PIC101

PIC16入門

A Leading Provider of Smart, Connected and Secure Embedded Control Solutions

❑使用軟體工具

- MPLAB X IDE V5.50(或更新版本)
- pic-as Assember & hlink Linker(內建於XC8)

❑使用硬體工具

- MPLAB SNAP
- Microchip Taiwan office APP_All_MCU_2023實驗板

❑參考文件

- Microchip PIC16F18446 Data Sheet(DS40001985)
- MPASM to MPLAB XC8 PIC Assembler Migration Guide(DS-50002973)
- MPLAB XC8 PIC Assembler User's Guide(DS-50002974)
- MPLAB XC8 PIC Assembler User's Guide for Embedded Engineers(DS50002994)
- MPLAB X IDE User's Guide(DS-50002027)
- APP_All_MCU_2023線路圖

課程目標

- 使用 Microchip 開發工具
- 認識 Microchip 的 PICmicro
- 組合語言的正確撰寫格式
- 基礎的程式寫作
  - I/O 的使用
  - 延時副程式 (Delay Subroutine)
  - 計時器(Timer)
  - PWM控制 & 輸出
  - 類比 / 數位 轉換器 (ADC) 的使用
- MPLAB-SNAP 及多功能實驗板的操作

Microchip 開發工具&實驗板

MICROCHIP

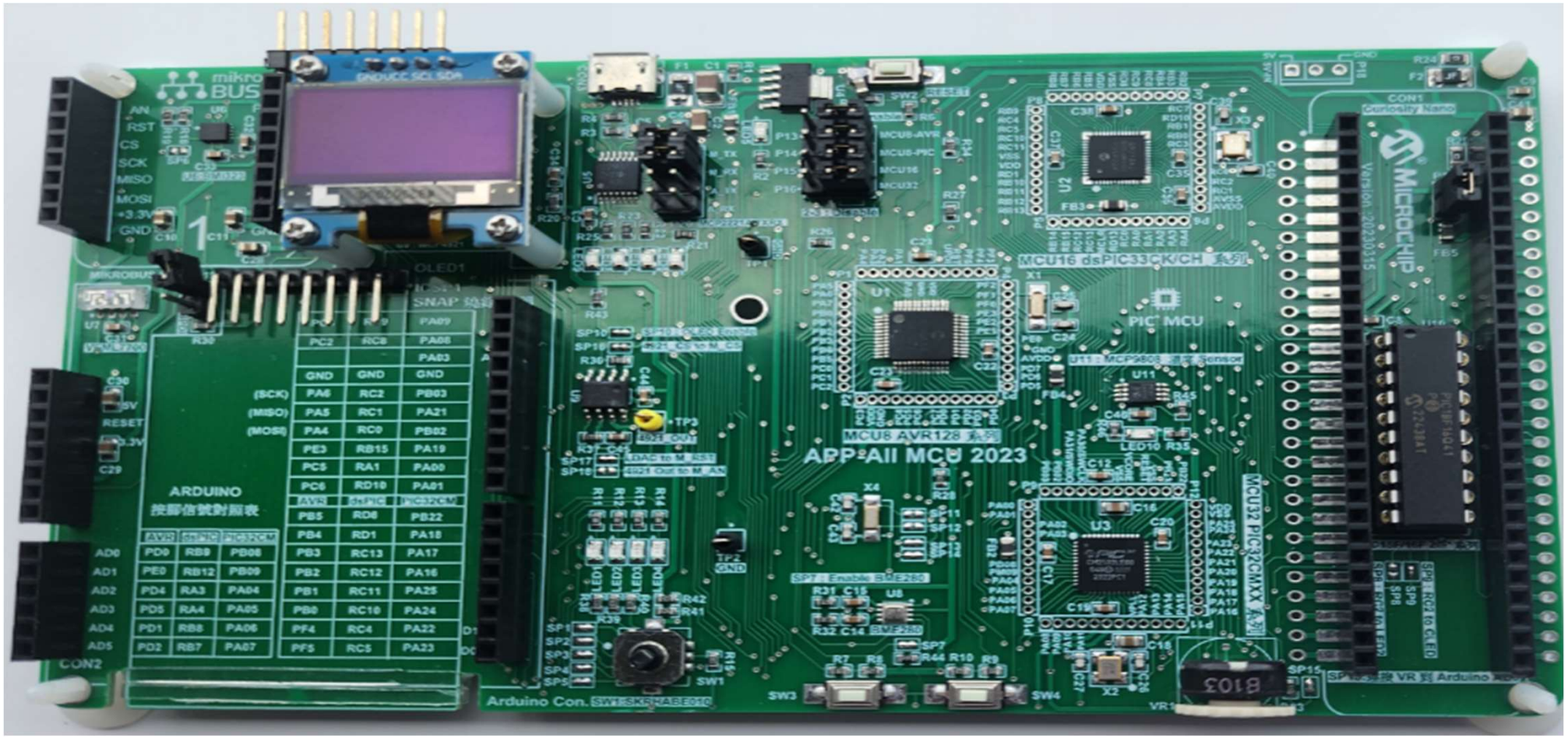
開發工具

- APP_All_MCU_2023實驗版
- MPLAB X IDE
- MPLAB-SNAP

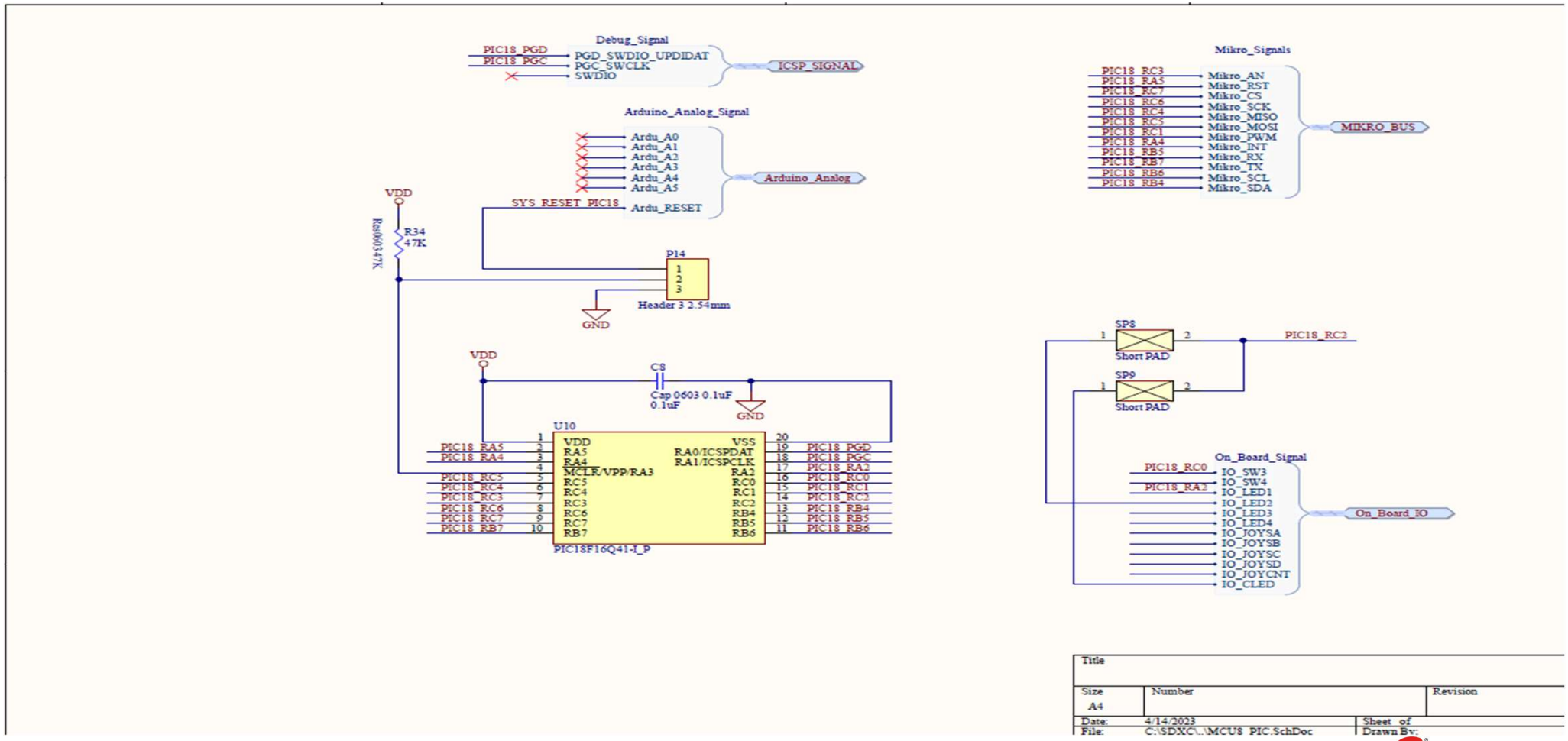
APP_All_MCU_2023實驗版功能

- 此實驗板提供AVR128DA48, dsPIC33CH256MP505, PIC32CM2532LE00048 & PIC18 & 16 20pin MCU可供實驗
- 今天以PIC16F18446為介紹基礎
- P13,15 & 16必須連接在2 & 3 pin以disable以上三個MCU, P14必須接在1 &2 pin以enable PIC16F18446
- 此實驗版提供以下零件於此板上,可實驗多種通訊界面&功能
  - **I2C** 介面的六軸**IMU -BOSCH BMI323**
  - **I2C** 介面的**Lighting Sensor –Vishay** 的**VEML7700-TT**
  - **I2C** 介面的**Humidity sensor -BOSCH BME280**
  - **I2C** 介面的溫度**Sensor –Microchip MCP9808**
  - **I2C** 介面的**OLED Display -**單色**128 * 64**
  - **SPI** 介面的**DAC –Microchip MCP4921**
  - **WS2812B One-Wire Color LED**
  - **MCP2221A** 作實驗板上的**UART** 以及**I2C** 介面轉換至**USB** 的介面**IC**
  - **ALPS** 的**SKRHABE010** 五向開關

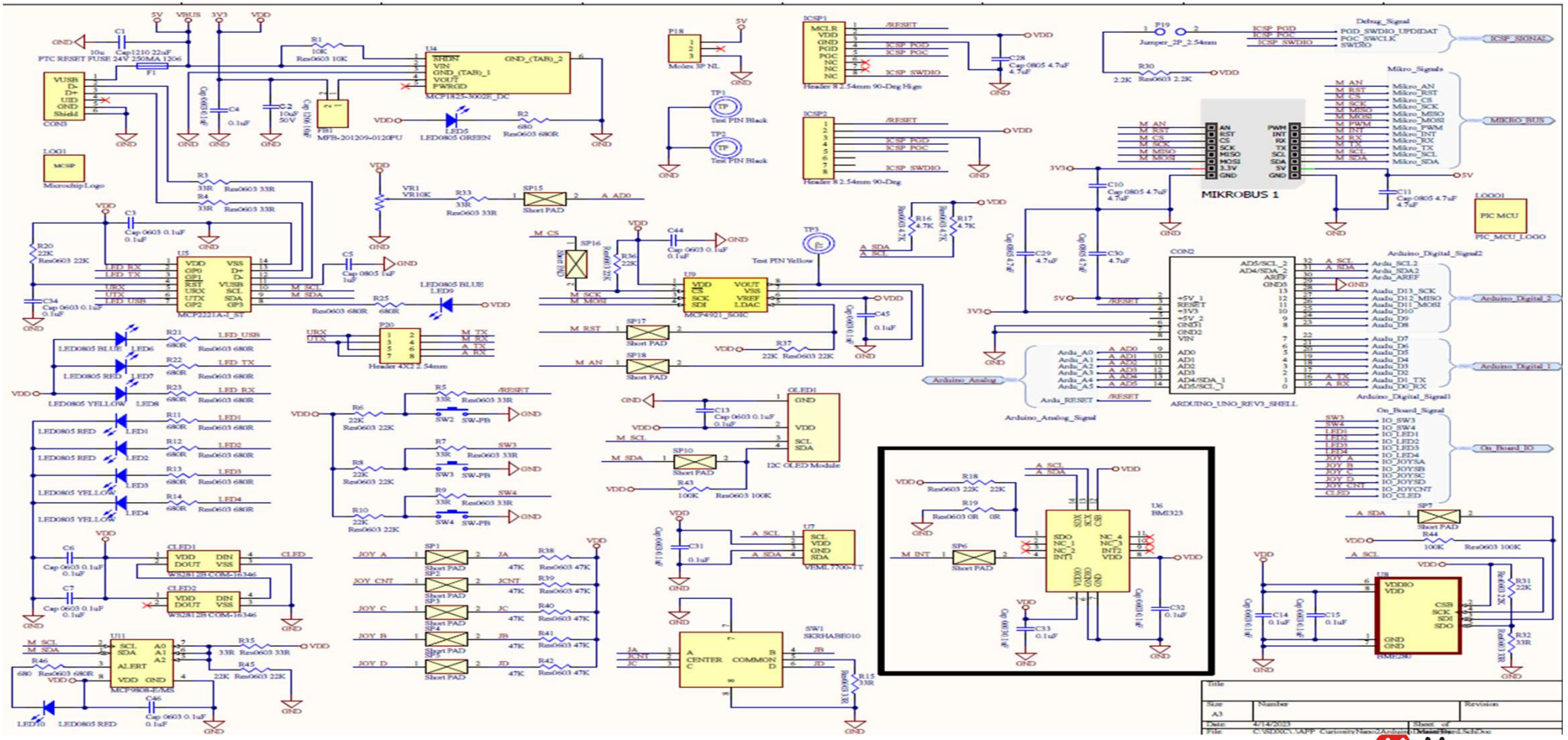
APP_All_MCU_2023實驗版



APP_All_MCU_2023實驗版線路圖

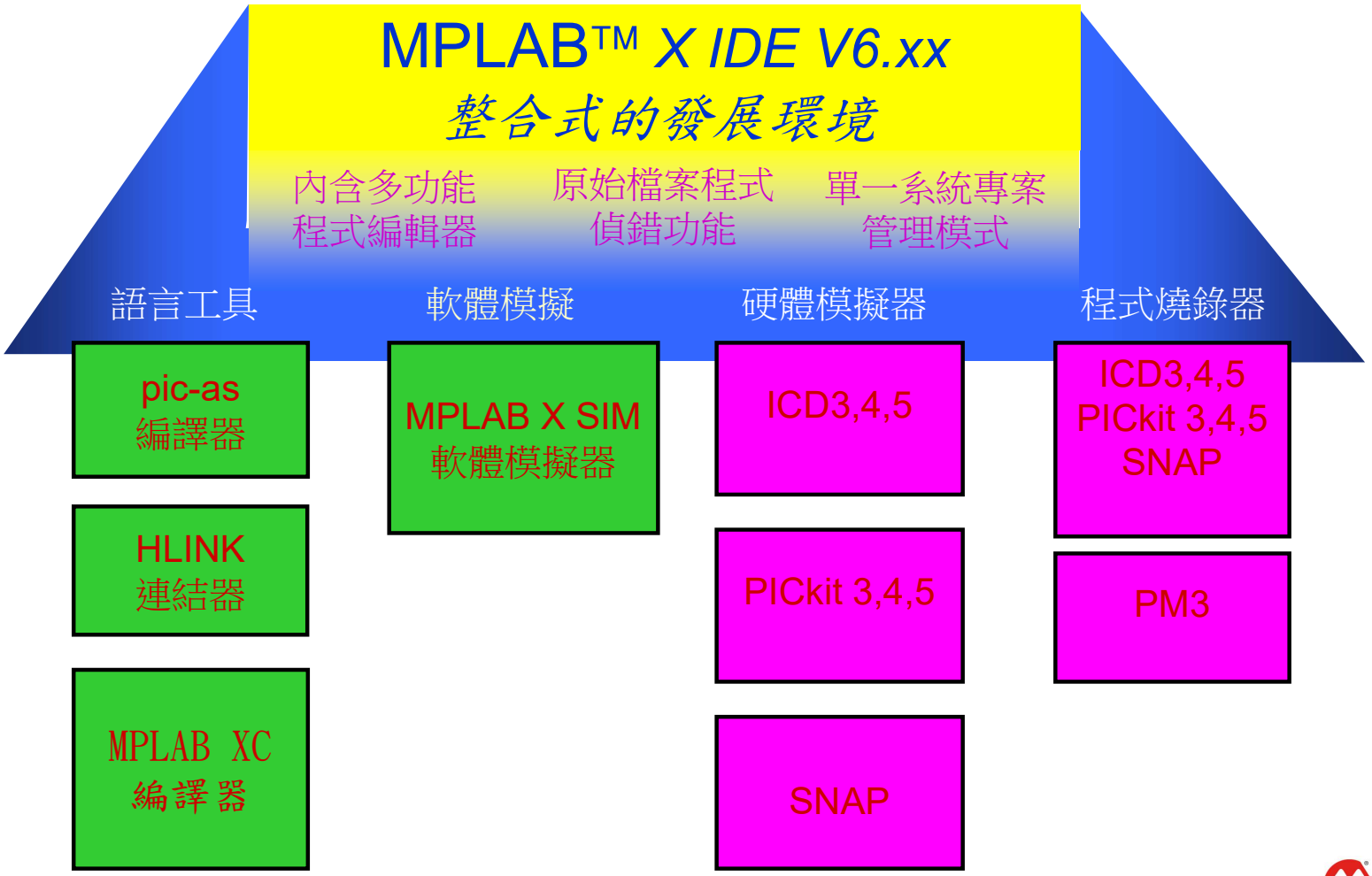


APP_All_MCU_2023實驗版線路圖

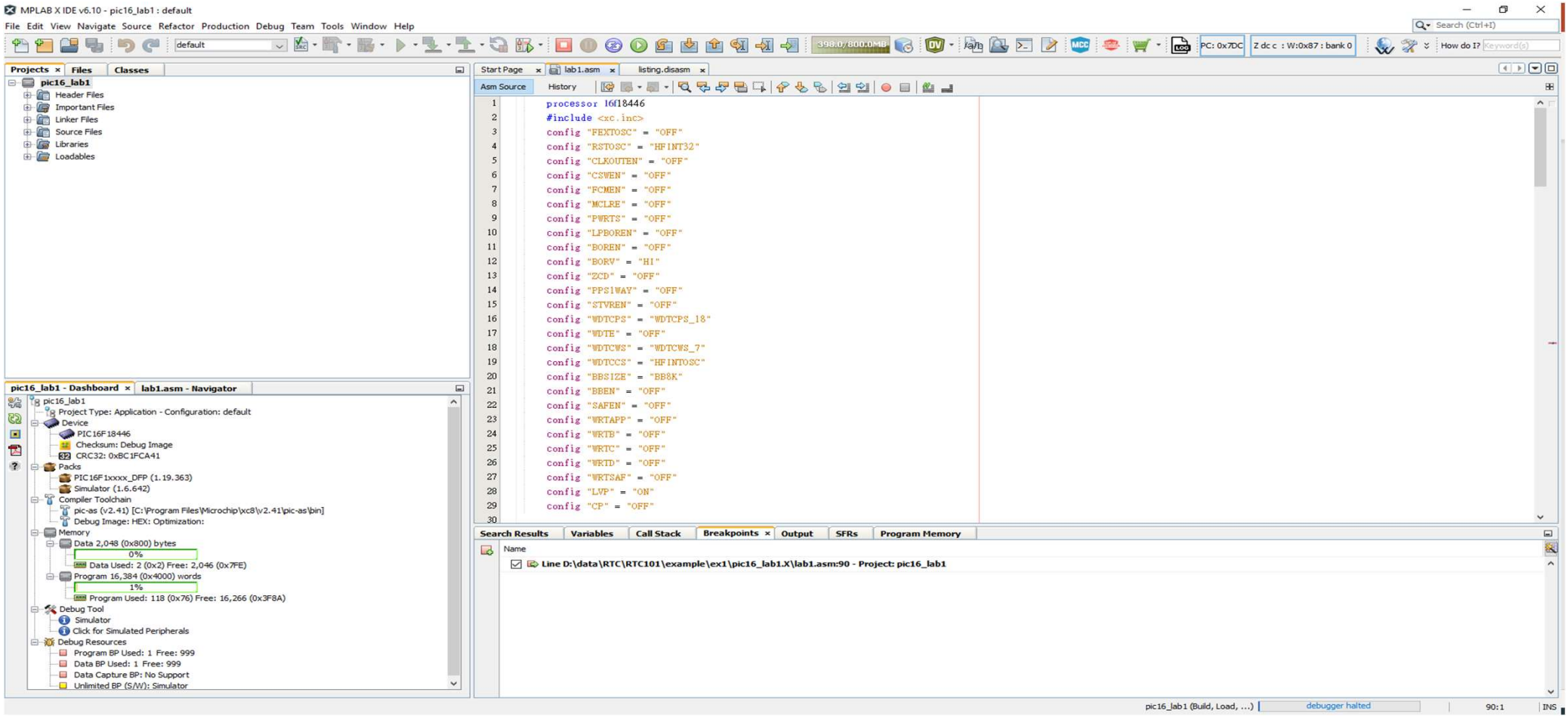


實驗板網購網站

- 杰鼎先進-Pcgome商店街：https://www.pcstore.com.tw/jdingtech
- 艾斯霸-露天網購：https://www.ruten.com.tw/store/icboxpro
- 艾斯霸-蝦皮網購：https://shopee.tw/icboxpro
- 艾斯霸-Yahoo網購：https://tw.bid.yahoo.com/booth/Y0801193781



