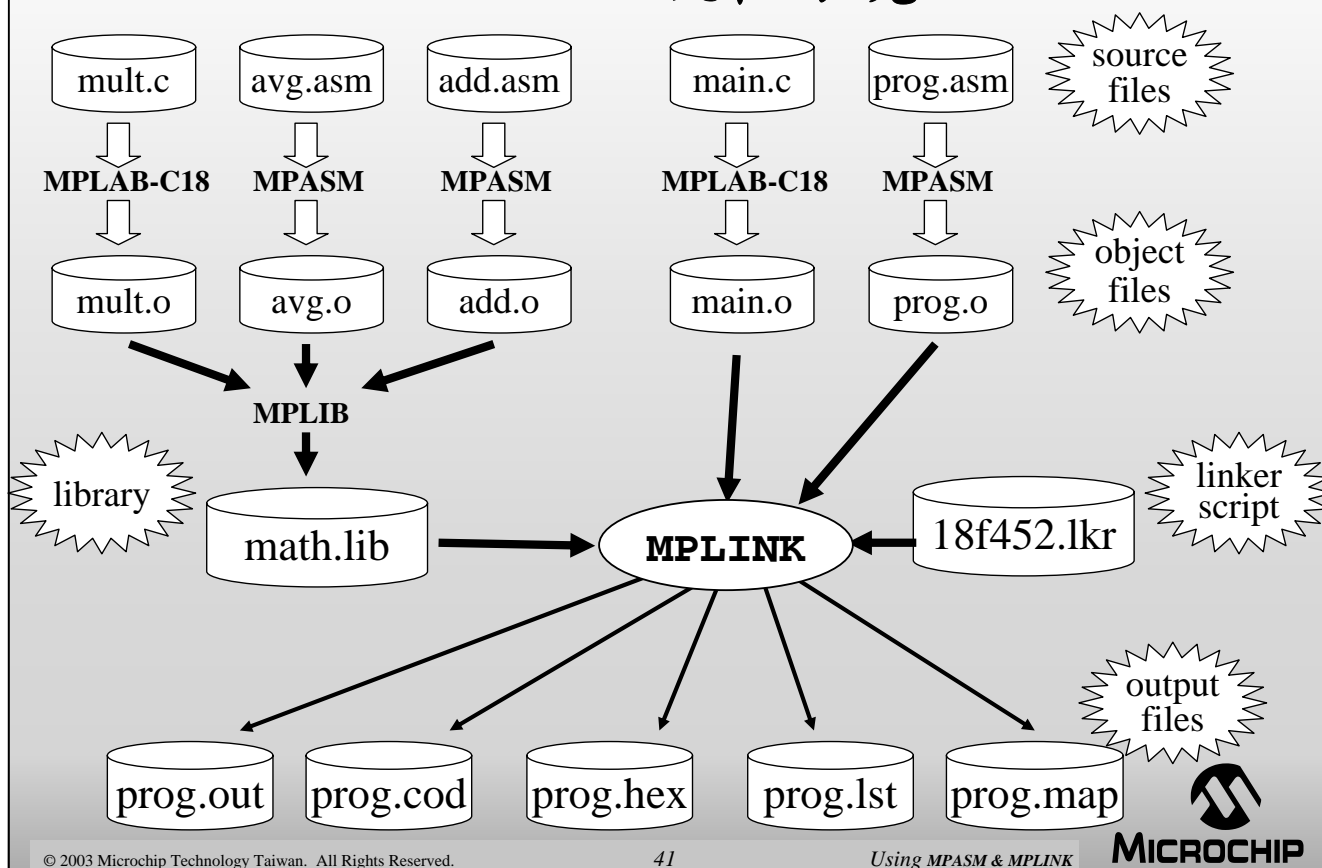
◆ MPLINK 能作做什麼？

- ◆ 安排位址給程式 (CODE) 及資料暫存器 (RAM)
- ◆ 產生執行檔 (HEX)
- ◆ 安排堆疊位址及深度給 MPLAB C-17 / C18
- ◆ 提供 COD 檔以利程式偵錯
- ◆ 讓程式更容易模組化
- ◆ 連結資料庫 (Library)



MICROCHIP

MPLINK 流程方塊



© 2003 Microchip Technology Taiwan. All Rights Reserved.

41

Using MPASM & MPLINK

MICROCHIP

可重新定位程式

◆ 固定位址組合語言程式

- 程式執行位址及變數位址在程式中已強制指定不需 MPLINK 來重新排定位址
- 此種方式適用於較小的程式及獨立個人程式的研發

◆ 可重新定位程式

- 採用高階語言時程式編譯後，程式及變數需連接資料庫以決定最後的執行位址
- 多個組合語言程式間相互位址的安排
- 適用開發大程式及多人合作撰寫的程式

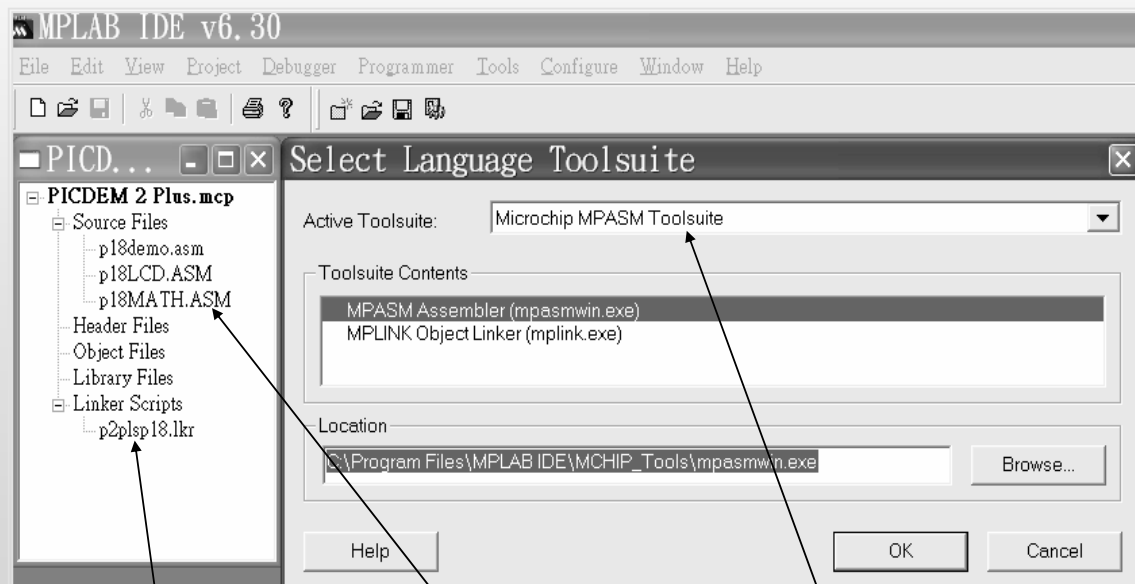
© 2003 Microchip Technology Taiwan. All Rights Reserved.