MPLAB X IDE畫面



MPLAB X IDE & compiler相關檔案位置

- C:\Program Files\Microchip : X IDE & compiler均放於此目錄底下的子目錄
  - C:\Program Files\Microchip\MPLABX\v6.10\docs : X IDE相關的使用手冊
  - C:\Program Files\Microchip\xc8\v2.41\docs\chips : C & assembler設定chip configuration word參考文件
  - C:\Program Files\Microchip\xc8\v2.41\docs : XC8 C compiler & Assembler使用手冊
  - C:\Program Files\Microchip\xc8\v2.41\pic\dat\ini : 各chip架構,記憶體空間大小&起始位置, 特殊暫存器的宣告&其記憶體位置
  - C:\Program Files\Microchip\xc8\v2.41\pic\include\proc : 宣告各chip內部暫存器&位元組(bits)名稱, C : 使用.h file, assembler : 使用.inc file

MPLAB X IDE 軟體除錯工具：Simulator

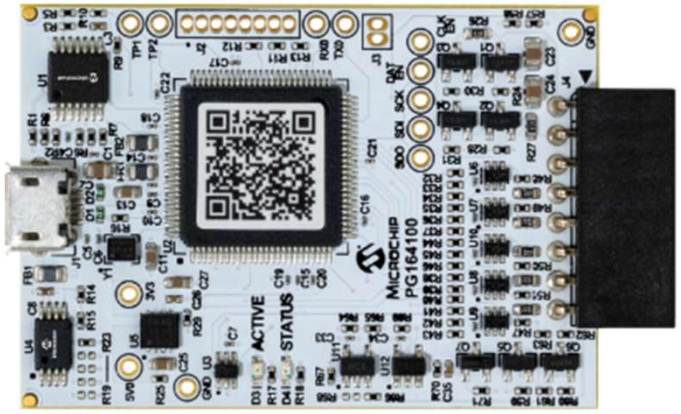
- 不需要硬體除錯工具,即可進行除錯
- 沒有硬體PCB時也可以進行除錯
- 用Stopwatch功能可以觀看一段程式執行所需時間,此功能是硬體除錯器均無提供
- 有時問題出現時,不知是硬體or軟體造成的,可使用simulator排除硬體的干擾,從而專心於軟體的除錯
- 中斷點：因simulator是使用軟體中斷,所以可設中斷點的數量可高達到1000個
- 操作方式與硬體除錯器相同
- 此工具不用錢

MPLAB X IDE軟體除錯工具：Simulator

- 可使用於PIC(8,16 & 32 bit), AVR, SAM, dsPIC MCU
- 整合於MPLAB X IDE環境之中
  - Interrupt
    - Timers
    - CCP/ECCP
    - UART
    - Change on Port RB
    - External interrupt INT pin
    - Comparator
    - A/D converter
    - EEPROM write complete
  - Core CPU
    - Reset
    - Sleep
    - Watchdog Timer
  - Peripheral
    - Timers
    - CCP/ECCP
    - Comparator
    - A/D converter
    - UART
    - EEPROM data memory

硬體除錯工具: SNAP簡介

- 整合在MPLAB X IDE & MPLAB IPE內,可做debug & 燒錄
- 可支援PIC, AVR, SAM, dsPIC & PIC32.
- 與電腦之間連線使用high speed USB 2.0
- SNAP使用300MHz SAM E70 MCU
- 與device之間的介面 : ICSP, 2- & 4-wire JTAG, & serial wire debug



- 低單價
- 不建議作為量產時的燒錄器

認識PIC16

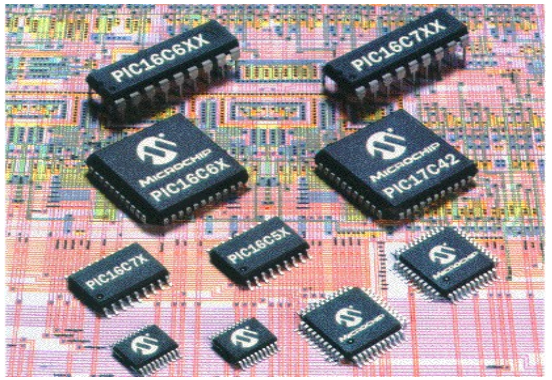
- 瞭解PIC16基本的內部工作原理
- 指令集
- 瞭解記憶體架構 & 定址模式
- 瞭解如何寫簡單程式
- 動手實驗lab1 & lab2
- 配置字

嵌入式控制器(Embedded Controller, Single Chip, Microcontroller, MCU)

- Embedded Controller ：
  - 整合產品所需的各項功能於單一晶片中
- 一般的嵌入式控制器包括以下部份
  - CPU core
  - Program Memory (ROM / OTP ROM / MASK ROM / FLASH)
  - Data Memory (RAM)
  - Data EEPROM
  - I/O
  - 各種周邊 ， 如 ADC ， PWM ， TIMER ， I^2C ， USART … 等
  - 看門狗 (Watch Dog) ， 電源異常偵測 (Brown Out Detect) ， 低電壓偵測 (Low Voltage Detect) ，及內部重置 (Internal RESET)
  - LCD Driver

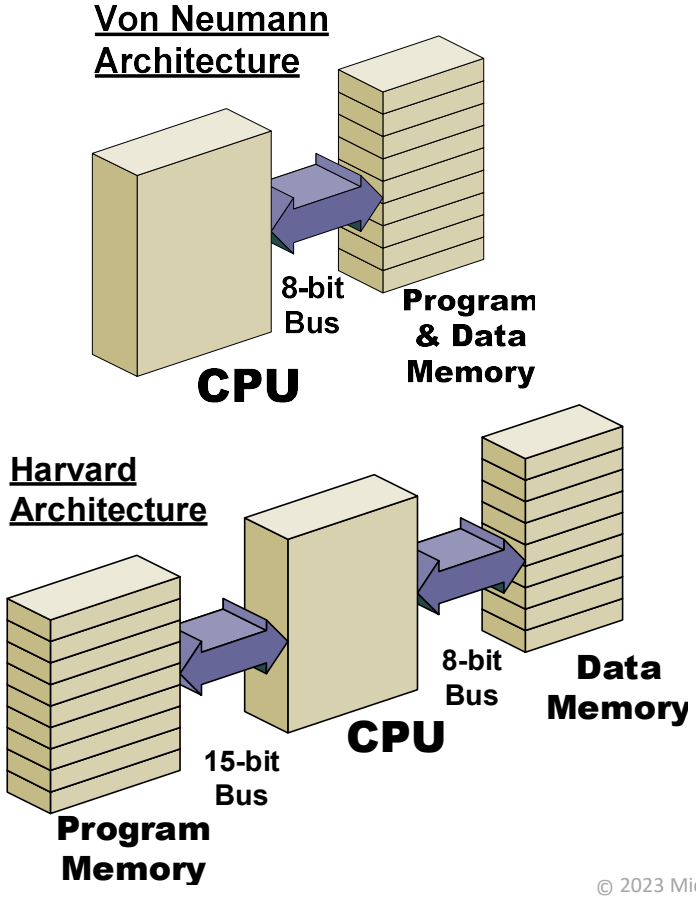
PICmicro® MCU 架構

- 俱備 RISC Microcontroller 的特點
- 高效能的 PICmicro 具備 RISC Microcontroller 的特點 , 其主要的優點如下 :
  - 使用 Harvard 管線架構
  - 大部分指令均能在單一週期值內執行完成
  - 支援指令預提取功能(Pilelining)
  - 所有指令為 “Single Word”
  - Long Word 的指令編碼
  - 指令集共有49個指令
  - 存取I/o pin, 各周邊暫存器 & 記憶體,所使用的指令均相同,無須特殊指令



PICmicro 的架構比較

PIC 使用 Harvard Architecture



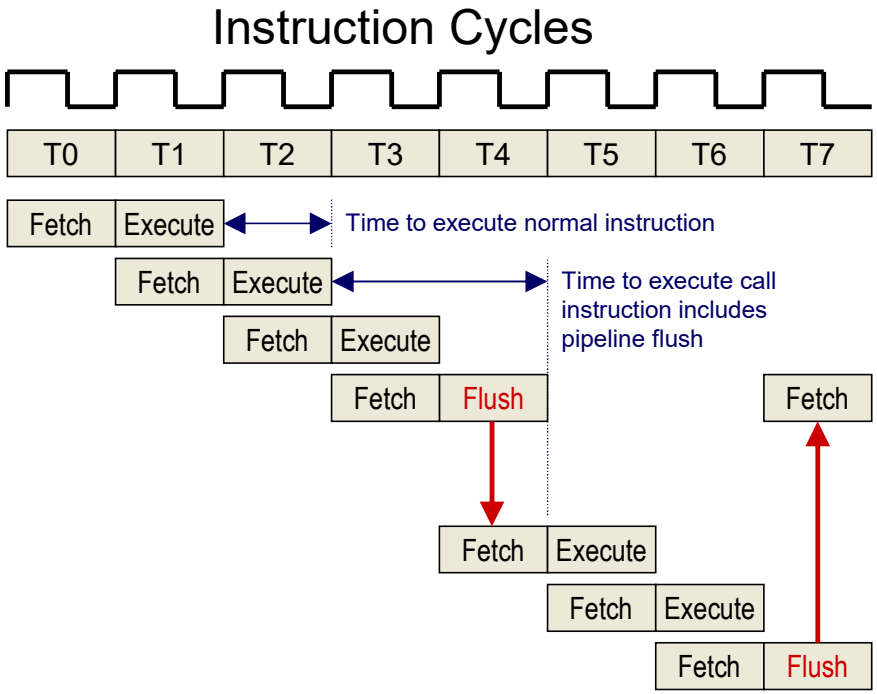
- 經由相同的記憶體來提取指令與存取資料
  - 指令與資料無法有效率的同時被處理
  - MCU 的操作效率受到此結構影響而變差
- 使用兩個不同的空間與管線來存取程式和資料
  - 增加處理資料的效能
  - 使得MCU可以具有不同寬度的程式記憶體與資料記憶體 (8 Bit 寬的 Data Memory , 12/14/16 Bit 寬的 Program Memory)

指令管線(Instruction Pipelining)

- 對大部份的MCU而言, 指令的提取與執行是連續發生但其每一個指令周期只有一個動作發生

Example Program

```
 1  MAIN  movlw   0x05
 2        movwf   REG1
 3        call SUB1
 4        addwf   REG2
             :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```



指令管線(Instruction Pipelining)

movlw 0x05

Instruction Cycles

T0

Fetch

Example Program

```
 1  MAIN  movlw    0x05
 2        movwf    REG1
 3        call SUB1
 4        addwf    REG2
          :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```

指令管線(Instruction Pipelining)

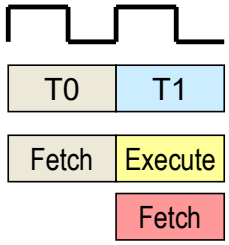
Pre-Fetched Instruction

```
movwf REG1
```

Executing Instruction

```
movlw 0x05
```

Instruction Cycles



Example Program

```
 1  MAIN  movlw   0x05
 2        movwf   REG1
 3        call SUB1
 4        addwf   REG2
          :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```


MICROCHIP

指令管線(Instruction Pipelining)

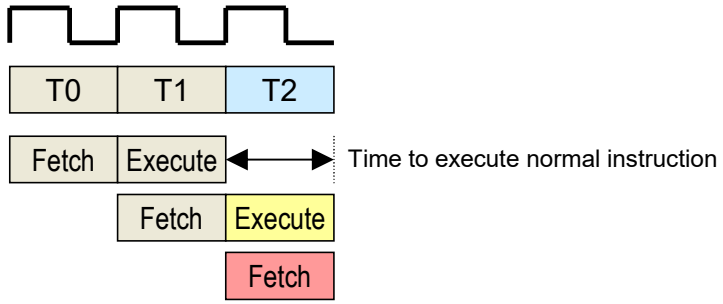
Pre-Fetched Instruction

```
call SUB1
```

Executing Instruction

```
movwf REG1
```

Instruction Cycles



Example Program

```
1   MAIN  movlw   0x05
2         movwf   REG1
3         call SUB1
4         addwf   REG2
          :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```


MICROCHIP

指令管線(Instruction Pipelining)

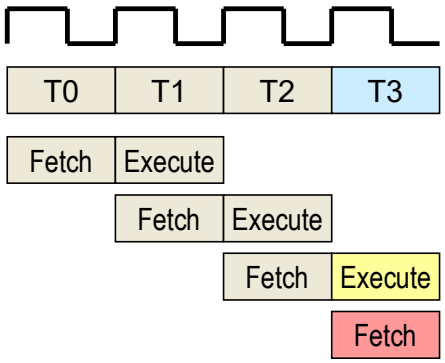
Pre-Fetched Instruction

```
addwf REG2
```

Executing Instruction

```
call SUB1
```

Instruction Cycles



Example Program

```
1   MAIN  movlw   0x05
2         movwf   REG1
3         call SUB1
4         addwf   REG2
          :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```


MICROCHIP

指令管線(Instruction Pipelining)

Pre-Fetched Instruction

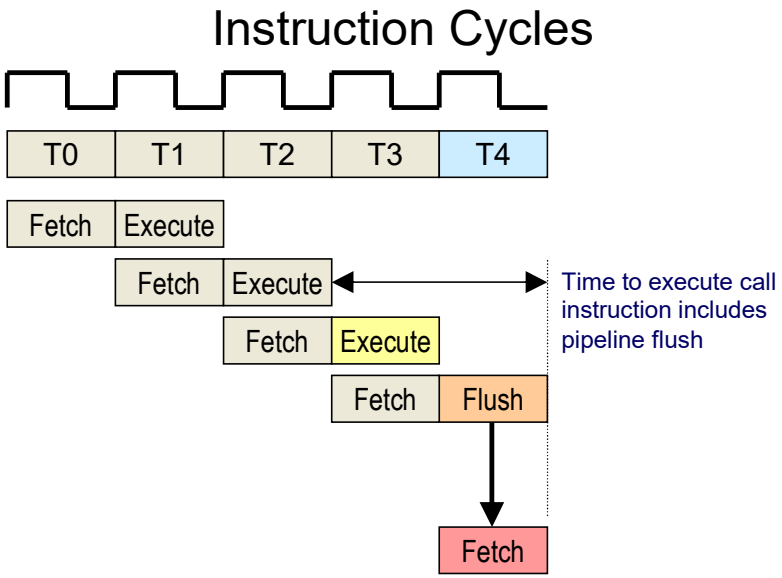
```
movf  PORTB,w
```

Executing Instruction

```
call  SUB1
```

Example Program

```
1   MAIN  movlw   0x05
2         movwf   REG1
3         call SUB1
4         addwf   REG2
          :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```



MICROCHIP

指令管線(Instruction Pipelining)

Pre-Fetched Instruction

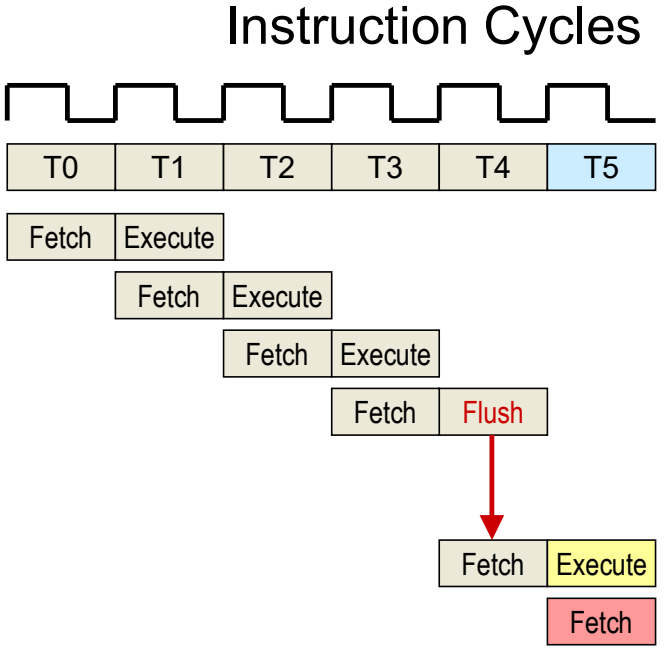
```
return
```

Executing Instruction

```
movf PORTB,w
```

Example Program

```
1   MAIN  movlw   0x05
2         movwf   REG1
3         call SUB1
4         addwf   REG2
          :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```



MICROCHIP

指令管線(Instruction Pipelining)

Pre-Fetched Instruction

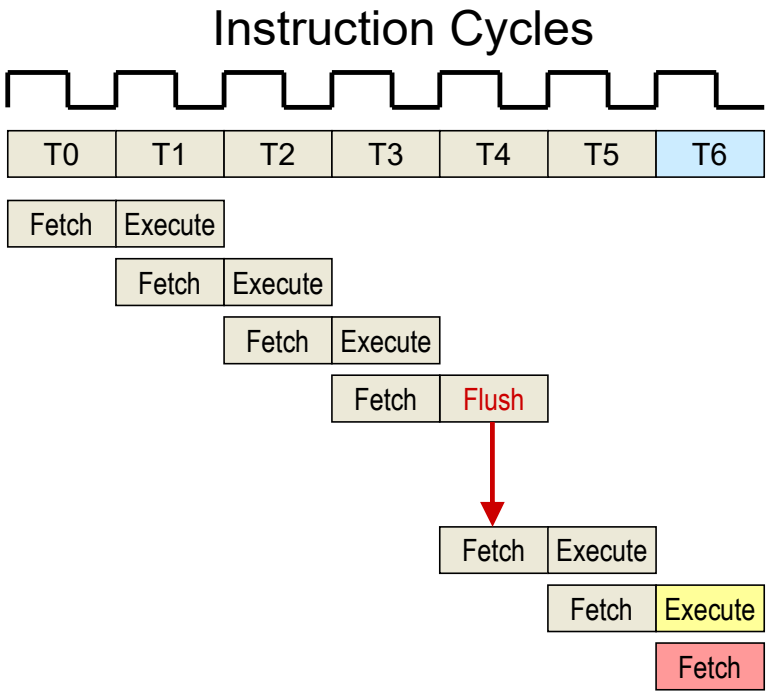
```
movf  PORTC,w
```

Executing Instruction

```
return
```

Example Program

```
1   MAIN  movlw    0x05
2         movwf    REG1
3         call SUB1
4         addwf    REG2
          :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```



指令管線(Instruction Pipelining)

Pre-Fetched Instruction

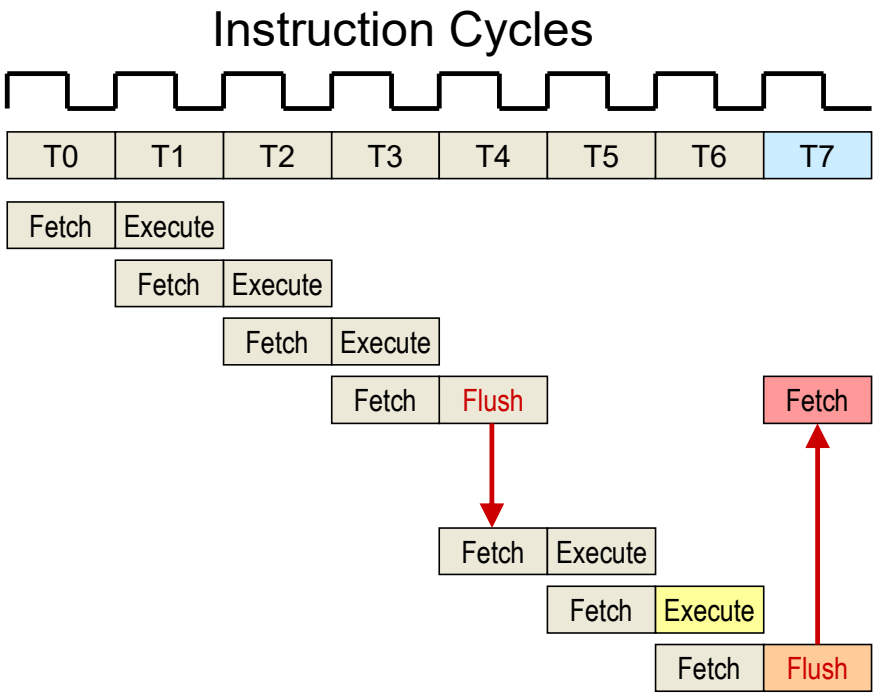
```
addwf  REG2
```

Executing Instruction

```
return
```

Example Program

```
 1  MAIN  movlw   0x05
 2        movwf   REG1
 3        call SUB1
 4        addwf   REG2
          :
51  SUB1  movf PORTB,w
52        return
53  SUB2  movf PORTC,w
54        return
```