42

Using MPASM & MPLINK

MICROCHIP

在 MPLAB 下使用 MPLINK



LINK 描述檔

組合語言
原始程式檔

語言工具
列的設定

// Sample linker command file for 16F877
LIBPATH .

```
CODEPAGE NAME=vectors START=0x0 END=0x4 PROTECTED
CODEPAGE NAME=page0 START=0x5 END=0x7FF
CODEPAGE NAME=page1 START=0x800 END=0xFFF
CODEPAGE NAME=page2 START=0x1000 END=0x17FF
CODEPAGE NAME=page3 START=0x1800 END=0x1FFF
CODEPAGE NAME=.idlocs START=0x2000 END=0x2003 PROTECTED
CODEPAGE NAME=.config START=0x2007 END=0x2007 PROTECTED
CODEPAGE NAME=eedata START=0x2100 END=0x21FF PROTECTED
```

```
DATABANK NAME=sfr0 START=0x0 END=0x1F PROTECTED
DATABANK NAME=sfr1 START=0x80 END=0x9F PROTECTED
DATABANK NAME=sfr2 START=0x100 END=0x10F PROTECTED
DATABANK NAME=sfr3 START=0x180 END=0x18F PROTECTED
```

```
DATABANK NAME=gpr0 START=0x20 END=0x6F
DATABANK NAME=gpr1 START=0xA0 END=0xEF
DATABANK NAME=gpr2 START=0x110 END=0x16F
DATABANK NAME=gpr3 START=0x190 END=0x1EF
```

```
SHAREBANK NAME=gprnobnk START=0x70 END=0x7F
SHAREBANK NAME=gprnobnk START=0xF0 END=0xFF
SHAREBANK NAME=gprnobnk START=0x170 END=0x17F
SHAREBANK NAME=gprnobnk START=0x1F0 END=0x1FF
```

```
SECTION NAME=STARTUP ROM=vectors // Reset and interrupt vectors
SECTION NAME=PROG1 ROM=page0 // ROM code space - page0
SECTION NAME=PROG2 ROM=page1 // ROM code space - page1
SECTION NAME=PROG3 ROM=page2 // ROM code space - page2
SECTION NAME=PROG4 ROM=page3 // ROM code space - page3
SECTION NAME=IDLOCS ROM=.idlocs // ID locations
SECTION NAME=CONFIG ROM=.config // Configuration bits location
SECTION NAME=DEEPROM ROM=eedata // Data EEPROM
```

MPLINK 描述檔 (16F877.1kr)

程式位址宣告

特殊暫存器位址宣告

資料RAM位址宣告

共用RAM位址宣告

內定的程式節區宣告

連結描述檔 – 命令行

- LIBPATH – 指定資料庫(Library)的路徑
- LKRPATH – 指定連結描述檔路徑
- FILES – 將所指定的特定資料庫(.lib)或.obj 檔案加入到目前編譯的程式中

連結描述檔 – 記憶體的命令

- CODEPAGE – 程式記憶區域
- DATABANK – 切換 Bank 的資料記憶區域
- SHAREBANK – 共用的資料記憶區域 (16F87x)
- ACCESSBANK – 共用的資料記憶區域 (18Fxxxx)

PROTECTED 關鍵字可以被用來設定某些特定位址的保護，除了使用者強行設定外，一般MPLINK是不會將變數及程式安排此位址

第一階段的虛指令



常用的虛擬指令 -- #DEFINE

#DEFINE <定義名稱> <取代的字串>

- ◆ 通常用來定義文字、標記 & 常數。
- ◆ 在此定義中，<取代的字串>所描述的文字是以 <定義名稱> 內的文字為助憶文字，在組合語言中可以用 <定義名稱> 來代替 <取代的字串>以 增加程式的閱讀性。

例：

```
#DEFINE S_CLK PORTC,3
#DEFINE S_DATA PORTC,2
#DEFINE LENGTH .20
```

```
list p=16f877
#include <p16f877.inc>
;
val_ff equ h'F8'
;
#define m_flag Count,6
#define clk PORTC,0
#define sda PORTC,1
:
:
bsf clk
btfss sda
goto sda_low
call sda_hi
```



常用的虛擬指令 -- SET

Label

SET

H'3F'

- ◆ 通常用來宣告變數 & 常數。
- ◆ 在使用 SET 所宣告之變數或常數均可重覆使用。SET 再對同一 Label 重新宣告時，組譯器會使用最近的宣告為主。
- ◆ SET 和 EQU 之功能極為類似，但 EQU 的宣告變數或常數不可重複。
- ◆ 例：

```
area      set      .20
width     set      h'3F'
area      set      width * .5
```

```
list      p=16f877
#include   <p16f877.inc>
;
val_hi    set      h'ff' - .101
val_low   set      h'00'
mask      set      b'10110001'
;
:
movlw     val_hi
movwf     TMR1H
movlw     val_low
movwf     TMR1L
:
val_hi    set      h'ff' - .201
movlw     val_hi
movwf     TMR1H
```

常用的虛擬指令 -- TITLE & PAGE

TITLE “*Interrupt Service Routine*”
PAGE

- ◆ <TITLE> 是用標示主題程式以增加程式的閱讀性。

例如：

```
title     “中斷程式處理”
:
title     “數學運算副程式”
```

- ◆ 若有<TITLE>的設定，在列印 .LST 檔案時，會自動在每頁報表的第一行加入此顯示設定。

- ◆ <PAGE> 是用來設定檔案列印 (.LST) 的換頁控制

- ◆ 在列印 .LST 檔時，若使用了<PAGE>指令則在列印時印表機會自動換頁列印。

常用的虛擬指令 -- DT

Lable *DT* *<expr>, <string>*

- ◆ 用來設定查表的資料
- ◆ 每一資料的內容是以8-bit為主，其內容將會與 RETLW 指令相互結合以 "RETLW K" 的形式置放於程式中。

說明：

```
out_tab    addwf    PCL,F
           retlw    'a'
           retlw    'b'
           retlw    'c'
```

可以用以下的指令取代之

```
out_tab    addwf    PCL,F
           dt        "abc"
```

```
list p=16f877
#include <p16f877.inc>
index_vol    equ    h'00'
;
           movlw    HIGH br_tab
           movwf    PCLATH
           movlw    index_vol
           call    br_tab
           :
           org      0x300
br_tab       addwf    PCL,F
           dt        "Baud Rate!"
ev_tab       addwf    PCL,F
           dt        "even parity!"
od_tab       addwf    PCL,F
           dt        "odd parity!"
```

練習二

- ◆ 試著修改 16F877A.lkr 檔案中的位址設定
 - 目錄在 ..\Exercise\Exer2
 - 原始程式為 Exer2.asm , LCD.asm
 - 建立的Project名稱為 Exer2.mcp
- ◆ 變更節區名稱：STARTUP , PROG1
 - 檢查 map 檔的程式位址是否有變
 - 沒有 map 檔案，在 "Build Options → Project → MPLINK" 勾選 Generate map file

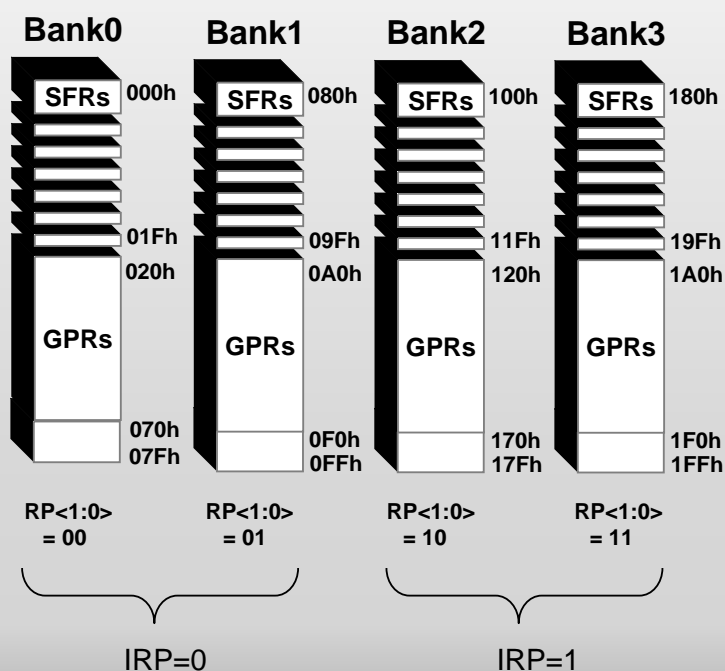
安排的 RAM 位址

RAM 的配置
BANK 切換問題
不可重新定位語法
可重新定位語法



MICROCHIP

PIC16F87x RAM 的架構



- ◆ 共有四個 Banks，每個 bank 的大小為 128 bytes
- ◆ 特殊暫存器(SFRs)的位址分佔每個 Bank 最前面的 32 byte 位址
- ◆ Banks 的切換動作需由 STATUS 暫存器中的 RP0，RP1及 IRP 位元來完成
- ◆ Share Bank 的位址在每個 Bank 的最後 16 byte



MICROCHIP

換Bank的好幫手 -- BANKSEL

BANKSEL TRISA

- ◆ 依據 BANKSEL 指令所指到的暫存器，自動加入該暫存器頁 (BANK) 的設定指令。
- ◆ 透過 BANKSEL 的自動設定與 UDATA (Uninitialized data) 宣告，可輕鬆的進行多原始檔的組譯並加以連結成一 HEX 檔。

以PIC16C77為例：共有四個暫存器頁(BANK) 就會使用到兩個換頁指令

000B	<u>1683 1303</u>	00049	<u>banksel</u> ADCON1
Message[302]: Register in operand not in bank 0. Ensure that bank			
000D	019F	00050	clrf ADCON1
Message[302]: Register in operand not in bank 0. Ensure that bank			
000E	170C	00051	bsf PIE1, ADIE
000F	<u>1283 1303</u>	00052	<u>banksel</u> ADCON0
0011	30C1	00053	movlw h'c1'
0012	009F	00054	movwf ADCON0

定義變數(RAM)的位址

- ◆ 簡單常用方式
 - EQU, SET
- ◆ 不可重新定位語法
 - CBLOCK, ENDC
- ◆ 可重新定位語法
 - UDATA
 - UDATA_ACS
 - UDATA_SHR
 - RES (需配合 UDATA 使用)

建議的寫法

- ◆ 有關常數的寫法
 - 最好都用大寫以利於區別於變數
 - 常數的定義建議使用 `#define`，避免使用 `EQU` 和 `SET` 來指定
- ◆ 有關變數的寫法
 - 第一個字母最好用大寫，以“_”區隔
 - `Volt_Range`，`Rec_Buffer`，`Counter`
- ◆ 有關 label 的寫法
 - 第一個字母最好用“_”來識別
 - 大、小寫不限制
 - `_LCD_Table`，`_Sine_Table`



MICROCHIP

一、使用 EQU & SET 定義變數位址

- ◆ 定義 RAM 位址最簡單的方式
- ◆ 可用在兩種定位址的模式中

```
***** VARIABLE DEFINITIONS
;
Count_us      EQU    0x20      ; Delay function variable for uS
W_temp        EQU    0x71      ; variable used for context saving
Status_Temp   EQU    0x72      ; variable used for context saving
Pclath_Temp    EQU    0x73      ; variable used for context saving
;
#define        Carry    STATUS,C
#define        SW2      PORTA,4    ; Define KEY SW2 = PA4 pin
#define        Motor    PORTC,2    ; Define Motor drive pin
;
M_KEY0        EQU    0x01      ; Define Manufacture Code (LSB)
M_KEY1        EQU    0x23
M_KEY2        EQU    0x45
```



MICROCHIP

EQU 定義變數於不同 Bank

- ◆ EQU 對定義不同 Bank 的變數位址需採全域位址的方式，如此方可正確的切換正確的 Bank

```
;***** VARIABLE DEFINITIONS
B0_ABC      EQU      0x20          ; Variable on Bank 0 at 0x20 Location
B1_ABC      EQU      0xA0          ; Variable on Bank 1 at 0x20 Location
B2_ABC      EQU      0x120        ; Variable on Bank 2 at 0x20 Location
B3_ABC      EQU      0x1A0        ; Variable on Bank 3 at 0x20 Location
;
                banksel    B0_ABC      ; Select Bank0
                movlw      0x00
                movwf      B0_ABC      ; Save WREG to 0x20
;
                banksel    B1_ABC      ; Select Bank1
                movlw      0x01
                movwf      B1_ABC      ; Save WREG to 0xA0
;
                banksel    B3_ABC      ; Select Bank3
                movlw      0x03
                movwf      B3_ABC      ; Save WREG to 0x1A0
```

二、 不可重新定位的變數定義

- ◆ CBLOCK & ENDC 的語法(一)

CBLOCK <強制定義位址>

<變數名稱1> , <變數名稱2> , <變數名稱n>

ENDC

- ◆ 語法(二)

CBLOCK <強制定義位址>

<變數名稱1> [:位址的增量值] ,

<變數名稱2> [:位址的增量值] ,

ENDC



CBLOCK 的強制定位方式

- ◆ CBLOCK 無須經 MPLINK 排定變數位址，故使用時需在程式裡強制設定變數群的起始位址
- ◆ 程式裡可以使用多個 CBLOCK 來定位址，但一定要用 ENDC 指令來結束該變數區塊的設定
- ◆ CBLOCK 所指定的位址必須是一個有效位址，使用時應注意
- ◆ 如果需定義不同 Bank 的變數位址，需採用全域位址的方式來定義位址