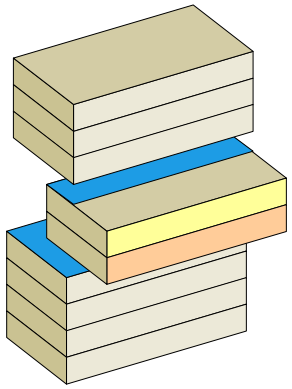
MICROCHIP

長指令（Long Word Instruction）

- 將指令與資料匯流排分開的方式， 使得PICmicro可以依照需求來調整所需的指令寬度
- 8 bit PICmicro 指令寬度，依不同型號，分別為 12， 14 or 16-bits，稱為一個 Instruction Word，每一指令只佔一個 Word
- 8 bit PICmicro 的資料匯流排(Data Bus)寬度是 8 bits
- 若於 PIC16 具備 2K x 14 words 的程式記憶體可完成的工作約相當於其他有4K bytes記憶體的MCU
- 單一周期的指令運作使MCU的處理能力提高許多

長指令（Long Word Instruction）

8-bit Program Memory



典型8-bit MCU的 8-bit指令

例子: ‘Load Accumulator A’:

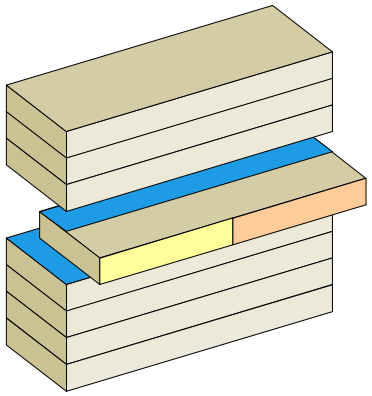
- 占用2個程式記憶體位置
- 執行此指令需2個指令週期

```
ldaa     #k
```

1	0	0	0	0	1	1	0
k	k	k	k	k	k	k	k

- 頻寬受限
- 需要增加記憶體空間

14-bit Program Memory



14-bit Instruction on PIC16 8-bit MCU

Example: ‘Move Literal to Working Register’

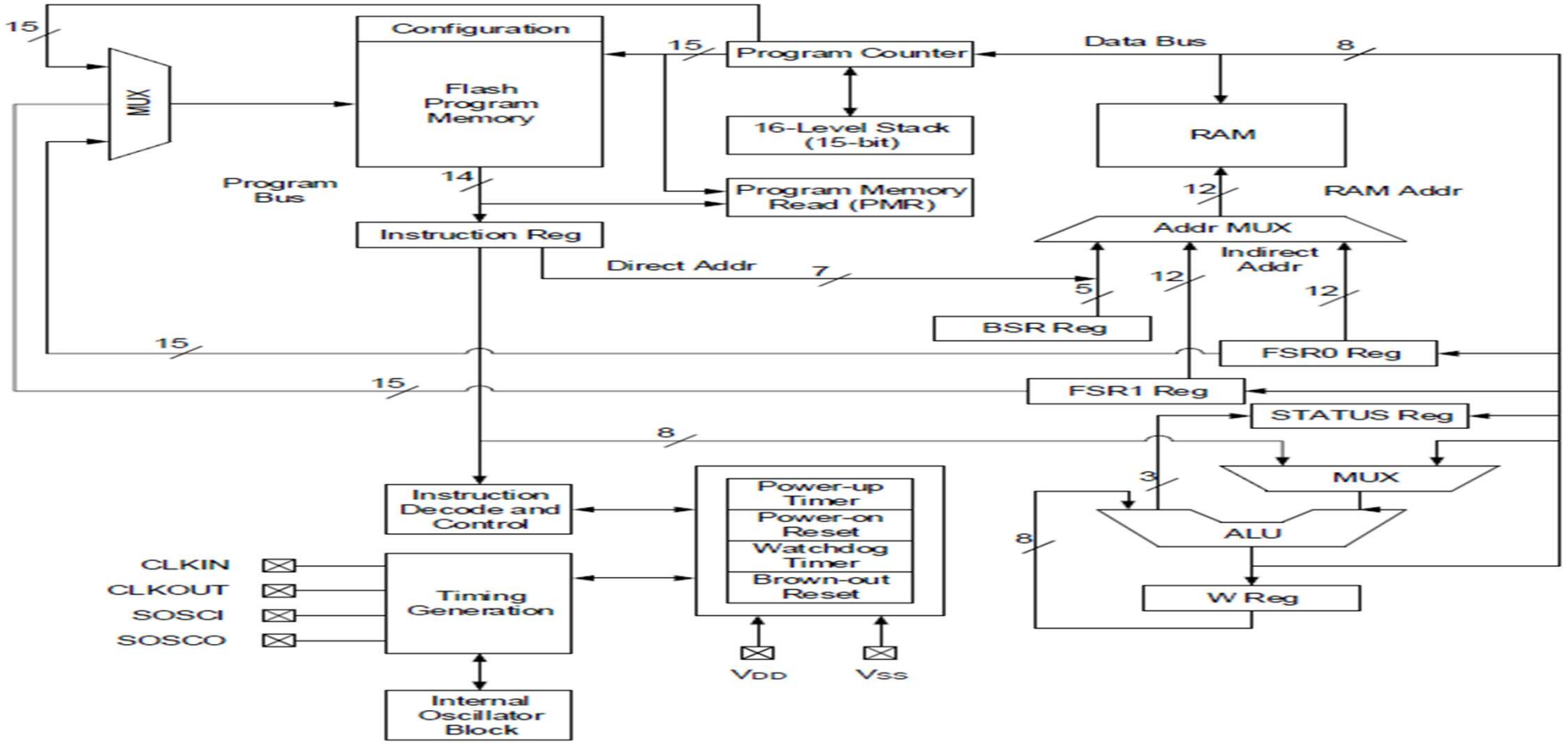
- 占用1個程式記憶體位置
- 執行此指令需1個指令週期

```
movlw    k
```

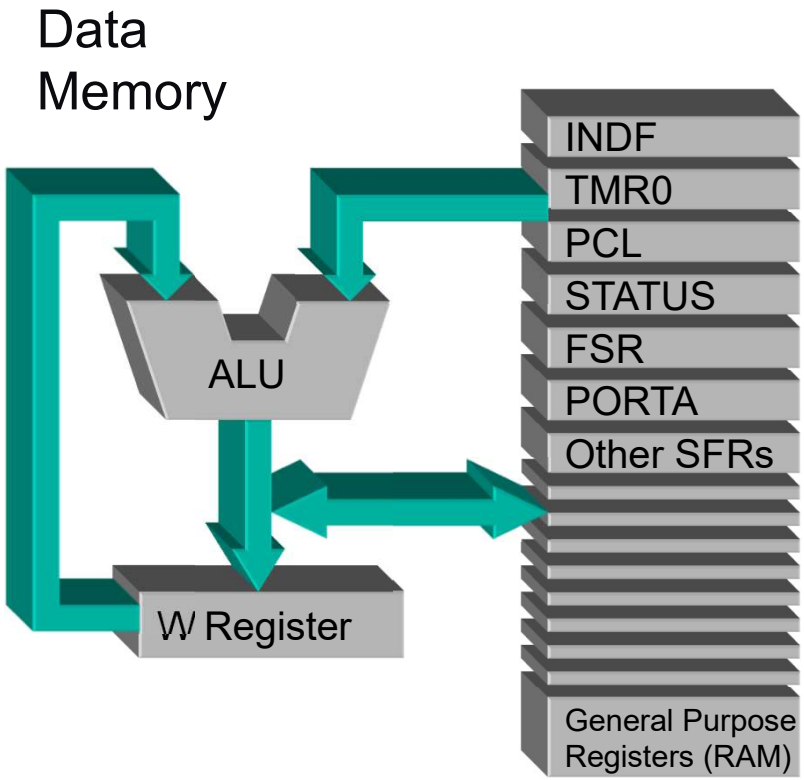
1	1	0	0	0	0	k	k	k	k	k	k	k	k

- 不同的資料匯流排允許匯流排有不同的寬度
- 2k x 14 約等於 4k x 8

PIC架構方塊圖



PIC架構 應用Register File 的概念



14-bit 指令格式的範例:資料存取

- 暫存器庫(Register Bank)由許多一般暫存器與特殊用途暫存器 (SFR) 組成
- 所有周邊(如：I/O)的操作與暫存器相同
- 任何一個暫存器都可被用於存取Register File的指令所操作
- Long word 的指令編碼方式使得指令與暫存器的位址僅以一個word即可表示

指令集

BYTE-ORIENTED OPERATIONS		
ADDWF	f, d	Add WREG and f
ADDWFC	f, d	Add WREG and CARRY bit to f
ANDWF	f, d	AND WREG with f
ASRF	f, d	Arithmetic Right Shift
LSLF	f, d	Logical Left Shift
LSRF	f, d	Logical Right Shift
CLRF	f	Clear f
CLRW	–	Clear WREG
COMF	f, d	Complement f
DECF	f, d	Decrement f
INCF	f, d	Increment f
IORWF	f, d	Inclusive OR WREG with f
MOVF	f, d	Move f
MOVWF	f	Move WREG to f
RLF	f, d	Rotate Left f through Carry
RRF	f, d	Rotate Right f through Carry
SUBWF	f, d	Subtract WREG from f
SUBWFB	f, d	Subtract WREG from f with borrow
SWAPF	f, d	Swap nibbles in f
XORWF	f, d	Exclusive OR WREG with f
BYTE-ORIENTED SKIP OPERATIONS		
DECFSZ	f, d	Decrement f, Skip if 0
INCFSZ	f, d	Increment f, Skip if 0
BIT-ORIENTED SKIP OPERATIONS		
BTFSC	f, b	Bit Test f, Skip if Clear
BTFSS	f, b	Bit Test f, Skip if Set
BIT-ORIENTED FILE REGISTER OPERATIONS		
BCF	f, b	Bit Clear f
BSF	f, b	Bit Set f

LITERAL OPERATIONS		
ADDLW	k	Add literal and WREG
ANDLW	k	AND literal with WREG
IORLW	k	Inclusive OR literal with WREG
MOVLB	k	Move literal to BSR
MOVLP	k	Move literal to PCLATH
MOVLW	k	Move literal to W
SUBLW	k	Subtract W from literal
XORLW	k	Exclusive OR literal with W
CONTROL OPERATIONS		
BRA	k	Relative Branch
BRW	—	Relative Branch with WREG
CALL	k	Call Subroutine
CALLW	—	Call Subroutine with WREG
GOTO	k	Go to address
RETFIE	k	Return from interrupt
RETLW	k	Return with literal in WREG
RETURN	—	Return from Subroutine
INHERENT OPERATIONS		
CLRWDT	—	Clear Watchdog Timer
NOP	—	No Operation
RESET	—	Software device Reset
SLEEP	—	Go into Standby or Idle mode
TRIS	f	Load TRIS register with WREG
C COMPILER OPTIMIZED		
ADDFSR	n, k	Add Literal k to FSRn
MOVIW	n, mm	Move Indirect FSRn to WREG with pre/post inc/dec modifier, mm
	k[n]	Move INDFn to WREG, Indexed Indirect
MOVWI	n, mm	Move WREG to Indirect FSRn with pre/post inc/dec modifier, mm
	k[n]	Move WREG to INDFn, Indexed Indirect

資料移動指令
(MOVF , MOVWF)

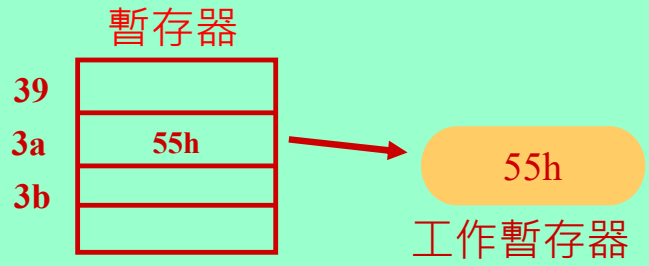
MOVF

- 語法 : MOVF f , d
- 操作 : 將所指到的暫存器 (f) 的內容傳送到工作暫存器 (w)

例:

movf h'3A',W

將位址 $3A_{(16)}$ 的暫存器內容傳送到工作暫存器 (w) 中。



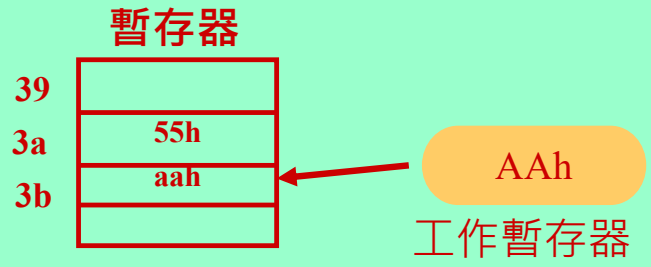
MOVWF

- 語法 : MOVWF f
- 操作 : 將工作暫存器 (w) 的內容傳送所指到的暫存器 (f)

例:

movwf h'3B'

將工作暫存器 (w) 中的內容傳送到位址為 $3B_{(16)}$ 的暫存器中。



資料遞減指令
(DECF , DECFSZ)

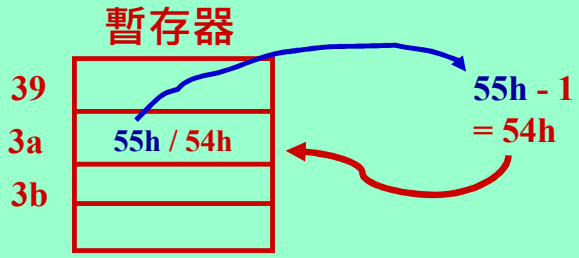
DECF

- 語法： DECF　　f , d
- 操作 : 將所指到的暫存器 (f) 的內容減一後回存到 (d)

例:

decf　h'3A', F

將位址 $3A_{(16)}$的暫存器內容減一後回存到位址為$3A_{(16)}$暫存器中。



DECFSZ

- 語法： DECFSZ　　f , d
- 操作 :將所指到的暫存器 (f) 的內容減一後回存到 (d)，並測試減一後的結果是不是等於零以決定是否跳過下一個指令。
  - 結果不等於零：執行下一個指令
  - 結果等於零：忽略下一個指令而直接執行下下一個指令

例:

```
decfsz   h'3A' , F
goto     結果不等於零
call     結果等於零
```

位元清除、設定指令 (BCF , BSF)

BCF

- 語法 : BCF f , b
- 操作 : 將所指到的暫存器 (f) 中的第 (b) 個位元設定為 0。

例:

```
PORTA   equ    h'05'
        bcf    PORTA , 2
```

將位址 $05_{(16)}$ 的暫存器 (PORTA) 的 bit 2 清除為 “0” 。

```
        bcf    h'3a' , 7
```

將位址 $3a_{(16)}$ 的暫存器的 bit 7 清除為 “0” 。

BSF

- 語法 : BSF f , b
- 操作 : 將所指到的暫存器 (f) 中的第 (b) 個位元設定為 1。

例:

```
PORTA   equ    h'05'
        bsf    PORTA , 2
```

將位址 $05_{(16)}$ 的暫存器 (PORTA) 的 bit 2 設定為 “1” 。

```
        bsf    h'3a' , 7
```

將位址 $3a_{(16)}$ 的暫存器的 bit 7 設定為 “1” 。

位元測試指令
(BTFSC , BTFSS)

BTFSC

- 語法 : BTFSC f , b
- 操作 :測試所指到的暫存器 (f) 的第 (b) 個的位元內容是不是等於 “0” 以決定是否跳過下一個指令。
  - Bit (b) 不等於 “0” ：執行下一個指令 (條件不成立) 。
  - bit (b) 等於 “0” ：忽略下一個指令而直接執行下下一個指令，也就是說其測試條件成立。

例:

```
btfsc   h’3A’ , 5
goto    結果不等於 “0”
goto    結果等於 “0”
```

BTFSS

- 語法 : BTFSS f , b
- 操作 :測試所指到的暫存器 (f) 的第 (b) 個的位元內容是不是等於 “1” 以決定是否跳過下一個指令。
  - Bit (b) 不等於 “1” ：執行下一個指令 (條件不成立) 。
  - bit (b) 等於 “1” ：忽略下一個指令而直接執行下下一個指令，也就是說其測試條件成立。

例:

```
btfss   h’3A’ , 5
goto    結果不等於 “1”
goto    結果等於 “1”
```

常數載入、運算指令
(MOVLW , ADDLW , SUBLW, IORLW , ANDLW)

MOVLW

- 語法 : MOVLW　　H’3A’
- 操作 :將所指定的常數 (3A) 載入到工作暫存器 (W)中。

IORLW

- 語法 : IORLW　B’11110000’
- 操作 :將所指定的常數 (F0)與工作暫存器 (W)的內容做 OR Gate 的運算後，放回工作暫存器 (W) 。

SUBLW

- 語法 : SUBLW　　H’3A’
- 操作 :將所指定的常數 (3A)減去工作暫存器 (W)的內容，其差放回工作暫存器 (W)

ANDLW

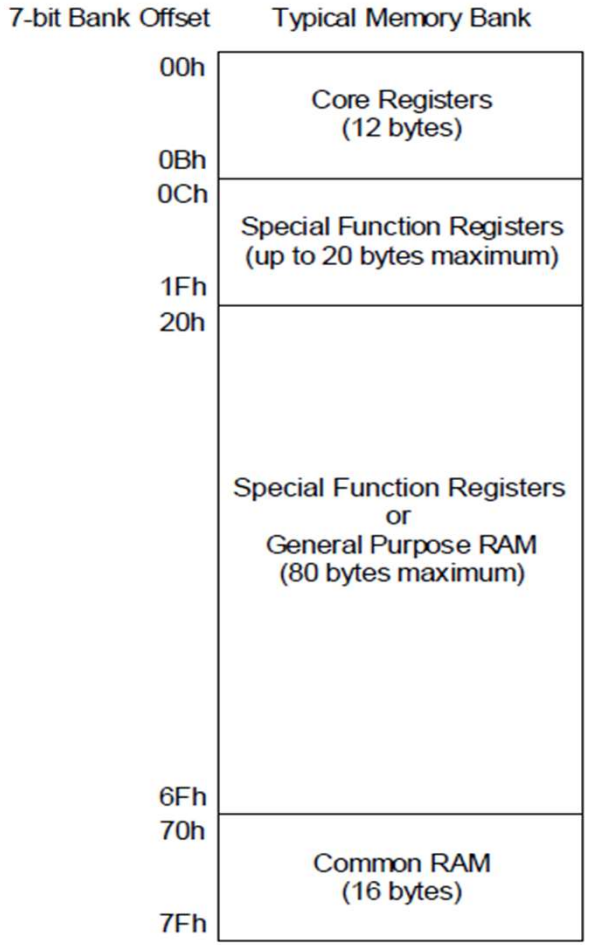
- 語法 : ANDLW　B’00001111’
- 操作 :將所指定的常數 (0F)與工作暫存器 (W)的內容做 AND Gate 的運算後，放回工作暫存器 (W) 。

PIC16 定址模式

- 存取資料記憶體:
  - 直接定址 `addwf <data_address>, <d>`
  - 間接定址 `addwf INDFx, <d>`
- 存取程式記憶體:
  - 立即定址 (Literal) `movlw <constant>`
  - 絕對定址 `goto <program_address>`
  - 相對定址 `addwf PCL,f`
  - 間接定址 `moviw k[FSRx]`