MICROCHIP

CBLOCK & ENDC 的語法(一)

```
00010      list  p = 16f877
00011      include <p16f877.inc>
00001      LIST
00002 ; P16F877.INC Standard Header File, Version 1.1 Microchip Technology, Inc.
00013 ;
00014      CBLOCK 0x00 ← 強制定址在 0x00
00000000 00015      C_Hold_Delay      ; variable used for the sample hold charge time
00000001 00016      Flag_Reg          ; Flag Register
00000002 00017      Hex_Temp         ; Hex to ASICC convert buffer
00018      ENDC
00019
00020      CBLOCK 0x70 ← 強制定址在 0x70
00000070 00021      WREG_TEMP        ;variable used for context saving
00000071 00022      STATUS_TEMP     ;variable used for context saving
00000072 00023      BSR_TEMP        ;variable used for context saving
00024      ENDC
00025 ;
00026 #define TMR1_VAL .32768      ; Define Timer1 time period are 1 SEC
00027 #define TxD_Flag  Flag_Reg,0 ; Define the TX data flag
```



MICROCHIP

CBLOCK & ENDC 的語法(二)

```

00010          list  p = 16f877
00011          include <p16f877.inc>
00001          LIST
00002 ; P16F877.INC Standard Header File, Version 1.1 Microchip Technology, Inc.
00013 ;
00014          CBLOCK 0x00
00000000 00015          Int_Var : 2          ; 16-bit integer variable
00000002 00016          Long_Var : 4         ; 32-bit variable reserved
00000006 00017          Hex_Temp : 2         ; 2-bytes convert buffer
00018          ENDC
00019
00020          CBLOCK 0x70
00000070 00021          WREG_TEMP      ;variable used for context saving
00000071 00022          STATUS_TEMP    ;variable used for context saving
00000072 00023          BSR_TEMP      ;variable used for context saving
00024          ENDC
00025 ;
00026 #define TMR1_VAL .32768      ; Define Timer1 time period are 1 SEC
00027 #define TxD_Flag Flag_Reg,0 ; Define the TX data flag

```

該變數佔有4個 Bytes

CBLOCK & ENDC 的 Bank 切換

- ◆ 如需使用各個 Bank 的 RAM，CBLOCK 的強制定位需擴及到全域位址

```

;***** VARIABLE DEFINITIONS
          CBLOCK 0x20          ; Variable on Bank 0 at 0x20 Location
          B0_ABC , B0_DEF
          ENDC
;
          CBLOCK 0xA0          ; Variable on Bank 1 at 0x20 Location
          B1_ABC , B1_DEF
          ENDC
;
          banksel B0_ABC      ; Select Bank0
          movlw 0x00
          movwf B0_ABC        ; Save WREG to 0x20
;
          banksel B1_ABC      ; Select Bank1
          movlw 0x01
          movwf B1_ABC        ; Save WREG to 0xA0
;

```

變數宣告在 Bank1

三、可重新定位的變數定義

◆ UDATA

- 宣告未設定初始值變數的資料節區(Data Section)
- 適用於所有 PIC

◆ UDATA_SHR

- 宣告變數的資料節區在 Share Bank 區域
- 適用於 PIC16F87x

◆ UDATA_ACS

- 宣告變數的資料節區在 Access Bank 區域
- 適用於 PIC18Fxxxx

UDATA 的宣告

- ◆ UDATA 資料節區的宣告經 MPASM 組譯後，會先產生 obj 檔案再由 MPLINK 安排變數位址

- ◆ UDATA 語法

<Section Name> UDATA <RAM Address>

- <Section Name> RAM節區名稱：可省略
 - ✓ 如有指定節區名稱，會依 16F877A.lkr 檔案中對 RAM 該節區位址的設定進行RAM實際位址的安排
 - ✓ 如不指定節區名稱，會自動安排到 unprotected 的 GPR(n) RAM 的位址
 - ✓ 節區名稱不可重複
- <RAM Address> 自定 RAM 位址：可省略
 - ✓ 自定變數在 RAM 的起始位址
 - ✓ 該起始位址可以為 protected 或 unprotected 的區域

UDATA & RES 合併使用

- ◆ UDATA 是對RAM節區的宣告，必須使用 RES 來對RAM 位址的保留

範例一：最簡單的使用方式 (由 MPLINK 自動安排到 GPRn 的位址)

```
UDATA
Var1    RES    1
Double  RES    2
```

範例二：自定位址方式 (由 MPLINK 安排到 0x20 的起始位址)

```
UDATA    0x20
Var1     RES    1
Double   RES    2
```

範例三：節區位址方式 (由 MPLINK 安排到節區宣告的起始位址)

```
My_RAM  UDATA
Var1     RES    1
Double   RES    2
```

UDATA_SHR 的宣告 (16F87x)

- ◆ UDATA_SHR 宣告本節區位指定安排在 Share Bank
- ◆ Share Bank 位址在 0x70 ~ 0x7F，無須切換 Bank
- ◆ UDATA_SHR 語法

<Section Name> UDATA_SHR <RAM Address>

- <Section Name> RAM節區名稱：可省略
 - ✓ 有指定或不指定節區名稱都沒關係，自動安排到 Share bank 的位址
 - ✓ 節區名稱不可重複
- <RAM Address>：必須在 Share Bank 的位址
 - ✓ 使用 Share Bank 時，一般無須設定位址

在 MPLINK 的描述檔 (RAM部份)

// Sample linker command file for 16F877

// \$Id: 16f877.lkr,v 1.4 2002/01/29 22:10:01 sealep Exp \$

DATABANK	NAME=sfr0	START=0x0	END=0x1F	PROTECTED
DATABANK	NAME=sfr1	START=0x80	END=0x9F	PROTECTED
DATABANK	NAME=sfr2	START=0x100	END=0x10F	PROTECTED
DATABANK	NAME=sfr3	START=0x180	END=0x18F	PROTECTED

DATABANK	NAME=gpr0	START=0x20	END=0x6F
DATABANK	NAME=gpr1	START=0xA0	END=0xEF
DATABANK	NAME=gpr2	START=0x110	END=0x16F
DATABANK	NAME=gpr3	START=0x190	END=0x1EF

SHAREBANK	NAME=gprnobnk	START=0x70	END=0x7F
SHAREBANK	NAME=gprnobnk	START=0xF0	END=0xFF
SHAREBANK	NAME=gprnobnk	START=0x170	END=0x17F
SHAREBANK	NAME=gprnobnk	START=0x1F0	END=0x1FF

SECTION	NAME=Share_RAM	RAM=gprnobnk	// Shared Bank
SECTION	NAME=My_RAM0	RAM=gpr0	// RAM variable space - Bank0
SECTION	NAME=My_RAM1	RAM=gpr1	// ROM variable space - Bank1

可重新定位範例 – RAM 的宣告

;***** VARIABLE DEFINITIONS (examples)

; example of using Shared Uninitialized Data Section

Share_RAM	UDATA_SHR		
w_temp	RES	1	; variable used for context saving at 0x70
status_temp	RES	1	; variable used for context saving at 0x71
Rec_Data	RES	1	; USART received data buffer

; example of using Uninitialized Data Section

My_RAM0	UDATA		; variable on GPR0 at 0x20
temp_count	RES	1	; temporary variable
Count_ms	RES	1	
Count_us	RES	1	

My_RAM1	UDATA		; variable on GPR1 at 0xA0
Tx_Ring	RES	16	; reserved 16 bytes for Tx ring buffer
TX_Flag	RES	1	; 8-bit Tx status Flag

UDATA_ACS 的宣告 (18Fxxx)

- ◆ UDATA_ASC 宣告本節區位指定安排在 Access Bank
- ◆ Access Bank 位址在 0x00 ~ 0x7F，無須切換 Bank
- ◆ UDATA_ACS 語法

<Section Name> UDATA_ACS <RAM Address>

- <Section Name> RAM節區名稱：可省略
 - ✓ 指定或不指定節區名稱都沒關係，自動安排到 Access bank 的位址
 - ✓ 節區名稱不可重複
- <RAM Address>：必須在 Access Bank 的位址
 - ✓ 使用 Access Bank 時，一般無須設定位址



練習三

- ◆ 試著修改 Exer3.mcp 檔案中的變數設定
 - UDATA 與 p16f877a.lkr 關係
 - UDATA_SHR 的位址
- ◆ 執行程式後，觀察變數在 RAM 的位址
- ◆ 虛指令 BANKSEL 是否很重要？



安排的程式執行位址

程式記憶體位址配置

Page 切換問題

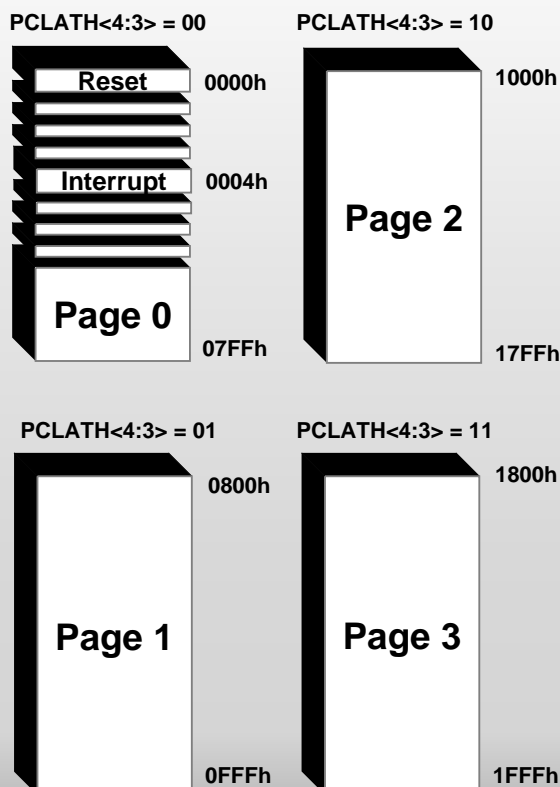
不可重新定位語法

可重新定位語法



MICROCHIP

PIC16F87x 程式記憶體架構



- ◆ 最大的程式定址範圍為8K words (13 bits)
- ◆ 8K的程式範圍再劃分四個2K大小的程式頁(Page)
- ◆ 程式頁(Page)的切換許使用PCLATH<4:3>的兩個位元
- ◆ 程式超過2K時，需要考慮到換頁的問題
- ◆ Reset 向量位址在 0000h
- ◆ 中斷向量位址在 0004h



MICROCHIP

程式頁 (Page) 的切換 -- PAGESEL

PAGESEL Subroutine

- ◆ 依據 PAGESEL 指令所指到的程式位址，自動加入程式頁 (PAGE) 的設定指令。
- ◆ PAGESEL 通常與 CALL 及 GOTO 指令一起使用，並可擴展使用多個原始檔之間的互相呼叫。

以PIC16f877為例:共有四個程式頁(BANK)，就會使用到兩個換頁指令

```
0000          00033      org      0x0
0000  158A 120A      00034      pagesel initial
Message[306]: Crossing page boundary -- ensure page bits are set.
0002  2230          00035      call      initial
0003  118A 160A      00036      pagesel start
Message[306]: Crossing page boundary -- ensure page bits are set.
0005  2FFF          00037      goto      start
☛ initial 起始位址設定在 h'0A30'
☛ start  起始位址設定在 h'17FF'
```



程式記憶體跳頁問題

- PAGESEL 可以幫助您完成因程式超過 2K 範圍所需對 PCLATH 的設定工作
- 對可重新定位的程式因MPLINK會重新安排執行位址也許這時候程式就會有換頁的問題產生，故適時的使用 PAGESEL 是有必要的

例如：要呼叫的位址為 TARGET_ADDR

使用下列的敘述

```
PAGESEL      TARGET_ADDR
CALL         TARGET_ADDR
```

相當於

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



定義程式(ROM)的位址

- ◆ 不可重新定位語法
 - ORG - 設定程式起始位址
- ◆ 可重新定位語法
 - CODE - 程式節區位址的宣告

CODE 的宣告

- ◆ 宣告 obj 檔案的程式節區起始位址，需經由 MPLINK 安排最終的程式執行位址
- ◆ 語法：

<Section Name>	CODE	<ROM Address>
-----------------------------	-------------	----------------------------

 - **<Section Name>** 程式節區名稱：可省略
 - ✓ 如有指定節區名稱，會依 16F877.lkr 檔案中對 ROM 節區位址的設定進程式位址的安排
 - ✓ 如不指定節區名稱，會自動安排到 unprotected 的 ROM 區
 - ✓ 程式節區名稱不可重複
 - **<ROM Address>** 自定 ROM 的位址：可省略
 - ✓ 自定程式執行的起始位址
 - ✓ 該起始位址可以為 protected 或 unprotected 的區域

CODE 使用範例

◆ PROG1 CODE

- 節區名稱為 PROG1，Linker 會排定在Page0 的區域 (詳細請參閱 p16f877.lkr 的檔案)

◆ ABC CODE

- 宣告程式節區名稱為ABC
- 如在 Link 描述檔中有指定名稱則按指定位址排定
- 如無指定名稱，則由Link自行安排到unprotected區域

◆ CODE 0x0004

- 無指定節區名稱，但強制定址到0x04的位址
- 常用於特定位址的設定，例:中斷向量位址

◆ CDE CODE 0x2000

- 節區名稱僅供參考，強制指定位址到0x2000

在 MPLINK 的描述檔 (程式部份)

```
// File: c:\Program files\mplab\16c77.lkr
// Sample linker command file for 16c77
```

CODEPAGE	NAME	START	END	PROTECTED
CODEPAGE	NAME=vectors	START=0x0	END=0x4	PROTECTED
CODEPAGE	NAME=page0	START=0x10	END=0x7FF	
CODEPAGE	NAME=page1	START=0x800	END=0xFFFF	
CODEPAGE	NAME=page2	START=0x1000	END=0x17FF	
CODEPAGE	NAME=page3	START=0x1800	END=0x1FFF	
CODEPAGE	NAME=.idlocs	START=0x2000	END=0x2003	PROTECTED
CODEPAGE	NAME=.config	START=0x2007	END=0x2007	PROTECTED

SECTION	NAME	ROM	Comment
SECTION	NAME=STARTUP	ROM=vectors	// Reset and interrupt vectors
SECTION	NAME=SUB_PROG0	ROM=page0	// ROM code space - page0
SECTION	NAME=SUB_PROG1	ROM=page0	// ROM code space - page1
SECTION	NAME=MAIN	ROM=page1	// ROM code space - page2
SECTION	NAME=PROG4	ROM=page3	// ROM code space - page3
SECTION	NAME=IDLOCS	ROM=.idlocs	// ID locations
SECTION	NAME=CONFIG	ROM=.config	// Configuration bits location