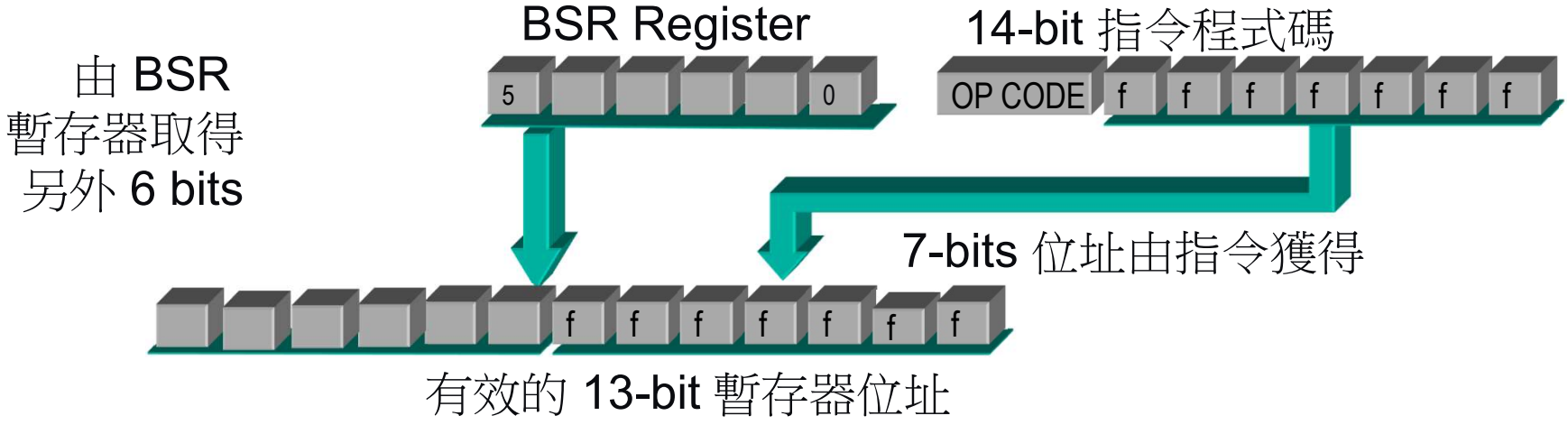
典型資料記憶體庫(Bank)

- 12個 core register : 與CPU運作相關的register, 此為各bank都相通,所以要存取此區域的register不需要切換bank
- Special function register(SFR) : 位於每個bank的0xc ~ 0x1F, 緊接於core register之後的20個byte. 其所含的registe與MCU的各種功能有關, 如timer, ADC…, 各bank所含的register不同,故要存取此區域的register需要切換bank
- General purpose RAM : 位於每個bank 0x20 ~ 0x6F之間, 共80個byte,其功能就是儲存程式中會使用到的變數.但有些chip,如含有USB or CAN功能的chip, SFR會延伸到此區域,需要切換bank
- Common RAM :位於每個bank 0x70 ~ 0x7F, 適合用在程式中到處會使用到的全域變數,不需要切換bank



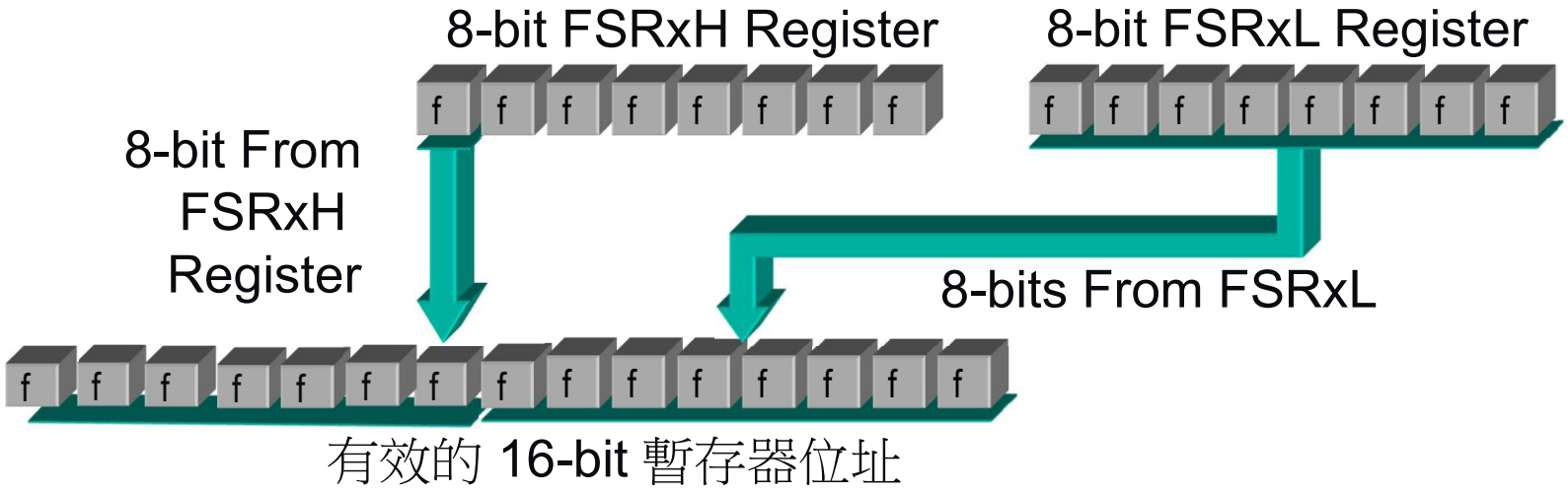
資料記憶體：直接定址

- Mid-Range (14-bit 指令PIC) 的每一個暫存器庫為 128 Bytes
- PIC16FXXX 的最大DATA RAM大小為 4 個 BANK (512 Bytes), PIC16FXXXX的最大DATA RAM大小為 64 個 BANK (8K Bytes),但現在chip實際最大為32個BANK(4K Bytes)
- 資料記憶體的有效位址為 13 bits
- 7-bit 的位址直接來自於指令中
- 6-bit 由 BSR 暫存器獲得, 以定址到完整的 64 個 BANK



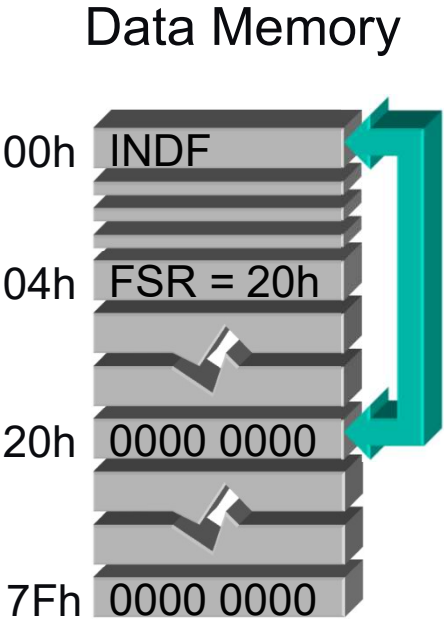
資料記憶體：間接定址

- Mid-Range (14-bit PIC) 的資料記憶體有效位址為 13 bits, 最大定址範圍是 8K Bytes (64 BANKs)
- 資料記憶體13-bit 的位址由 FSRxL & FSRxH (File Select Register) 獲得
- FSR 可提供 8K Bytes資料記憶體的定址能力 (64 BANKs)
- 若FSRxH bit 7設為1, 則可定址到程式記憶體



資料記憶體: 間接定址的使用範例

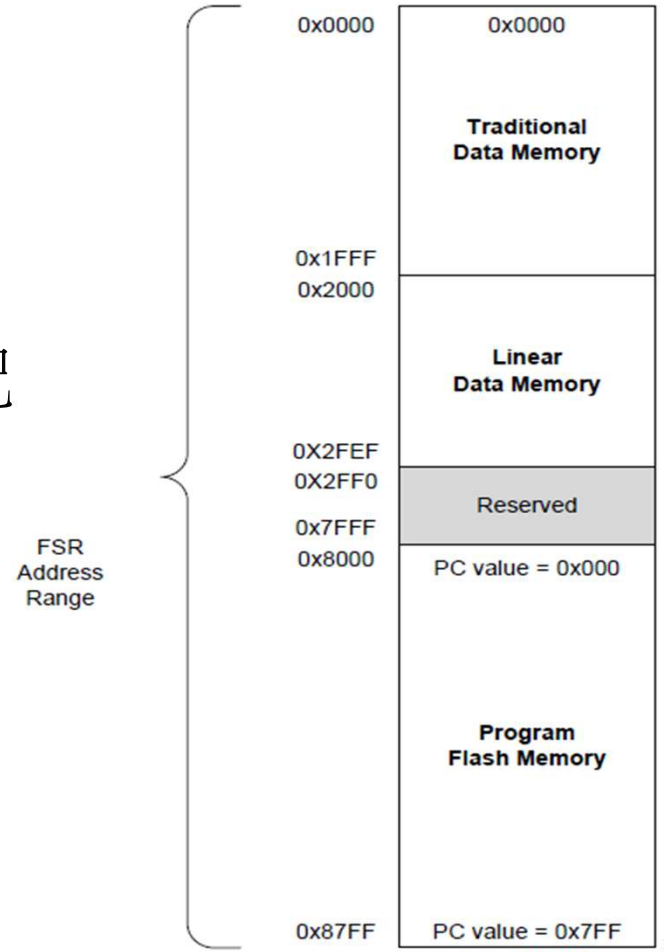
- 清除由位址 0x20 to 0x7F 的資料暫存器
  - 所需間接位址被寫入 FSR0 中
  - 當 INDF0 被當成運算元(Operand)時, 被 FSR0暫存器內容所指到的位址才是真正操作對象




```
        clrf    FSR0H
        movlw   0x20
        movwf   FSR0L
LOOP    clrf    INDF0
        incf    FSR0L,F
        btfss   FSR0L,7
        goto    LOOP
        <next instruction>
```

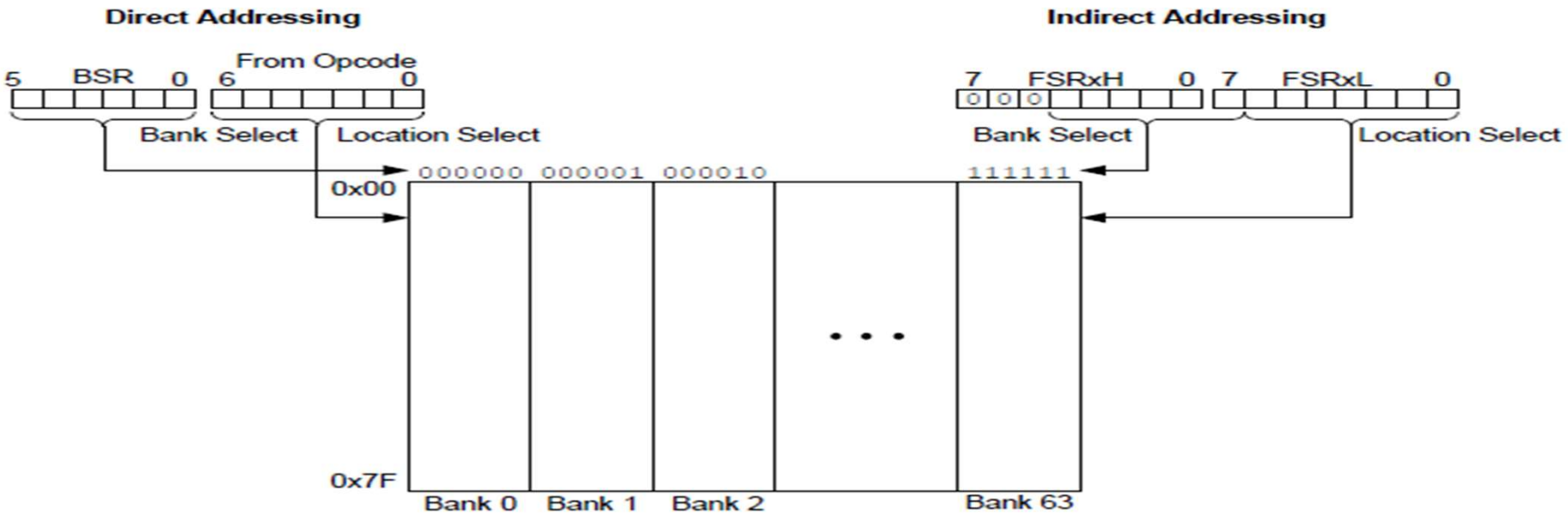
記憶體: 間接定址

- 現在不論是資料 or 程式記憶體均可使用間接定址模式
- 傳統PIC 8bit MCU間接定指模式只可以定址到資料記憶體(RAM)空間
- 傳統資料記憶體現在可以以線性資料記憶體的方式存取,資料記憶體位址從 0x2000 ~ 0x2FEF
- 程式記憶體位址從 0x8000 ~ 0x87FF



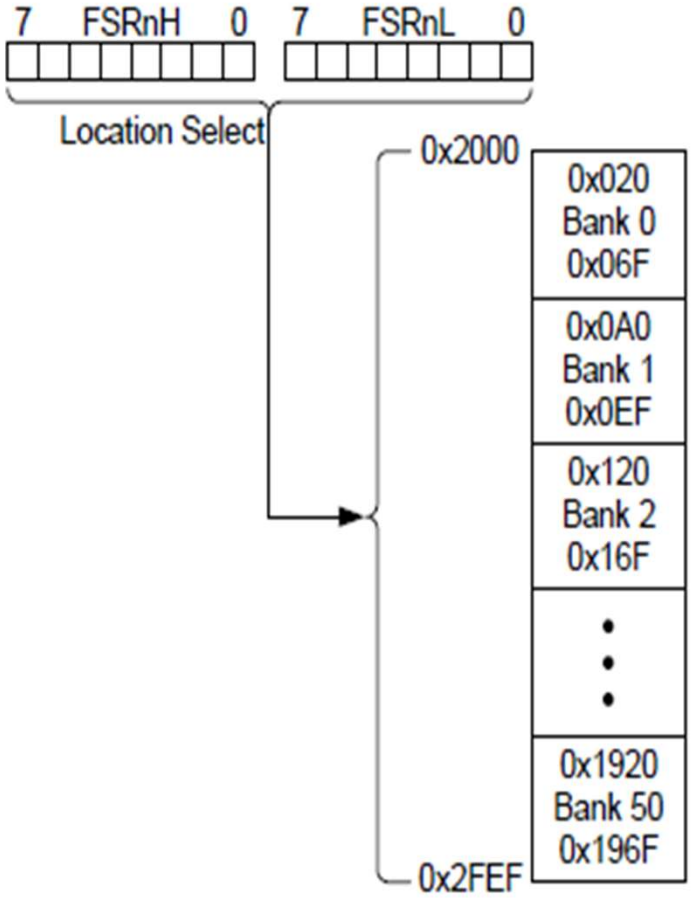
資料記憶體：傳統直接 & 間接定址的比較

- 直接定址以BSR register切換bank
- 間接定址以FSRX register的內容為定址的位置



線性資料記憶體

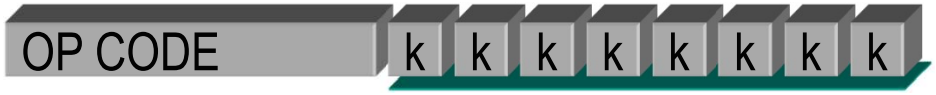
- 此線性資料記憶體空間址是以FSRx register存取資料
- 空間位址為 0x2000 ~ 0x2FEF
- 此空間為一虛擬空間,實際是指到每個bank的一般暫存器(GPR) 0x20 ~ 0x6F,每個bank共80 bytes



程式記憶體：立即值定址

- 將 8-bit 的常數(Literal) 直接置於指令的bit-0 至 bit-7
- 配合不同的OP Code 將常數做為運算元
- 可直接將常數與 W 暫存器運算
- 使用於常數操作指令, 如 movlw, addlw, retlw,, 等

14-bit Instruction for Literal Instructions



程式記憶體: 程式計數器 (PC) 的絕對定址

- 使用於 CALL 及 GOTO 等控制指令
- 藉由改變 PC (Program Counter) 來更改程式的執行位址
- CALL 和 GOTO 指令直接將 11-bit 位址置於指令中, 可於相同的 2K 程式頁直接操作
- 若範圍超過 2K 的程式頁範圍, 需更新 PCLATH 的內容
  - MOVLP TARGET_ADDR
  - CALL TARGET_ADDR

14-bit Instruction for call and goto

程式記憶體: 程式計數器 (PC) 的絕對定址

- 虛指令 “PAGESEL” 可以幫助您完成因超過 2K 程式範圍所需對 PCLATH 的設定工作
- 若欲 Call 或 Goto 的位址在 2K 的 Page 範圍之外或不確定是否同一 Page, 可用 PAGESEL 來幫助程式的處理
- 例如 : 要前往的位址為 TARGET_ADDR
  - 使用下列的敘述

```
PAGESEL   TARGET_ADDR
CALL      TARGET_ADDR
```

  - 相當於

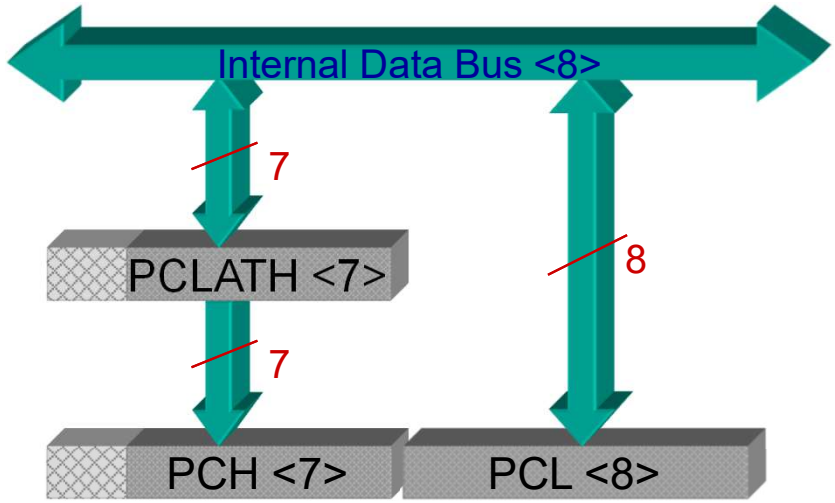
```
MOVLP     TARGET_ADDR
CALL      TARGET_ADDR
```

程式記數器(PC) 的相對定址

- 以計算PC (程式計數器) 來達成相對定址的工作
- 應用於計算式 goto 的程式計巧
  - 將一個偏移值直接與 15-bit 的Program Counter 相加, 以獲得最終的跳躍位址
  - 可於整個 32K 的程式範圍中操作
  - 使用 PCLATH 來保持位址的 7 個高位元
  - 當 PCL 被當成運算的目的地時, 將PCLATH 以及運算結果整個一次寫入15-Bit 的 Program Counter

程式記數器(PC) 的相對定址

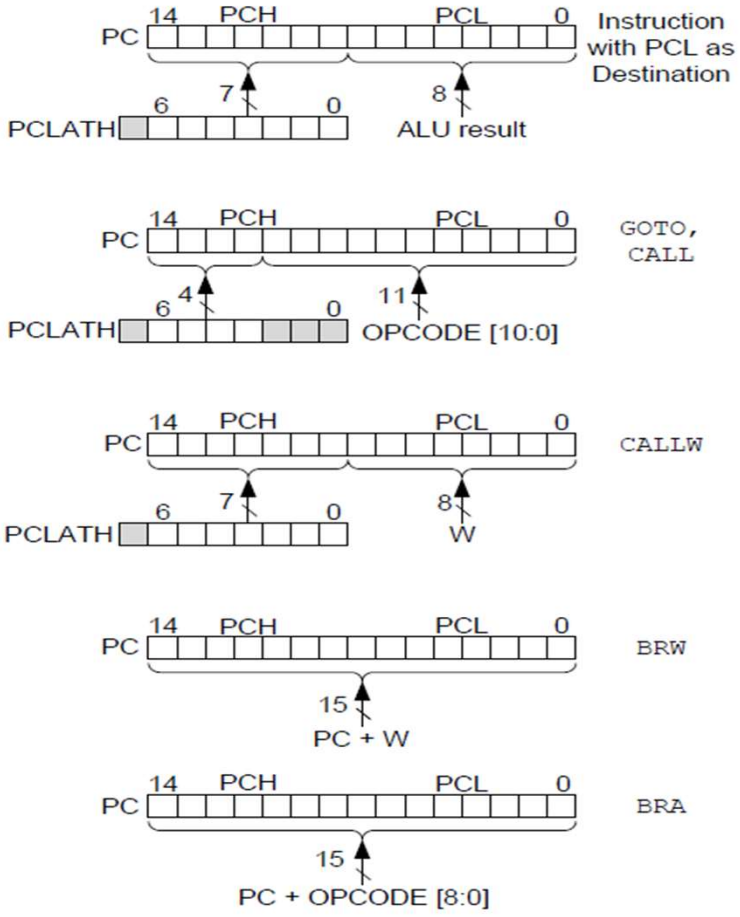
- 先將目標位址的 High Byte 寫入 PCLATH.
- 再將 low byte 寫入 PCL, 如此將會把整個15-Bit 的值寫入程式記數器 (PC)
- 若要讀取PC的值
  - 可由讀取 PCL 來得到PC 的 Low Byte
  - 讀取 PCLATH 的內容並不會得到當時的PCH值 (Bit 8 - 14 of PC)