STARTUP	CODE	SUB_PROG0	CODE
	goto Initial		clrwdt
	nop		clrw
	nop		movlw b'00001111'
	nop		banksel TRISA
	goto Int_service		movwf TRISA

練習四

- ◆ 利用 MPLAB IDE 開啟 \Exercise\Exer4 的 Exer4.mcp
- ◆ 檢查一下 Ist 檔案裡的程式分配位置與 PAGESEL 的跳頁切換



MICROCHIP

特殊位址的安排

Configuration Word

ID Location

Internal EEPROM



MICROCHIP

常用的虛擬指令 -- `__CONFIG`

`__CONFIG _CP_ALL & _XT_OSC & _LVP_ON`

- ◆ 設定 IC 硬體工作方式，此項設定是非常重要的，倘若設定錯誤將使程式發生不可預期的誤動作。
- ◆ 每一種設定選項均須用一符號 & (AND) 相連。
- ◆ 需注意使用右表的助憶文字時必需使用該 IC 預設的定義檔

例如：

```
#include <p16f877.inc>
```

- ◆ Configuration Word 的位址是 0x2007 (14-bit core only)

Configuration Bits of PIC16F877

```
;  
_CP_ALL           _CP_HALF  
_CP_UPPER_256     _CP_OFF  
;  
_DEBUG_ON         _DEBUG_OFF  
_WRT_ENABLE_ON    _WRT_ENABLE_OFF  
_CPD_ON           _CPD_OFF  
_LVP_ON           _LVP_OFF  
_BODEN_ON         _BODEN_OFF  
_PWRTE_OFF        _PWRTE_ON  
_WDT_ON           _WDT_OFF  
;  
_LP_OSC           _XT_OSC  
_HS_OSC           _RC_OSC
```



常用的虛擬指令 -- `__IDLOCS`

`__IDLOCS H'1234'`

- ◆ 設定此 IC 共四位數的辨別碼。
- ◆ 識別碼位元組的位址是 <2000h:2003h>，每個位址有 14 個位元但只有最低的 4 位元有效
- ◆ 此識別碼不論設定與否，均不影響程式之執行，但有些 IC 設定程式保護模式後會受該識別碼影響而改變 Check-Sum。
- ◆ 在使用 `__IDLOCS` 指令之前必須事先設定微處理器的型號。



設定範例

程式經過組譯過後將會在0x2000~0x2003產生ID辨識碼，在0x2007產生Configuration Word 的設定值；兩者所產生的機械碼會以Hex格式存在於程式碼裡。

```
MPASM 02.30.10 Intermediate    TEST.ASM    3-21-2000    15:13:47    PAGE 1
;
LOC      OBJECT CODE      LINE      SOURCE TEXT
          00001           title  "Test  program"
          00002           list  p=16c74b
          00003 #include   <p16c74b.inc>
2000      0001 0002 0003 00004      _ _idlocs      H'1234'
          0004
2007      3FF1           00005      _ _config      _XT_OSC & _PWRTE_ON
                                     & _WDT_OFF & _CP_OFF
```

設定 EEPROM Data

- ◆ 利用 虛指令 DE 來設定EEPROM的內容
- ◆ 此方式設定的資料是8-bit
- ◆ 必須透過燒錄後才會將資料寫入EEPROM
- ◆ 由於EEPROM位址在0x2100，需用ORG或CODE訂定位址

- ◆ 語法:

<Label> DE <expr>,<expr>, ...

- <Label> 標記助憶符號，可省略
- <expr> 8-bit 型態的設定資料，也可為字串型態

DE 使用範例

◆ 用 ORG 定址

```
ORG      0x2100
DE       "Firmware Version v2.30" ,0
DE       'A' , 'B' , 0x07 , 0x0A , 0x0D
```

◆ 用 CODE 定址

DEEPROM CODE

```
DE       "Firmware Version v2.30" ,0
DE       'A' , 'B' , 0x07 , 0x0A , 0x0D
```

由Linker指定

練習五

◆ 利用 MPLAB IDE 開啟 \Exercise\Exer5 的 Exer5.mcp

- 設定 PIC16F877A 的 Configuration Word
- 設定 PIC16F877A 的 ID
- 設定 PIC16F877A 的 EEPROM Data

程式間相互的連結

變數傳遞
副程式的呼叫
不同程式的跳躍



不同程式間的呼叫

- ◆ 在同一個Project中，由三個不同的程式A，B，C 組成
 - A所宣告的變數，B與C都需用到
 - B需呼叫A與C的副程式
 - C會直接跳到A，A也會跳到C

- ◆ 如此錯綜複雜的關係應如何解決???



進階的虛擬指令 -- GLOBAL

GLOBAL 標記 1, 標記 2, 變數....

- ◆ **Global** 是用來宣告本指令後所銜接的標記或變數是屬於共用標記、變數。
- ◆ **Global** 通常是使用在多個原始檔的程式組譯，透過 **Global** 和 **Extern** 指令的宣告，其它原始程式均可直接使用。

例如：

```

udata
global var1, var2
var1    res    1
var2    res    1
Startup code
global delay
delay   movlw  h'3f'
```

```

global mulplr, h_byte, l_byte,
global mpy_sub
mulcnt equ h'60'
mulplr equ mulcnt + 1
h_byte equ mulcnt + 2
l_byte equ mulcnt + 3
count  equ mulcnt + 4
;
code
mpy_sub clrf h_byte
clrf    l_byte
movlw   h'8'
movwf   count
movf    mulcnt, W
bcf     STATUS, C
loop    rrf    mplplr, F
btfsc   STATUS, C
addwf   h_byte, F
```



進階的虛擬指令 -- EXTERN

EXTERN 標記 1, 標記 2, 變數....

- ◆ **Extern** 是用來告知組譯器，其後所連接的標記或變數已經在其它原始檔中宣告過。
- ◆ **Extern** 通常是使用在多個原始檔的程式組譯，透過 **Extern** 指令的宣告，目前的原始程式可直接使用 **Global** 所宣告的標記或變數。
- ◆ 使用 **Global** 和 **Extern** 指令的時機通常是撰寫多個原始檔時所需使用的指令，**MPLINK** 會自動安排相關的位址。

```

extern mulplr, h_byte, l_byte
extern mpy_sub
;
PROG1 code
;
movf    PORTA, W
movwf   mulplr
movf    PORTB, W
movwf   mulcnt
;
call    mpy_sub
movf    h_byte, W
movwf   PORTC
movf    l_byte, W
movwf   PORTD
:
```



MPLINK 的範例 – (MAIN.ASM)

本程式在 PAGE0

宣告外部的標記

EXTERN DELAY_1MS, HEX_BCD, DECODE
EXTERN RS_232, EEPROM, KEYBOARD

本程式在 PAGE1

```

000000 3008 MOVLW 0x8
000001 008a MOVWF 0xa
000002 2000 CALL 0x0
000004 0009 RETFIE
000800 0064 CLRWDT
000801 0103 CLRW
000802 300f MOVLW 0xf
000803 1683 BSF 0x3,0x5
000804 1303 BCF 0x3,0x6
000805 0085 MOVWF 0x5
000806 30f0 MOVLW 0xf0
000807 0086 MOVWF 0x6
000808 1283 BCF 0x3,0x5
000809 1303 BCF 0x3,0x6
00080a 3000 MOVLW 0x0
00080b 008a MOVWF 0xa
00080c 201a CALL 0x1a
00080d 2017 CALL 0x17
00080e 201c CALL 0x1c
00080f 2010 CALL 0x10
000810 201e CALL 0x1e
000811 3008 MOVLW 0x8
000812 008a MOVWF 0xa
000813 280a GOTO 0xa
    
```

STARTUP
pagesel

call

Int_Service
MAIN
initial

CODE
initial

在 LKR 檔中宣告位址

直接宣告位址= 0x4

在 LKR 檔中宣告位址

```

initial
CODE
retfie
CODE
clrwdt
clrwdt
movlw b'00001111'
banksel TRISA ; Select RAM bank1
movwf TRISA
movlw b'11110000'
movwf TRISB
banksel count ; Select RAM bank0
loop
pagesel RS_232 ; Select PAGE 0
call RS_232
call DECODE
call EEPROM
call DELAY_1MS
call KEYBOARD
movlw HIGH loop
movwf PCLATH ; Select PAGE 1
goto loop
end
    
```



MPLINK 的範例 – (SUB1.ASM)

本副程式在 PAGE0

; P16C77.INC Standard Header File, Version 1.01 Microchip Technology, Inc.

```

val_ff
count
count_ms
;
;
SUB_PROG1
;
000010 30f8 MOVLW 0xf8 DELAY_1MS
000011 00a0 MOVWF 0x20
000012 03a0 DECF 0x20,0x1 dec_loop
000013 1d03 BTFSS 0x3,0x2
000014 2812 GOTO 0x12
000015 0008 RETURN
;
000016 0008 RETURN HEX_BCD
000017 0008 RETURN DECODE
    
```

title "SUB1.ASM"

list p=16c77

#include <p16c77.inc>

#include <main.equ>

#include <p16c77.inc>

equ 0xf8

equ 0x20

equ 0x21

GLOBAL DELAY_1MS, HEX_BCD, DECODE

CODE

```

movlw val_ff
movwf count
decf count,f
btfss STATUS,Z
goto dec_loop
return
    
```

return
return
end



MPLINK 的範例 – (SUB2.ASM)

```
title      "SUB2.ASM"
list       p=16c77
;
#include <p16c77.inc>
; P16C77.INC Standard Header File, Version 1.01 Microchip Technology, Inc.
#include <main.equ>
val_ff     equ      0xf8
count      equ      0x20
count_ms   equ      0x21
;
GLOBAL RS_232, EEPROM, KEYBOARD
SUB_PROG2  CODE
;
00001a 0000 NOP      RS_232  nop
00001b 0008 RETURN   return ;
00001c 0000 NOP      EEPROM   nop
00001d 0008 RETURN   return ;
00001e 0000 NOP      KEYBOARD nop
00001f 0008 RETURN   return ;
end
```

本副程式在 PAGE0

© 2003 Microchip Technology Taiwan. All Rights Reserved.

95

Using MPASM & MPLINK



有關定位虛指令整理

◆ 不可重新定位指令

- 程式：ORG
- 變數：CBLOCK, ENDC

◆ 可重新定位指令

- 程式：CODE
- 變數：UDATA, UDATA_SHR, UDATA_ACS

© 2003 Microchip Technology Taiwan. All Rights Reserved.

96

Using MPASM & MPLINK



練習六

- ◆ 利用 MPLAB IDE 開啟 \Exercise\Exer6 的 Exer6.mcp
 - 練習不同程式間的呼叫與變數傳遞
 - 熟悉 GLOBAL 與 EXTERN 的用法
 - 注意不同 Page 的切換



第二階段的虛指令

BANKSEL
DA , DATA , DB , DE , DT, DW
MACRO , LOCAL , ENDM
FILL , TITLE , SUBTITLE



BANKISEL

- ◆ 產生索引定址模式中的第九位元
- ◆ PIC16F87x RAM的位址從0x00~0x1FF，索引定址暫存器(FSR)只能定址到八個位元，第九個位元就可由此虛指令設定
- ◆ 語法：

BANKISEL <RAM 的助憶符號>

範例：

```
movlw      Var1    ; Var1的bit 9 是個未知值
movwf      FSR
bankisel    Var1
...
movwf      INDF
```

DA

- ◆ 產生一組以14-bit 為型態的兩個 7-bit ASCII 字元
- ◆ DA 所產生的字元或字串存放在 ROM
- ◆ 語法：

<Label> DA <expr> , <expr1> , ...

範例：

```
01B9  30E2 31E4 32E6    00482  Table_1  DA    "abcdef"
01BC  18B2 19B4 1AB6    00483                DA    "123456",0
                                0000
01C0  3FFF                00484                DA    0x3FFF
                                00485 ;
```

DATA

- ◆ 在程式中建立數字或字串資料(word)
- ◆ 建議使用在PIC18Fxxxx元件中
- ◆ 語法：

<Label> DATA <expr> , <expr1> , ...

範例:

```
01C1  3132 3334 2162      00487          DATA      "1234abcd"
      2364
```

Message[303]: Program word too large. Truncated to core size. (6162)

Message[303]: Program word too large. Truncated to core size. (6364)

(14-bit 架構無法置放 16-bit 資料所產生的訊息)

```
01C5  004E              00488          DATA      'N'
                        00489
```



MICROCHIP

DB

- ◆ 在程式中宣告 Byte 資料
- ◆ 建議使用在PIC18Fxxxx元件中
- ◆ 語法：

<Label> DB <expr> , <expr1> , ...

範例:

```
000298  2020      db      " Temperature "
00029a  6554
00029c  706d
00029e  7265
0002a0  7461
0002a2  7275
0002a4  2065
0002a6  2020
```



MICROCHIP

DE & DT

◆ DE

- 定義 Internal EEPROM 資料

◆ DT

- 定義 Mid-Range (14-bit) 程式查表資料

DW

- ◆ 在程式中宣告 Word 資料，類似 DB 虛指令
- ◆ 建議使用在 PIC18Fxxxx 元件中
- ◆ 語法：

<Label> DW <expr> , <expr1> , ...