15-Bit Program Counter

程式記數器(PC) 的相對定址

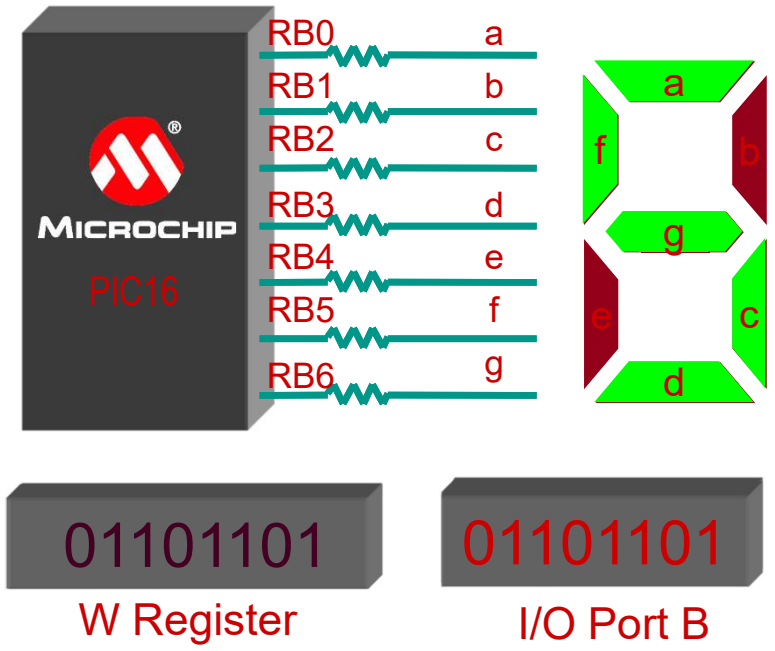
- 以PCL為資料目的地, PCL資料從指令,PCH的資料來自PCLATH bit 6 ~ 0
- GOTO, CALL指令: PC bit 10 ~ 0來自指令, bit 14 ~ 11來自PCLATH bit 6 ~ 3
- CALLW指令: PCL資料來自Wreg, PCH來自PCLATH bit 6 ~ 0
- BRW指令: PC的內容+ Wreg然後再存回PC
- BRA指令: PC的內容+指令內bit 8 ~ 0的值,然後再存回PC, 此9 bit資料是有號數



相對定址(Relative Addressing): lookup table

```
PSECT resetVec, class=CODE,delta=2
resetVec:
        pagesel start
        goto start
PSECT  code
start:
        movf    DisplayValue,W
        pagesel SevenSegmentDecode
        call    SevenSegmentDecode
        movwf   PORTB
        goto    Continue
SevenSegmentDecode:
        brw
        retlw   0b00111111 ;decode 0
        retlw   0b00000110 ;decode 1
        retlw   0b01011011 ;decode 2
        retlw   0b01001111 ;decode 3
        retlw   0b01100110 ;decode 4
        retlw   0b01101101 ;decode 5
        retlw   0b01111101 ;decode 6
        retlw   0b00000111 ;decode 7
        retlw   0b01111111 ;decode 8
        retlw   0b01101111 ;decode 9
Continue:
```

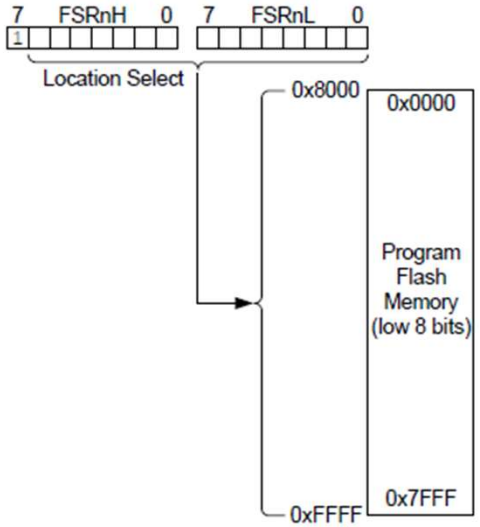
查表功能的實現範例



相對定址(Relative Addressing): with FSR

- 使用FSR讀取程式記憶體的方式與讀取資料記憶體類似
- FSR的最高位元設為1 (FSRxH,7),才會讀取到程式記憶體
- 此種方式只可以讀取程式記憶體的資料,不可以寫資料進程式記憶體.

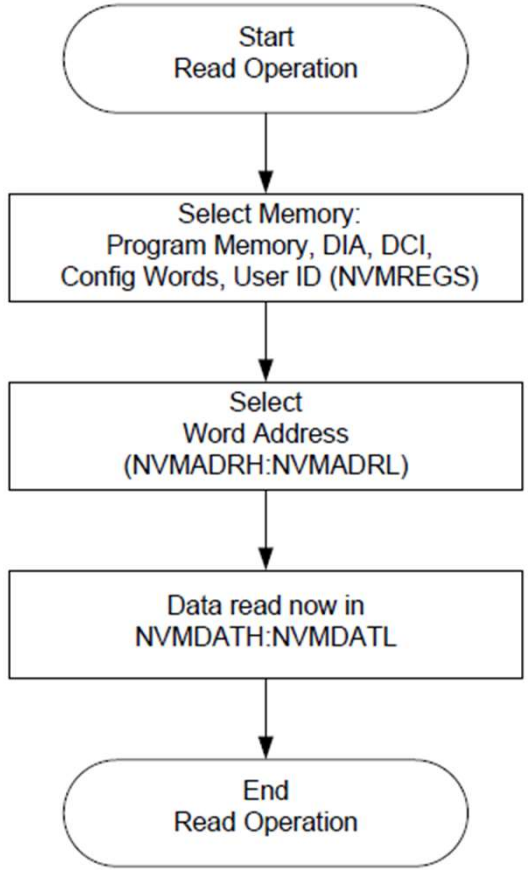
```
constants:
     RETLW DATA0 ;Index0 data
     RETLW DATA1 ;Index1 data
     RETLW DATA2
     RETLW DATA3
my_function:
     MOVLW LOW constants
     MOVWF FSR1L
     MOVLW HIGH (constants|0x8000)
     MOVWF FSR1H
     MOVIW 2[FSR1] ;DATA2 IS IN W
```



相對定址(Relative Addressing): with NVMREG

- 以此方法可讀&寫user ID, EEPROM & 程式記憶體
- 因寫的程序較為複雜,今天只介紹讀的程序
- 共有三個register要正確設定到才能正常完成讀取資料的動作, NVMADRL & H, NVMCON1,然後資料會被放在NVMDATH & L.

```
BANKSEL NVMADRL ; Select Bank for NVMCON registers
MOVLW low constants ;
MOVWF NVMADRL ; Store LSB of address
MOVLW high constants ;
MOVWF NVMADRH ; Store MSB of address
BCF     NVMREGS ; Do not select Configuration Space
BSF     RD ; Initiate read

MOVF    NVMDATL,W ; Get LSB of word
MOVWF PROG_DATA_LO ; Store in user location
MOVF    NVMDATH,W ; Get MSB of word
MOVWF PROG_DATA_HI ; Store in user location
```



MICROCHIP

組合語言的正確撰寫格式

MICROCHIP

正確的語法表達 (一)
數值、數字(字元)表示法

- 十進制表示**(Decimal):**
  - MOVLW 100 ; 載入常數 $100_{(10)}$ to W Reg.
- 十六進制表示**(Hexadecimal): <十六進制數目>H, <十六進制數目> h, 0x<十六進制數目>**
  - MOVLW 3FH ; 載入常數 $3F_{(16)}$ to W Reg.
  - MOVLW FEh ; 載入常數 $FE_{(16)}$ to W Reg.
  - MOVLW 0x3F ; 載入常數 $3F_{(16)}$ to W Reg.

正確的語法表達 (二)
數值、數字(字元)表示法

- 二進制表示**(Binary): <二進制數目>B, 0b <二進制數目>**
  - MOVLW 11110000B ; 載入常數 0xF0 to W Reg.
  - MOVLW 0b11110000 ; 載入常數 0xF0 to W Reg.

- 八進制表示**(Octal): <八進制數目>O, <八進制數目>Q, <八進制數目>o, <八進制數目>q**
  - MOVLW 200O ; 載入常數 128(10) to W Reg.
  - MOVLW 200o ; 載入常數 128(10) to W Reg.
  - MOVLW 200Q ; 載入常數 128(10) to W Reg.
  - MOVLW 200q ; 載入常數 128(10) to W Reg.

正確的語法表達 (三)
數值、數字(字元)表示法

- 字元**(ASCII):** ’ **<字元>** ‘
  - MOVLW ‘c’ ; 載入字元 “ c” to W Reg.
  - Const1 EQU ‘a’ ; Const1 = 小寫 的字元 A

- 標籤(Label): <label> :
  - loop1:
  - loop2: MOVLW 0x20 ;載入0x20 to W reg

必需要使用的虛擬指令

- PROCESS – 定義實際使用的Chip
  processor 16f18446
- #INCLUDE - 加入一原始檔、定義檔或敘述檔
  #include <xc.inc>
- EQU - 宣告常數
  memory equ 0x3f
- ds, db & dw - 宣告變數(可重新定位)
  - PSECT edata : 宣告位於EEPROM的變數
  - PSECT code
    - Bytes: DB 0x55 ; 宣告變數在flash memory, 且為byte(8bit)格式
    - Word: DW 0x55aa ;為word(16bit)格式
    - DoubleW: DDW 0x55aaaa55; 為double word(32bit)格式
  - PSECT udata : 由compiler決定變數存放位址
  - PSECT udata_shr : 指定變數存放於common RAM
  - PSECT udata_bank0 : 指定變數放於bank0
    - var1: ds 1 ; for 1 byte
    - Var2: ds 2 ; for 2 bytes
- 設定程式組譯的起始位址
  - Properies>pic-as Linker>Custom linker options “-presetVec=oh”
  - PSECT resetVec, class=CODE,delta=2 ; 組譯PIC16位址從 “00h”開始
  - PSECT resetVec, class=CODE, reloc=2 ; 組譯PIC18位址從 “00h”開始
  - 雖然可以使用org定址方式,但不建議使用
- **END - 程式結束**

虛擬指令的使用範例

Help→Online Help Contents→XC8 Toolchain→MPLAB XC8 PIC Assembler→About MPLAB XC8 PIC Assembler User's Guide for Embedded Engineers→ 4. Basic Example For Mid-range Devices

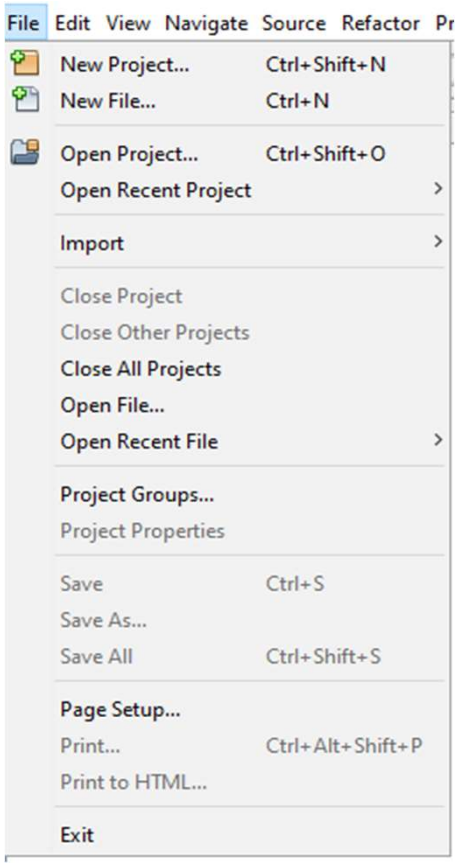
```
PROCESSOR 16F18446

#include <xc.inc>

CONFIG "FEXTOSC = OFF"

skipnc MACRO
    btfsc CARRY
ENDM

PSECT udata_bank0
max: DS 1 ;reserve 1 byte for max
tmp: DS 1 ;reserve 1 byte for tmp

PSECT resetVec,class=CODE,delta=2
resetVec:
      PAGESEL main ;jump to the main
       goto main

PSECT code
main:
;set up the oscillator
     movlw 0x62
     movlb 17
     movwf OSCCON1
     PAGESEL loop ;
     BANKSEL max ;starting point
     loop:
     BANKSEL PORTC ;read and store port
value
     movf BANKMASK(PORTC),w
     skipnc
     goto loop ;no - read again
     END resetVec
```

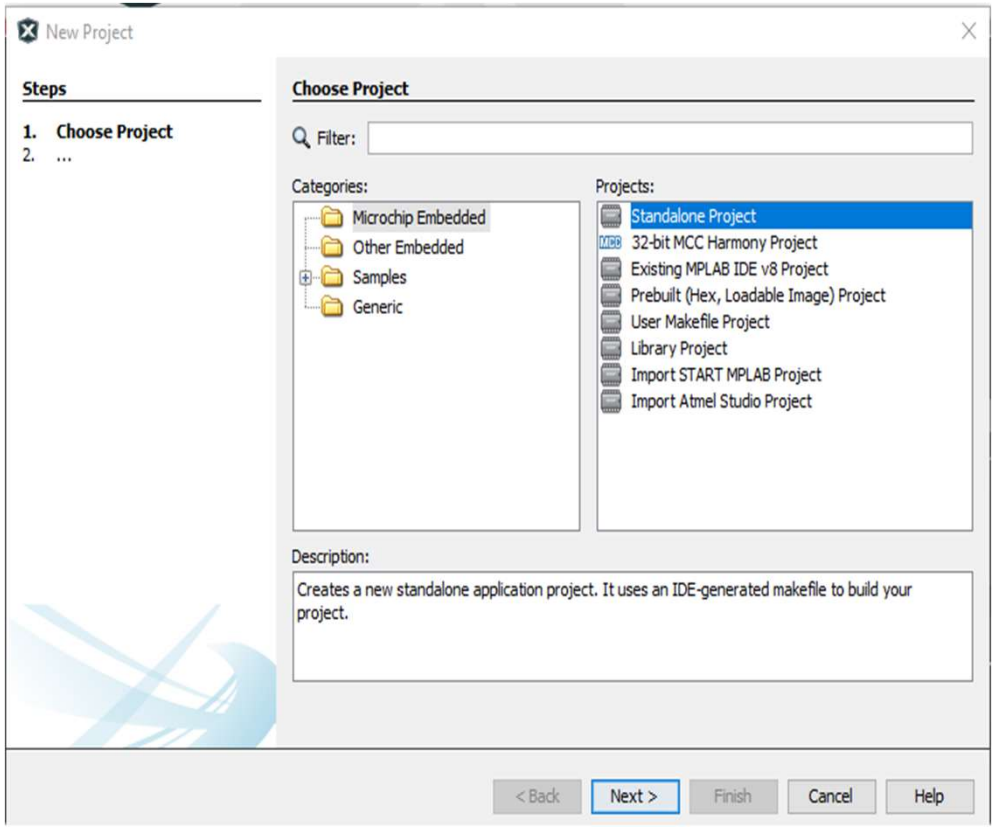
開啟專案&基礎的程式寫作

MICROCHIP

建立新專案(lab1)

- 實驗目標：學習建立新project &讀取不同記憶體的各種方式的比較
- Files→New Project
- Microchip Embedded→ Standalone Project
- Next

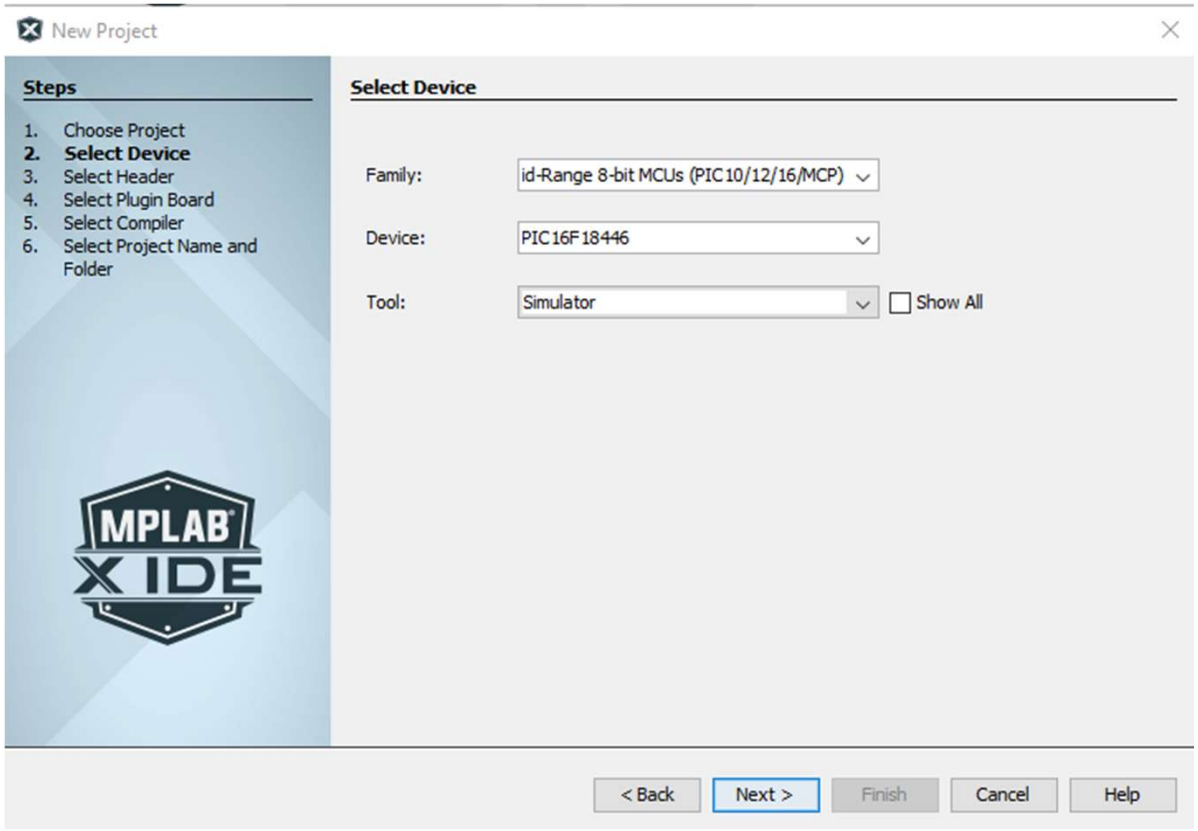




https://youtu.be/arkPJG3z55Q

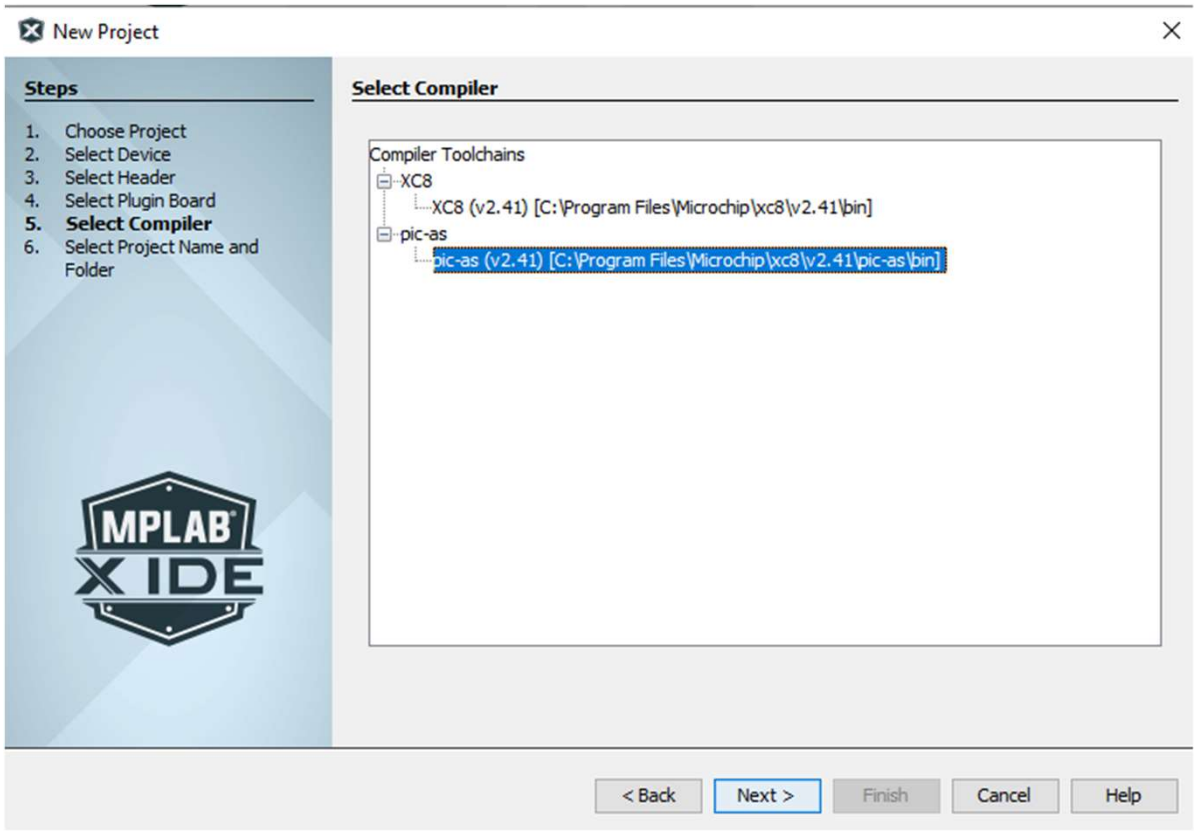
建立新專案(lab1)

- Family : Mid-Range 8-bit MCUs
- Device : PIC16F18446
- Tool : Simulator
- Next



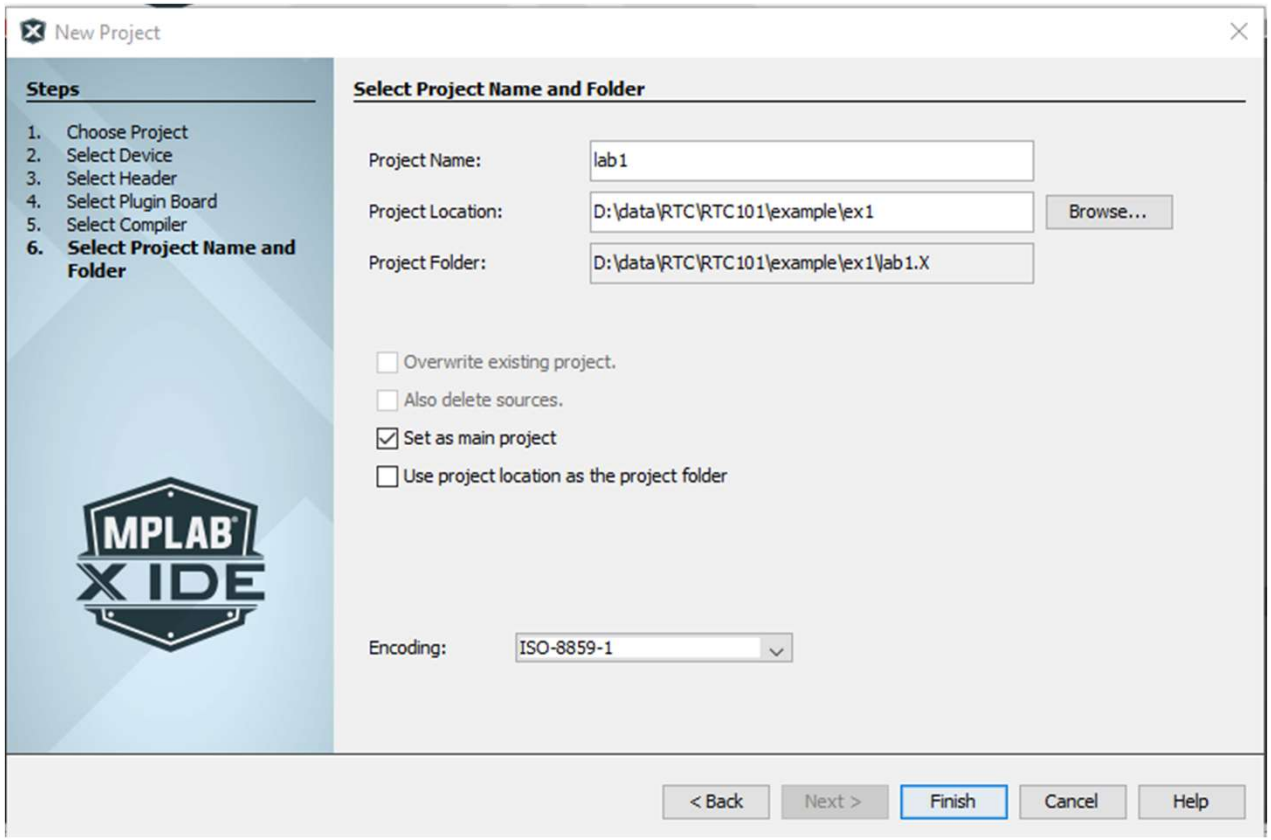
建立新專案(lab1)

- pic-as(v2.41)
- Next



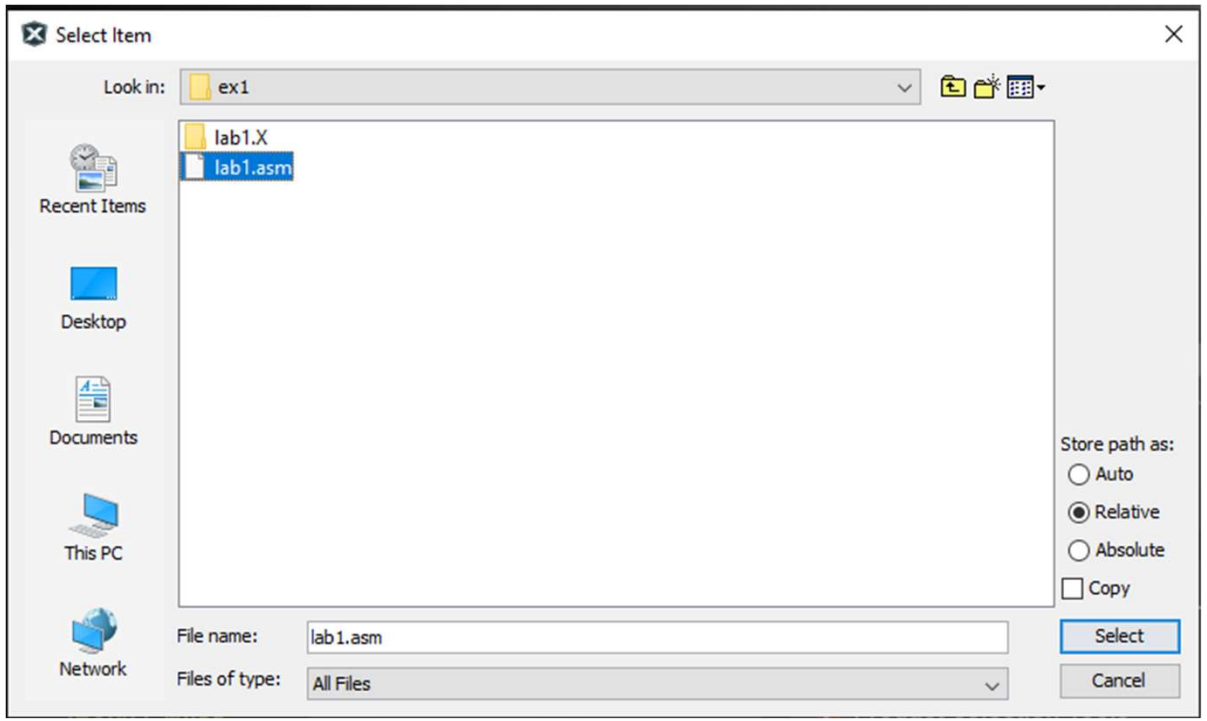
建立新專案(lab1)

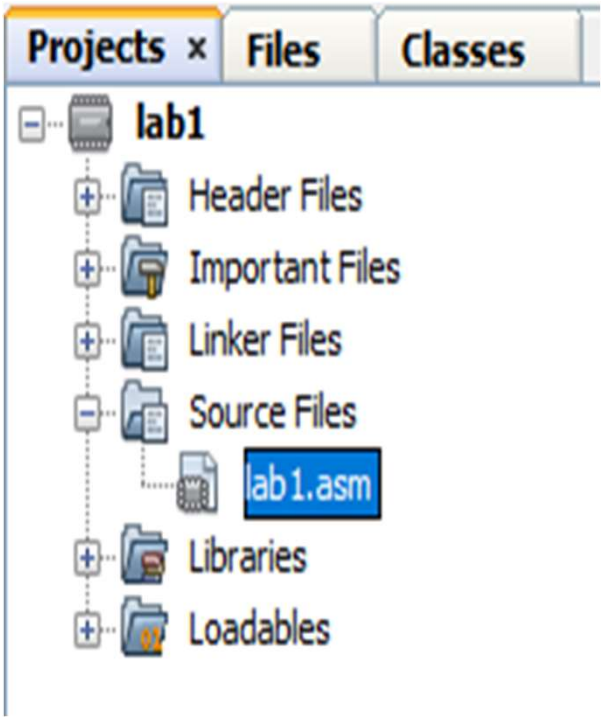
- Project Name : lab1
- Project Location : C:\RTC101\example\ex1
- Project Folder: C:\RTC101\example\ex1\lab1.X
- 勾選 Set as main project
- Finish
- 到此專案已經建立好了



加入source code到專案(一)(lab1)

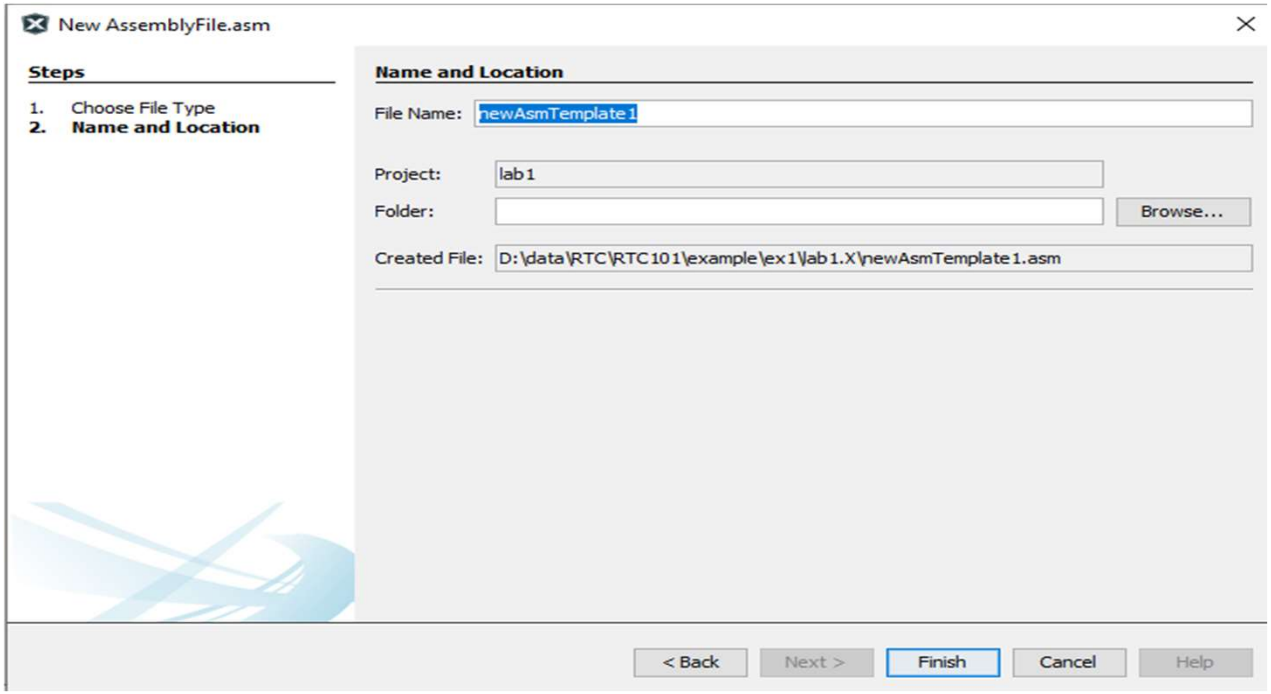
- 加入已經存在的檔案：Source Files→滑鼠右鍵→Add Existing Item→選擇存放檔案的目錄→選擇檔案

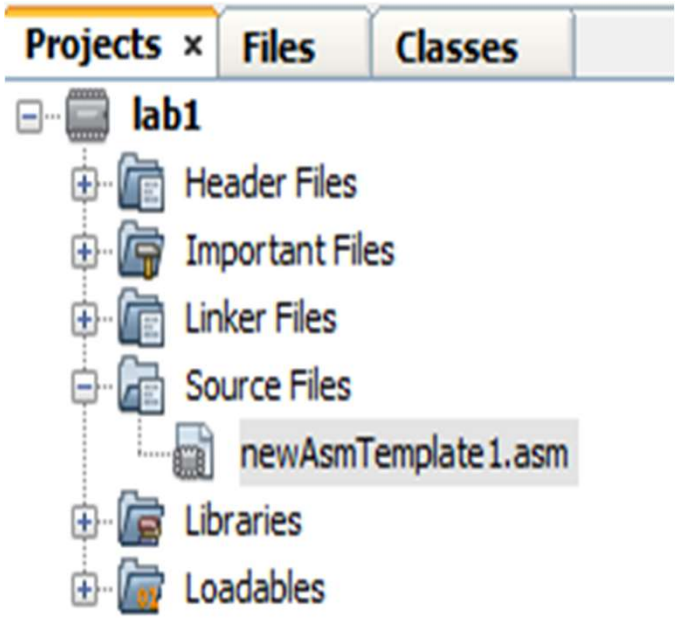




加入source code到專案(二) (lab1)

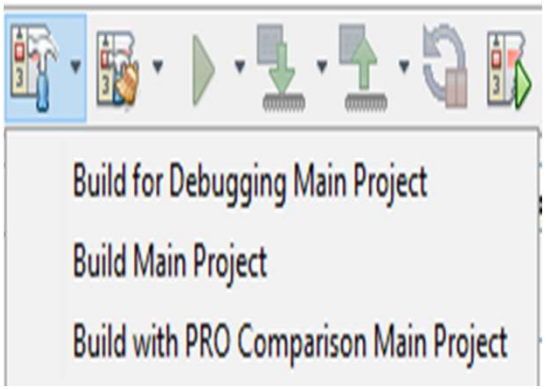
- 加入新檔案 : **Source Files→滑鼠右鍵→New**
  - →選擇AssemblyFile.asm→輸入 File Name:, 選擇目錄→ Finish
  - →選擇Other→Assembler→選擇AssemblyFile.asm or AssemblyFile.s均可→Finish

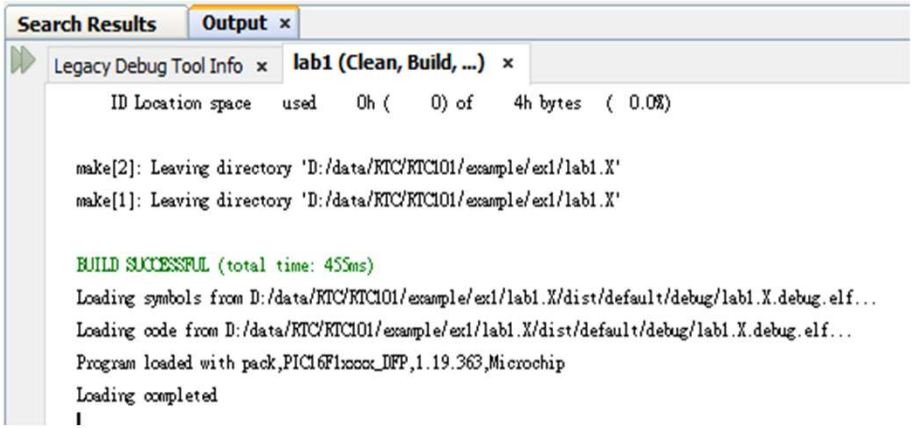




打開source code (lab1)

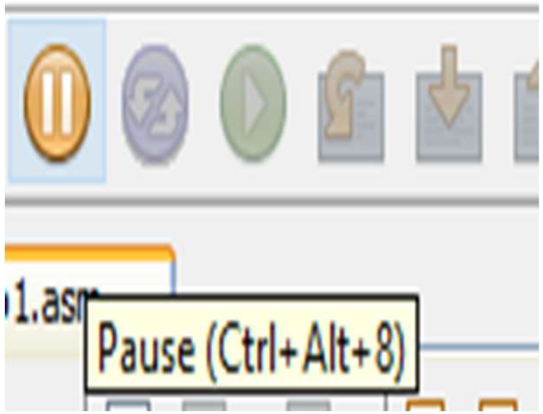
- **Source Files→**點選檔案
- 編輯你的檔案
- 編譯程式
  - Build
  - Clear and Build
- 編譯後的結果顯示在下方的**Output**視窗

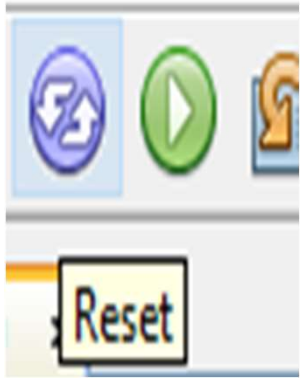




開始除錯(lab1)

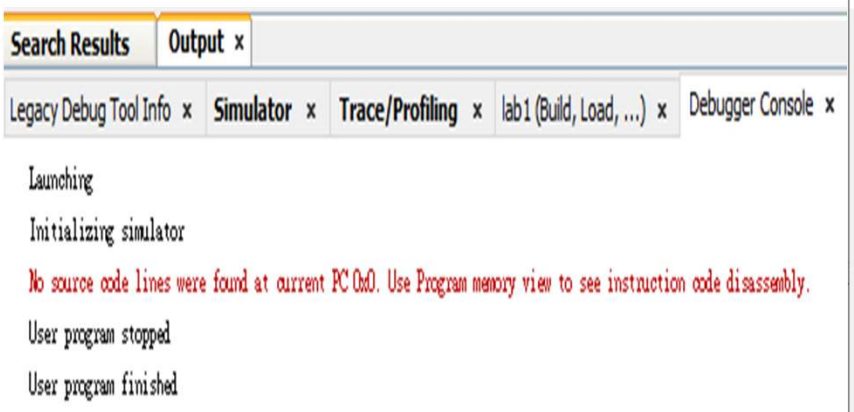
- 點選除錯
- 點選完後程式就開始一直跑,要停止下來,請按Pause
- 讓程式回到起始點,請按Reset
  - 另外一種選擇 : File →Project Properties →Simulator → Debug Options → Debug startup → Halt at Reset Vector, Debug reset → ResetVector→ Apply → OK

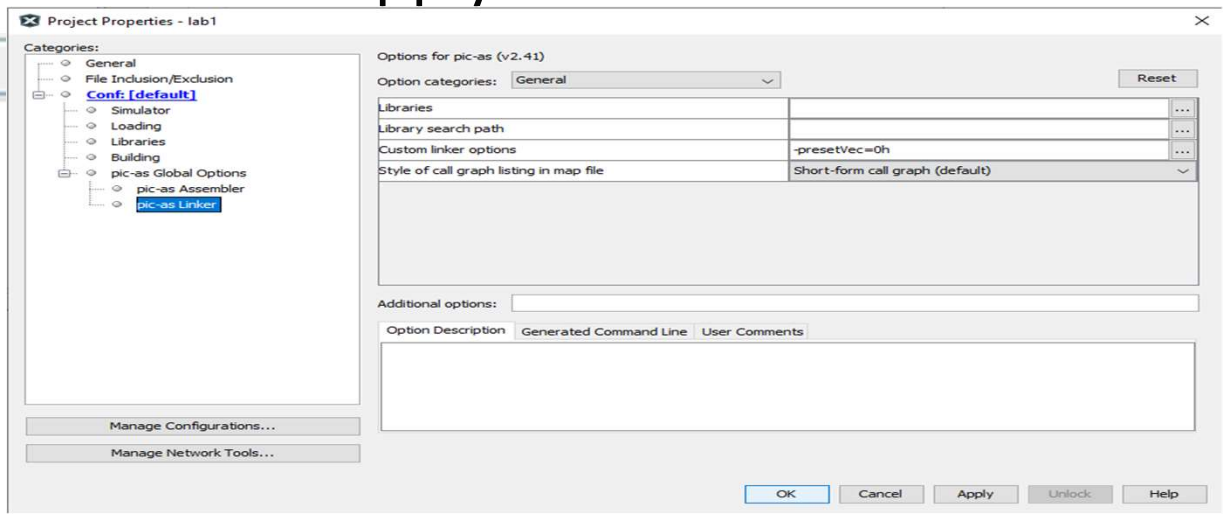




開始除錯(lab1)