範例:

```
000298 2020        dw     " Temperature "
00029a 6554
00029c 706d
00029e 7265
0002a0 7461
0002a2 7275
0002a4 2065
0002a6 2020
```

MACRO & LOCAL & ENDM

- ◆ MACRO 一般稱之為巨集函數，是用來制定使用者自己專用的指令，並可傳遞參數給巨集函數內的指令。

User_instru	MACRO	var_in1,var_in2,var_out
使用者定義的指令名稱		參數及變數傳遞

- ◆ LOCAL是用來定義在巨集函數內所使用到的變數(Variable)給本巨集函數使用，不同的巨集函數可使用相同的變數名稱。
- ◆ 在巨集函數內，如需執行 GOTO 指令可用 "\$" 符號做為跳躍的基準位址做加(往下跳)或減(往上跳)的短程跳躍。

swap_led	movlw	h'8'
	movwf	led_count
	rrf	led_right,F
	rlf	led_lift,F
	decfsz	led_count
	goto	\$.3

- ◆ ENDM是用來結束本巨集函數。
- ◆ 有 MACRO 的宣告就必須有 ENDM 的存在。



MICROCHIP

MACRO & LOCAL & ENDM – 範例

```
portc_bfr equ 0x70
w_buffer equ 0x71
s_buffer equ 0x72
;
PUSH MACRO
    movwf w_buffer
    swapf w_buffer,F
    swapf STATUS,W
    movwf s_buffer
ENDM
;
POP MACRO
    swapf s_buffer,W
    movwf STATUS
    swapf w_buffer,W
ENDM
```

```
MOV MACRO regd,regs
    movf regs,w
    movwf regd
ENDM
*****
;
; Program start
*****
;
;
; org 0x0
; goto start
;
;
; org 0x04 ; Interrupt
; PUSH
; MOV portc_bfr,PORTC
;
; POP
; retfie
```



MICROCHIP

FILL

- ◆ 填入立即值或程式碼
- ◆ “連續填入的個數”可以用 \$ 為參考位址來計算所剩餘的記憶容量。
- ◆ 在完成的程式中，所剩餘的記憶容量最好填入 GOTO H'0'以增加系統的穩定性。
- ◆ 語法：

FILL (填入值)，(連續填入的個數)

以使用 2K 的 PIC16C72A 為例：

```
          :  
          FILL (goto 0x00),(h'7ff' - $+1)  
          END
```

TITLE & SUBTITLE

- ◆ TITLE
 - 程式顯示列印名稱
 - TITLE “CAN Host Monitoring, ver 1.10”
- ◆ SUNTITLE
 - 副程式顯示列印名稱
 - SUBTITLE “Interrupt Service Routine”

條件組譯控制指令

虛擬指令	說 明	使用範例
IF	條件式組譯區塊的起始指令	IF Rate < 50 retlw Slow_Speed
ELSE	當 IF 條件不成立時用以執行相對組譯區塊的指令	ELSE retlw Fast_Speed
ENDIF	結束條件式組譯區塊的判斷	
IFDEF	判斷標記是否已被定義的指令	IFDEF __16F877
IFNDEF	判斷標記是否未被定義的指令	IFNDEF AD_10BIT

IF , ELSE , ENDIF

當欲使用某些條件的成立與否來控制 MPASM 的組譯流程時，可利用這些條件判斷指令來達成

- ◆ 語法：
IF <條件>
 “條件成立的組譯程式”
ELSE
 “條件不成立的組譯程式”
ENDIF
- ◆ 當條件成立(TRUE) 時則執行 IF 之下區塊之組譯。
- ◆ ELSE 用來處理當 IF 的條件不成立時(FALSE)的狀況。
- ◆ ENDIF 用於結束該條件式組譯

使用範例

```
#define           Motor_Type    50
:
:
call            Table_Read
:
:
Table_Read:
IF Motor_Type == 50
retlw   0xC0
ELSE
retlw   0x40
ENDIF
```

IFDEF, IFNDEF

當欲使用標記的宣告與否來控制 MPASM 的組譯流程時, 可利用這些條件判斷指令來達成

- ◆ IFDEF 的語法與 IFNDEF 相同, 皆為 IFDEF (or IFNDEF) <標記>。
- ◆ 標記可使用 #define 的虛指令來加以定義。
- ◆ 適當的標記宣告與使用可增加程式開發時的彈性與減少修改的複雜度。例如欲使用 16F87X 為 16C6X/7X 的發展 IC 時, 便可使用如右的方法來調適其間的差異

IFDEF 的使用範例

EX1:

```
list    p=16F877
:
IFDEF  __16F877
MESSG "Check Processor Type"
ENDIF
```

EX2:

```
#define  USE_ 16F877
:

IFDEF USE_16F877
; Device used 16F877
    movf ADRESH,W
ELSE
; Device used 16C74B
    movf  ADRES,W
ENDIF
```



Intel Hex 檔案補充說明



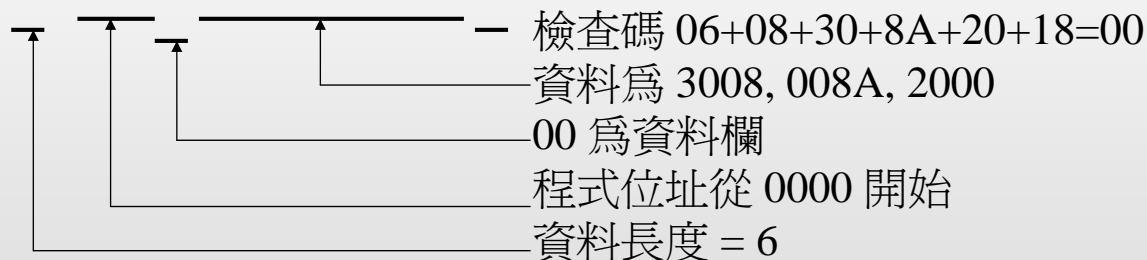
HEX檔格式說明

- ◆ MPASM & MPLINK 的 HEX 輸出格式共有三種：
 - Intel HEX Format (INHX8M) : 給標準 8-bit 格式燒錄器
 - Intel Split Hex Format (INHX8S) : 需要燒錄 奇、偶 分離的 ROM
 - Intel Hex 32 Format (INHX32) : 給需要 16-bit 格式燒錄器
- ◆ INHX8M 格式說明 -- : EE AABB TT LLHH LLHH CC
 - EE : 為二位 16 進制值，用以指示該行的資料長度
 - AABB : 為四位 16 進制值，用以指示資料的起始位址
 - TT : 為二位 16 進制值，為該行欄位功能指示
 - TT = "00" 表是該行是一般性資料
 - TT = "01" 表是資料結束
 - LLHH : 16 進制值 ROM 的資料 (即程式碼)，PIC 的程式碼是以字元(word)為單位，其低位的 8-bit (lsb)是放在 LL，較高的 8-bit (msb)是放在 HH 的欄位
 - CC : 為二位 16 進制的檢查碼，該行全部以 16 進制加總後，其和為零 (EE+AA+BB+TT+LL+HH+LL1+HH1+ .. +CC = Zero)



HEX檔格式範例

:0600000008308A00002018



:020008000900ED

:10002000F830A000A003031D1228080008000800F3

:040030000034003464

:0C003400000008000000080000000800A8

:10100000640003010F30831603138500F03086005F

:101010008312031300308A001A2017201C2010208E

:081020001E2008308A000A2896

:00000001FF

01 表示資料結束