- 這時Output Window顯示如下錯誤訊息,沒有source code存放到PC 0x0,意即程式沒有放在reset vector
- 前面有提到要在linker填寫 “-presetVec=oh”, File →Project Properties →pic-as Linker → General → Custom linker options → ResetVector→填寫-presetVec=oh → Apply → OK





開始除錯(lab1)

- 此程式的第一部分是清除0x20 ~ 0x7F RAM為0, 要如何看程式執行的結果是否正確?
- 在line 72點一下,設中斷點於此
- Window → Target Memory Views → File Registers
- 在line 81點一下,再次設中斷點,程式的第二部分是設0x20 ~ 0x7F RAM為0xFF. 有修改過內容的file register會被以紅色顯示

```
60   PSECT   code
61   start:
62   ;clear data memory with trandition
63           clrf    FSR0H
64           movlw   0x20
65           movwf   FSR0L
66   loop:   clrf    INDF0
67           incf    FSR0L,F
68           btfss   FSR0L,7
69           goto    loop
70
71   ;set data memory to 0xff with FSR, to compare with trandition
           clrf    FSR0H
73         movlw   0x20
```

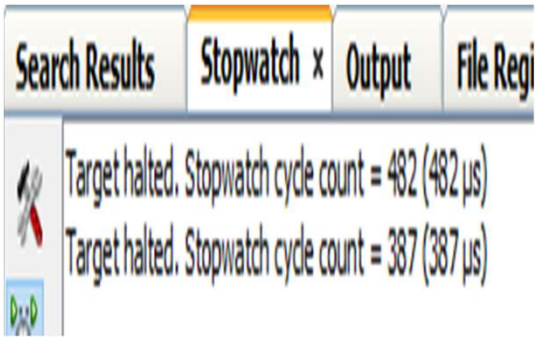
Search Results | Variables | Call Stack | Breakpoints | Output | File Registers ×

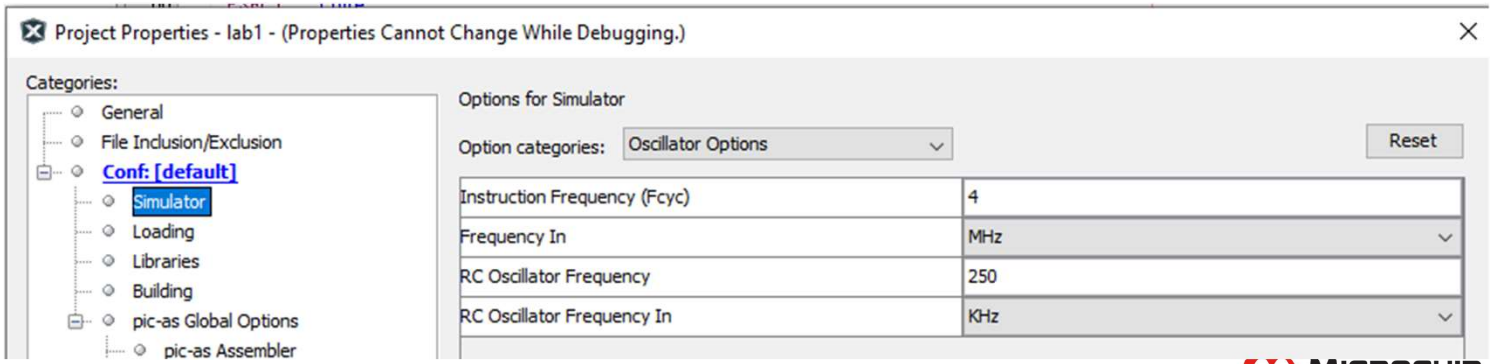
Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0010	--	--	3F	F0	FF	--	--	--	00	00	00	--	--	--	--	--	--?..--- ...-----
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0080	00	00	CE	18	80	00	00	00	00	20	07	01	00	00	00	00	....

Memory File Registers Format Hex

開始除錯(lab1)

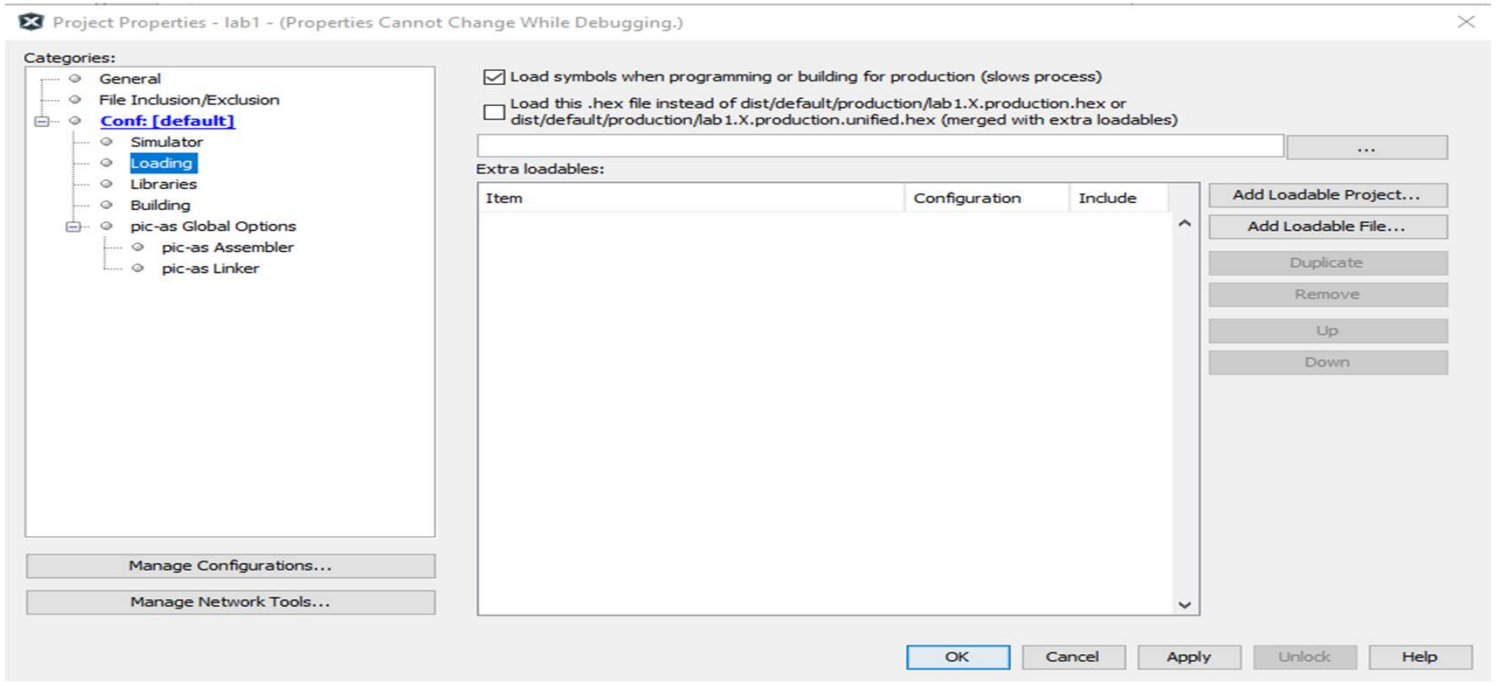
- 此程式的第一部分&第二部分基本上是相同的,差別只在於清為0 & 設為0xFF, &執行方式的不同,此時我們可以比較兩種方式的效率,分別程式記憶體所需空間 & 執行時間
- 兩者均占程式記憶體7個word
- Window → Debugging → Stopwatch
- 從下圖可看出,第二種方式比較節省時間
- File →Project Properties →Simulator → Oscilator Options → Instruction Frequency(Fcyc) → 4 → Apply → OK





開始除錯(lab1)

- 若程式是以C語言來編寫,如何檢查compiler編譯出來的結果?
- File →Project Properties →Loading → 勾選 Load symbols when programming or building for production → Apply → OK



開始除錯(lab1)

- Window → Debugging → Output → Disassembly Listing File

Start Page | lab1.asm | listing.disasm

```
                                                    50:         ds  1
                                                    51:
                                                    52:     ;define varible to bank0
                                                    53:     PSECT udata_bank0
                                                    54:
                                                    55:     PSECT resetVec, class=CODE,delta=2
                                                    56:     resetVec:
0000  3187      MOVLP 0x7                             57:         pagesel start
0001  2FC7      GOTO 0x7C7                            58:         goto start
                                                    59:
                                                    60:     PSECT  code
                                                    61:     start:
                                                    62:     ;clear data memory with trandition
07C7  0185      CLRF FSR0H                            63:         clrf    FSR0H
07C8  3020      MOVLW 0x20                            64:         movlw   0x20
07C9  0084      MOVWF FSR0                            65:         movwf   FSR0L
07CA  0180      CLRF INDF0                            66:     loop:  clrf    INDF0
07CB  0A84      INCF FSR0, F                          67:         incf    FSR0L,F
07CC  1F84      BTFSS FSR0, 0x7                       68:         btfss   FSR0L,7
07CD  2FCA      GOTO 0x7CA                            69:         goto    loop
                                                    70:
                                                    71:     ;set data memory to 0xff with FSR, to compare with trandition
07CE  0185      CLRF FSR0H                            72:         clrf    FSR0H
07CF  3020      MOVLW 0x20                            73:         movlw   0x20
07D0  0084      MOVWF FSR0                            74:         movwf   FSR0L
07D1  30FF      MOVLW 0xFF                            75:         movlw   0xff
07D2  001A      MOVWI FSR0++                          76:     loopa: movwi   FSR0++
07D3  1F84      BTFSS FSR0, 0x7                       77:         btfss   FSR0L,7
07D4  2FD2      GOTO 0x7D2                            78:         goto    loopa
                                                    79:
```

lab2

- 實驗目標 : 與lab1比較 Assembler & C語言的效率
- 此程式是以MPLAB的MCC所產生的C程式,此程式期望將0x20 ~ 0x7F RAM設為0xFF,結果?
- 請練習前一個實驗所使用的各種debug技巧, 包括中斷, File Registers, Stopwatch, Oscilator Options, Disassembly Listing File…

- RAM位址不是在所期望的位址
- 執行的時間比用assembler所寫的程式要長很多
- 程式記憶體所需空間也是多很多

lab3

- 實驗目標 : 實際寫一個Assembly語言,練習用軟體做一個時間延遲
- 首先依lab1的步驟建立new project, project name : lab3
- 分別在”To Do”寫下程式
- 用lab1所學習到的debug技巧, 觀看delay loop的執行時間
- 嘗試修改延遲時間

```
call      delay_1ms
movlw  VAL_US          ; 1us
movwf count     ; 1us
```

https://youtu.be/0xSw_xIALu8

配置字(Configuration word)

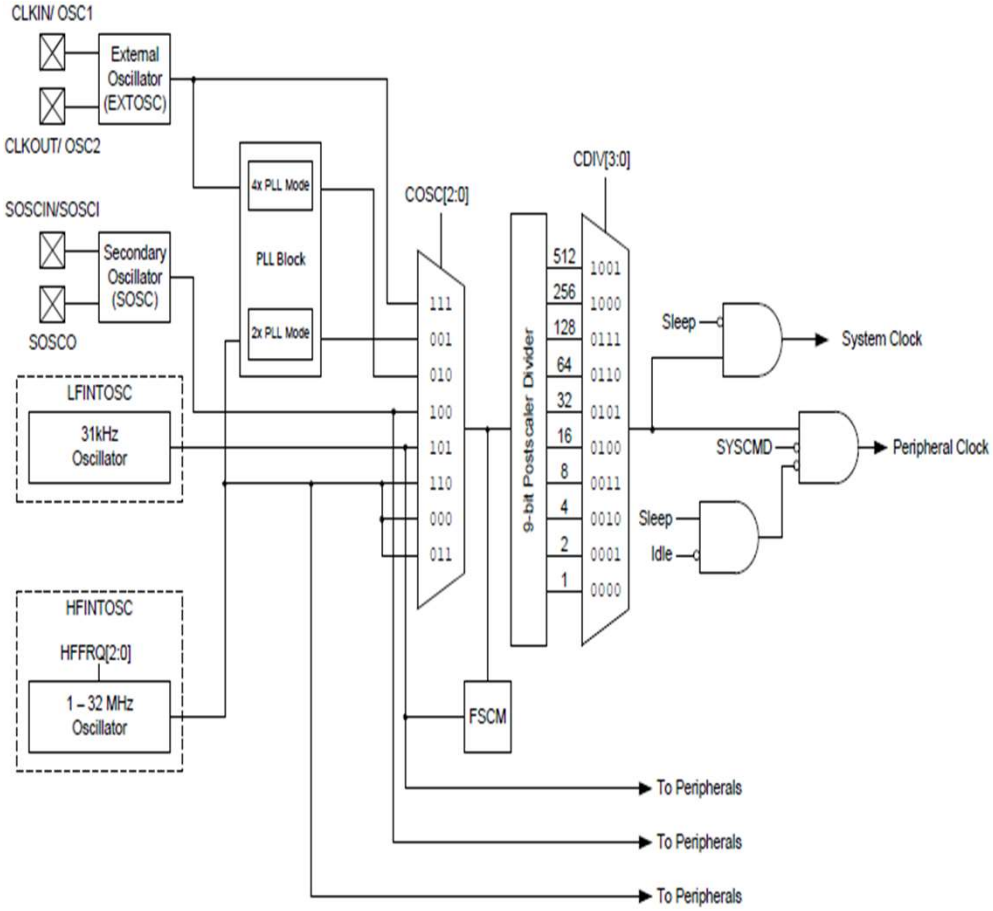
配置字 (Configuration word)

- 位於程式記憶體空間,但程式計數器(PC counter)無法指至此空間, 所以程式不會放在此空間
- 此空間是放置chip的一些設定的選項
  - Code Protection
  - Oscillator Mode
  - Watchdog Timer
  - Power Up Timer
  - Brown Out Reset
  - Low Voltage Programming
  - Flash Program Memory Write…

Offset	Name	Bit Pos.								
0x8007	CONFIG1	7:0		RSTOSC[2:0]				FEXTOSC[2:0]		
		13:8			FCMEN		CSWEN			CLKOUTEN
0x8008	CONFIG2	7:0	BOREN		LPBOREN			PWRTS[1:0]		MCLRE
		13:8			DEBUG	STVREN	PPS1WAY	ZCDDIS	BORV	
0x8009	CONFIG3	7:0		WDTE[1:0]		WDTCPS[4:0]				
		13:8			WDTCCS[2:0]			WDTCWS[2:0]		
0x800A	CONFIG4	7:0	WRTAPP			SAFEN	BBEN	BBSIZE[2:0]		
		13:8			LVP		WRTSAF	WRTD	WRTC	WRTB
0x800B	CONFIG5	7:0								CP
		13:8								

晶振的選擇--方塊圖

- **HFINTOSC : 1 ~ 32MHz**
  - 2xPLL
- **LFINTOSC : 31KHz**
- **EC**
- **LP : 32.768Khz, 耗電量最低**
- **XT : ~4MHz**
- **HS : ~20MHz, 放大倍率最高, 意謂最耗電**
- **Second Oscillator : 32.768KHz**



晶振相關的配置字

Name: CONFIG1
Offset: 0x8007
Configuration Word 1
Oscillators

Bit	15	14	13	12	11	10	9	8
			FCMEN		CSWEN			CLKOUTEN
Access			R/P	U	R/P	U	U	R/P
Reset			1	1	1	1	1	1

Bit	7	6	5	4	3	2	1	0
		RSTOSC[2:0]				FEXTOSC[2:0]		
Access	U	R/P	R/P	R/P	U	R/P	R/P	R/P
Reset	1	1	1	1	1	1	1	1

Bit 13 – FCMEN Fail-Safe Clock Monitor Enable bit

Value	Description
1	FSCM timer enabled
0	FSCM timer disabled

Bit 11 – CSWEN Clock Switch Enable bit

Value	Description
1	Writing to NOSC and NDIV is allowed
0	The NOSC and NDIV bits cannot be changed by user software

Bits 6:4 – RSTOSC[2:0] Power-up Default Value for COSC bits

This value is the Reset default value for COSC and selects the oscillator first used by user software. Refer to COSC operation.

Value	Description
111	EXTOSC operating per FEXTOSC bits
110	HFINTOSC (1 MHz), with OSCFRQ = '010' (4 MHz) and CDIV = '0010' (4:1)
101	LFINTOSC
100	SOSC
011	Reserved
010	EXTOSC with 4x PLL, with EXTOSC operating per FEXTOSC bits
001	HFINTOSC with 2x PLL (32 MHz), with OSCFRQ = '101' (16 MHz) and CDIV = '0000' (1:1)
000	HFINTOSC with OSCFRQ = 32 MHz and CDIV = 1:1

Bits 2:0 – FEXTOSC[2:0] FEXTOSC External Oscillator Mode Selection bits

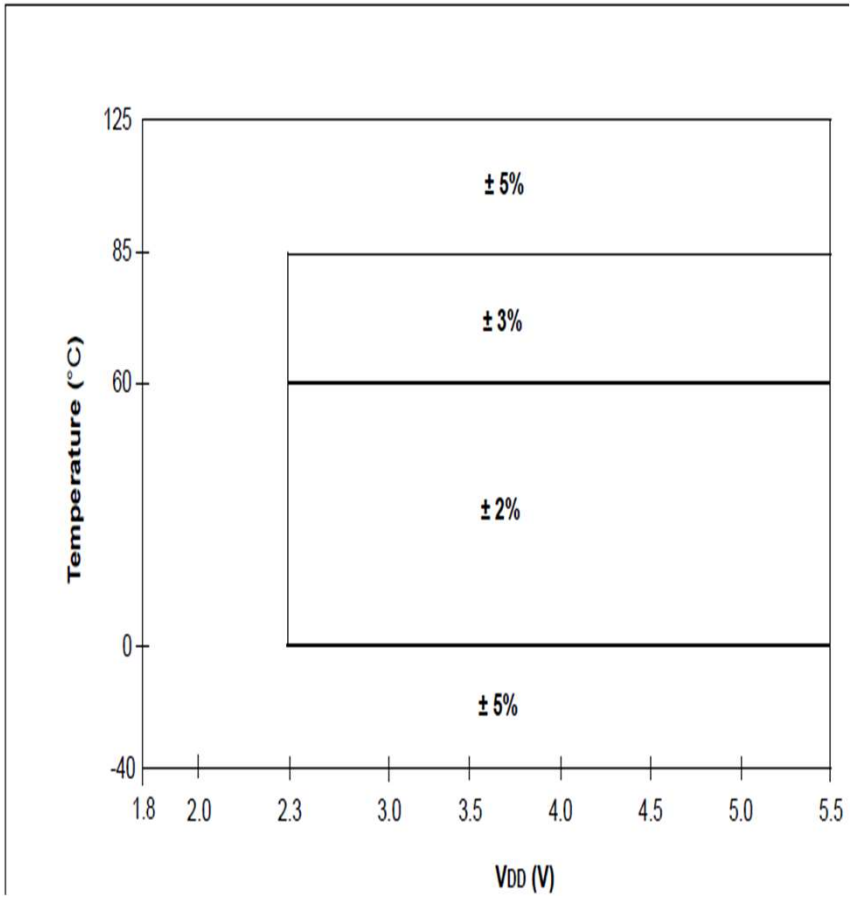
Value	Description
111	ECH (External Clock) above 8 MHz
110	ECM (External Clock) for 500 kHz to 8 MHz
101	ECL (External Clock) below 500 kHz
100	Oscillator not enabled
011	Reserved (do not use)
010	HS (Crystal oscillator) above 4 MHz
001	XT (Crystal oscillator) above 100 kHz, below 4 MHz
000	LP (Crystal oscillator) optimized for 32.768 kHz

內部晶振的誤差量

OS51	$F_{HFOSCLP}$	Low-Power Optimized HFINTOSC Frequency	0.92	1	1.08	MHz	-40°C to 85°C
			1.84	2	2.16	MHz	-40°C to 85°C
			0.88	1	1.12	MHz	-40°C to 125°C
			1.76	2	2.24	MHz	-40°C to 125°C

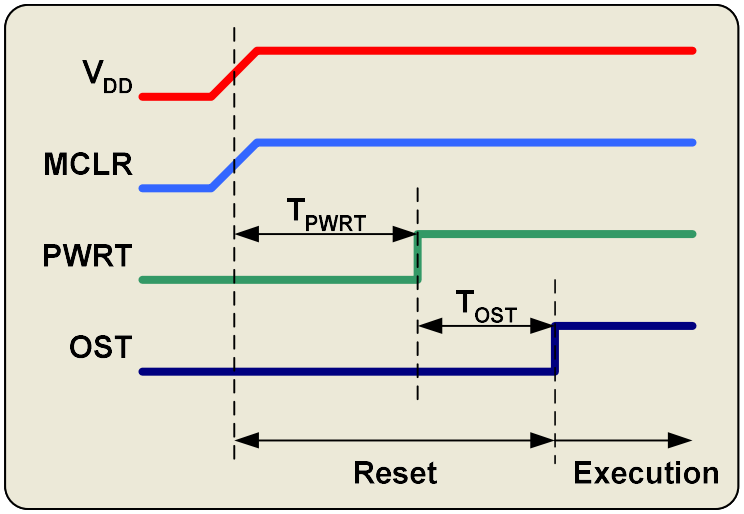
- 內部晶振是最低成本的晶振方式
- 且此方式也是最容易使**MCU**可以正常**&**穩定工作
- 考慮工作環境溫度**&**工作電壓,內部晶振的誤差量是否可以接受
- 若有使用通訊界面,如**UART, USB…**,誤差量是否在可接受範圍之內.若超出可接受範圍,將造成通訊品質低下

Figure 39-6. Precision Calibrated HFINTOSC Frequency Accuracy Over Device V_{DD} and Temperature



POR, OST,PWRT

- **POR : Power On Reset**
  - 若MCLR直接接至Vdd,當電壓升至一定的準位後,會產生一個reset脈衝
- **PWRT: Power-up Timer**
  - POR之後,PWRT持續將CPU keep在reset狀態,直至timer溢出
- **OST : Oscillator Start-up Timer**
  - 若振盪器是選擇外部的晶振,則會持續hold CPU於reset狀態,直至收到穩定的頻率&振幅1024個週期. 若選擇RC模式,則此功能自動disable



Value	Description
11	PWRT disabled
10	PWRT set at 64 ms
01	PWRT set at 16 ms
00	PWRT set at 1 ms

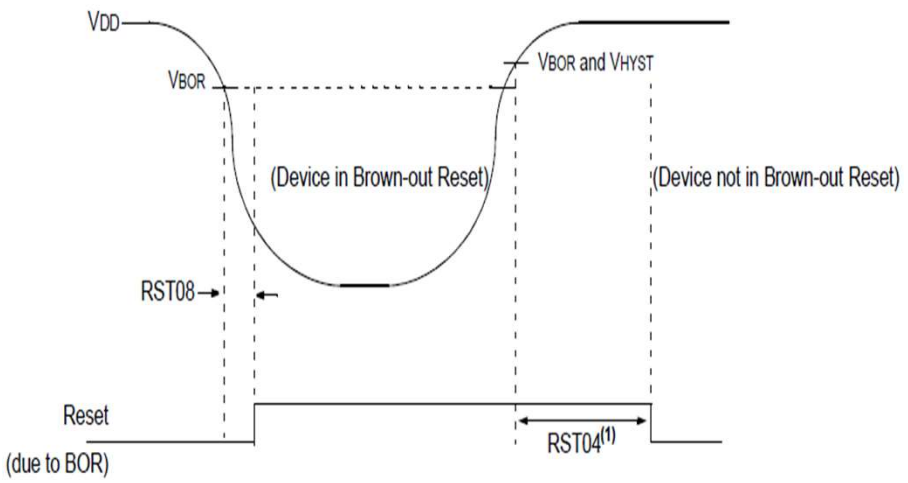
看門狗(Watchdog Timer)

- 當軟體進入無窮迴圈時,幫助MCU從軟體故障恢復.
- WDT有自己的free running RC振盪器,故MCU進入Sleep mode時,WDT仍然會繼續計時
- 當WDT溢位時會重置 MCU
- Time out週期可規劃1ms ~ 256s @31KHz
- 以CLRWDT指令清除WDT計數器
- 在Sleep mode, time out時會將CPU喚醒
- 可以在軟體執行中, 隨時enable or disable WDT
- 在debug階段請disable WDT,以免WDT time out產生的reset干擾debug的進行

(P)BOR – (Programmable) Brown Out Reset

- 當Vdd電壓下降到一定準位時,MCU就會被保持在reset狀態
- 此功能是為了防止電壓不穩定時,MCU執行不可預期的工作
- 消除外部BOR的線路
- 可選擇臨界電壓

Figure 39-9. Brown-out Reset Timing and Characteristics



V_{BOR}	Brown-out Reset Voltage	2.55 2.30 1.80	2.7 2.45 1.90	2.85 2.60(3) 2.05	V V V	BORV = 0 BORV = 1(F devices only) BORV = 1(LF Devices only)
V_{LPBOR}	Low-Power Brown-out Reset Voltage	1.8	1.9	2.2	V	LF Devices only
V_{BORHYS}	Brown-out Reset Hysteresis	–	40	–	mV	

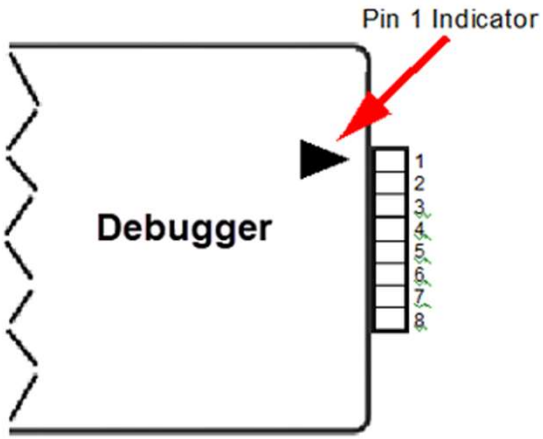
休眠模式(Sleep mode)

- 執行**SLEEP**指令就可以使**CPU**進入休眠模式
  - 系統振盪器停止運作
  - CPU狀態保持不變
  - 若有致能WDT,在此模式下,WDT保持運作
  - 最小的供應電流下降,大部分都是漏電流所造成(typ : 0.1 ~ 0.2uA)

Events that wake processor from sleep	
MCLR	Master Clear Pin Asserted (pulled low)
WDT	Watchdog Timer Timeout
INT	INT Pin Interrupt
TMR1	Timer 1 Interrupt (or also TMR3 on PIC18)
ADC	A/D Conversion Complete Interrupt
CMP	Comparator Output Change Interrupt
CCP	Input Capture Event
PORTB	PORTB Interrupt on Change
SSP	Synchronous Serial Port (I^2C Mode) Start / Stop Bit Detect Interrupt

ICSP(In-Circuit Serial Programming)

- 對PIC來說,若要燒錄chip,則只需要連接2 pins
- 方便用於系統燒錄：chip已經焊接在PCB時,要燒錄程式,或燒錄工廠生產的校正值.



Debugger Pin Descriptions

1 = $\overline{\text{MCLR}}$
2 = V_{DD}
3 = Ground
4 = PGD
5 = PGC
6 = Do not connect
7 = Unused for ICSP
8 = Unused for ICSP

	MPLAB Snap		DEBUG								
Connector	Pin #	Pin Name	ICSP (MCHP)	MIPS EJTAG	CORTEX® SWD	AVR® JTAG	AVR ISP(&DW)	UPDI	PDI	debugWIRE	TPI
	1	TVPP	$\overline{\text{MCLR}}$	$\overline{\text{MCLR}}$	$\overline{\text{MCLR}}$						
	2	TVDD	VDD	VIO_REF	VTG	VTG	VTG	VTG	VTG	VTG	VTG
	3	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND
	4	PGD	DAT	TDO	SWO	TDO	MISO	DAT	DAT		DAT
	5	PGC	CLK	TCK	SWCLK	TCK	SCK				CLK
	6	TAUX	AUX			$\overline{\text{RESET}}$	$\overline{\text{RESET}}$		CLK	dW	RST
	7	TTDI		TDI		TDI	MOSI				
	8	TTMS		TMS	SWDIO	TMS					

Lab3

- 實驗目標 : 以軟體延遲1ms
- 請自行建立新project, Device : PIC16F18446, Tool : Simulator, Project Name : lab3, Project Location : C:\RTC101\example\ex3, Project Folder: C:\RTC101\example\ex3\lab3.X
- 請key-in以右邊程式
- Compile成功後,以Stopwatch觀看執行時間是否為期望值?1ms?
- 可以嘗試改為延遲0.5, 10 or 200mS

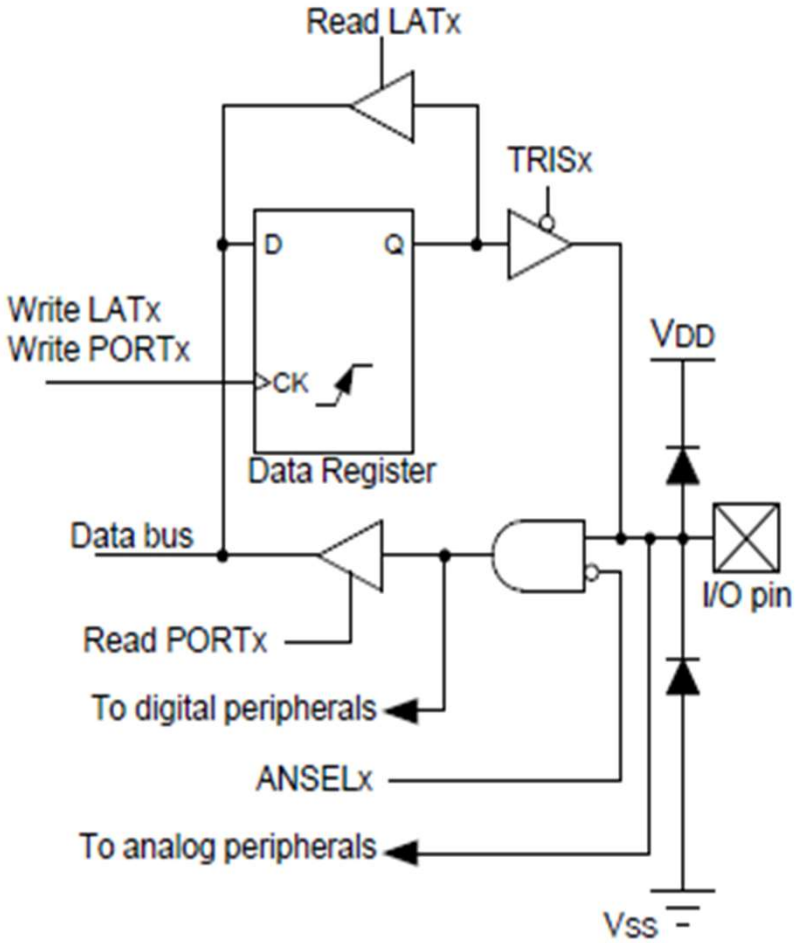
```
PSECT   code
start:
        call    delay_1ms       ; 2us
loop:   goto    loop

delay_1ms:
        movlw   VAL_US          ; 1us
        movwf   count           ; 1us

dec_loop:
        nop                     ; 1us
        decfsz  count,f         ; count=0, 2us
                                ; count>0, 1us
        goto    dec_loop        ; 2us
        return                  ; 2us

        end
```

MPLAB-SNAP 及多功能實驗板的操作

MICROCHIP

基本I/O操作

- 操作一個**I/O port**最基本相關**registers, PORTx, TRISx & LATx**
- 另外相關**registers:**
  - ANSELx : 類比選擇
  - WWPUx : 內部上拉電阻致能 or 禁用
  - INLVLx : 輸入電壓準位控制
  - SLRCONx :電壓準位上下速度控制
  - ODCONx : 漏極開路(open-drain)控制
- 若**I/O pin**有類比訊號功能,通常**chip power on reset**時的默認狀況都是致能類比功能,若要當成數位功能,需要將此類比功能禁用
- 硬體設計時要注意**I/O pin**可以承受**source or sink**最大電流量



設定I/O Pin

- TRISx為I/O輸入輸出控制暫存器
  - I/O要設為輸出時,相對應的位元設為”0”
  - I/O要設為輸入時,相對應的位元設為”1”
- 讀取PORTx是讀到pin的實際狀態.寫資料到PORTx實際上是先讀PORTx資料,然後修改資料後再寫到PORTx,也就是Read-modify-write, 另外也隱含資料同時也被寫至LATx
- LATx保持輸出資料&最後寫到LATx or PORTx資料
- ANSELx相對位元設為”1”是類比輸入, “0”是數位I/O pin
- 基本上要設定以上四個暫存器, I/O pin才會運作,另外四個暫存器是附加功能,善用可降低EMI or降低PCB整體成本

```
BANKSEL PORTB ;
CLRF PORTB          ;Init PORTB
BANKSEL LATB        ;Data Latch
CLRF LATB           ;
BANKSEL ANSELB      ;
CLRF ANSELB         ;digital I/O
BANKSEL TRISB ;
MOVLW 0b00111000    ;Set RB[5:3] as inputs
MOVWF TRISB         ;and set RB[2:0] as outputs
```

```
BANKSEL PORTB ;
CLRF PORTB          ;Init PORTB
CLRF LATB           ;
MOVLW 0b00111000    ;Set RB[5:3] as inputs
MOVWF TRISB         ;and set RB[2:0] as outputs
BANKSEL ANSELB      ;
CLRF ANSELB         ;digital I/O
```

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00 ... 0x0B	Reserved									
0x0C	PORTA	7:0			RA5	RA4	RA3	RA2	RA1	RA0
0x0D	PORTB	7:0	RB7	RB6	RB5	RB4				
0x0E	PORTC	7:0	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
0x0F ... 0x11	Reserved									
0x12	TRISA	7:0			TRISA5	TRISA4		TRISA2	TRISA1	TRISA0
0x13	TRISB	7:0	TRISB7	TRISB6	TRISB5	TRISB4				
0x14	TRISC	7:0	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
0x15 ... 0x17	Reserved									
0x18	LATA	7:0			LATA5	LATA4		LATA2	LATA1	LATA0
0x19	LATB	7:0	LATB7	LATB6	LATB5	LATB4				
0x1A	LATC	7:0	LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
0x1B ... 0x1F37	Reserved									
0x1F38	ANSELA	7:0			ANSELA5	ANSELA4		ANSELA2	ANSELA1	ANSELA0
0x1F39	WPUA	7:0			WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0
0x1F3A	ODCONA	7:0			ODCA5	ODCA4		ODCA2	ODCA1	ODCA0
0x1F3B	SLRCONA	7:0			SLRA5	SLRA4		SLRA2	SLRA1	SLRA0
0x1F3C	INLVLA	7:0			INLVLA5	INLVLA4	INLVLA3	INLVLA2	INLVLA1	INLVLA0
0x1F3D ... 0x1F42	Reserved									
0x1F43	ANSELB	7:0	ANSELB7	ANSELB6	ANSELB5	ANSELB4				
0x1F44	WPUB	7:0	WPUB7	WPUB6	WPUB5	WPUB4				
0x1F45	ODCONB	7:0	ODCB7	ODCB6	ODCB5	ODCB4				
0x1F46	SLRCONB	7:0	SLRB7	SLRB6	SLRB5	SLRB4				
0x1F47	INLVLB	7:0	INLVLB7	INLVLB6	INLVLB5	INLVLB4				
0x1F48 ... 0x1F4D	Reserved									
0x1F4E	ANSELC	7:0	ANSELC7	ANSELC6	ANSELC5	ANSELC4	ANSELC3	ANSELC2	ANSELC1	ANSELC0
0x1F4F	WPUC	7:0	WPUC7	WPUC6	WPUC5	WPUC4	WPUC3	WPUC2	WPUC1	WPUC0
0x1F50	ODCONC	7:0	ODCC7	ODCC6	ODCC5	ODCC4	ODCC3	ODCC2	ODCC1	ODCC0
0x1F51	SLRCONC	7:0	SLRC7	SLRC6	SLRC5	SLRC4	SLRC3	SLRC2	SLRC1	SLRC0
0x1F52	INLVLC	7:0	INLVLC7	INLVLC6	INLVLC5	INLVLC4	INLVLC3	INLVLC2	INLVLC1	INLVLC0

I/O Pin直接驅動LED

- **PIC16F18446基本電器規格**
  - Vdd pin最大電流：250mA
  - Vss pin最大電流：250mA
  - I/O pin最大電流：50mA

- **常見I/O Port名詞解釋**
  - V_{IL}
  - V_{IH}
  - V_{OL}
  - V_{OH}
  - Min
  - Typical
  - Max

Maximum current		
• on V_{SS} pin(1)	-40°C ≤ T_A ≤ +85°C	250 mA
	85°C < T_A ≤ +125°C	120 mA
• on V_{DD} pin(1)	-40°C ≤ T_A ≤ +85°C	250 mA
	85°C < T_A ≤ +125°C	85 mA
• on any standard I/O pin		±50 mA
Clamp current, I_K (V_{PIN} < 0 or V_{PIN} > V_{DD})		±20 mA
Total power dissipation(2)		800 mW

Param. No.	Sym.	Device Characteristics	Min.	Typ.†	Max.	Units	Conditions
Input Low Voltage							
	V_{IL}	I/O PORT:					
D300		• with TTL buffer	—	—	0.8	V	4.5V≤V_{DD}≤5.5V
D301			—	—	0.15 V_{DD}	V	1.8V≤V_{DD}≤4.5V
D302		• with Schmitt Trigger buffer	—	—	0.2 V_{DD}	V	2.0V≤V_{DD}≤5.5V
D303		• with I^2C levels	—	—	0.3 V_{DD}	V	
D304		• with SMBus levels	—	—	0.8	V	2.7V≤V_{DD}≤5.5V
D305		$\overline{MCLR}$	—	—	0.2 V_{DD}	V	
Input High Voltage							
	V_{IH}	I/O PORT:					
D320		• with TTL buffer	2.0	—	—	V	4.5V≤V_{DD}≤5.5V
D321			0.25 V_{DD}+0.8	—	—	V	1.8V≤V_{DD}≤4.5V
D322		• with Schmitt Trigger buffer	0.8V_{DD}	—	—	V	2.0V≤V_{DD}≤5.5V
D323		• with I^2C levels	0.7 V_{DD}	—	—	V	
D324		• with SMBus levels	2.1	—	—	V	2.7V≤V_{DD}≤5.5V
D325		$\overline{MCLR}$	0.7 V_{DD}	—	—	V	
D350	I_{PUR}		25	120	200	µA	V_{DD}=3.0V, V_{PIN}=V_{SS}
Output Low Voltage							
D360	V_{OL}	Standard I/O PORTS	—	—	0.6	V	I_{OL} = 10 mA, V_{DD} = 3.0V
Output High Voltage							
D370	V_{OH}	Standard I/O PORTS	V_{DD}-0.7	—	—	V	I_{OH} = 6 mA, V_{DD} = 3.0V

Lab4：基本I/O控制

- 實驗目標：學習使用硬體除錯器, 實驗板 & MCU I/O pin輸出訊號
- 利用Lab3的延遲副程式為基礎,做出一個延遲200ms副程式
- 每隔200ms點亮 or 熄滅LED1, LED1接到MCU的RA2 pin.
- 利用APP_All_MCU_2023實驗版 & SNAP進行除錯
- 因為要用到硬體除錯,請注意configuration word設定
- Crystallor使用內振32MHz,在經過除頻8,可得到系統頻率為4MHz. https://youtu.be/x0YNGA5vgvo

需要加程式至lab4

```
; configuration set OSC = 32MHz, has to divide 8, then OSC = 4MHz
        banksel OSCCON1
        movlw   0b01100011
        movwf   OSCCON1
; waiting OSC switch ready
wait:
        btfss   ORDY
        goto    wait


; set all port A pin as digital IO pin
; set all port B pin as digital IO pin
        clrf    ANSELA
        clrf    ANSELB
        movlw   0b00001000
        movwf   ANSELC


; enable or disable open-drain
        clrf    ODCONA
        movlw   0b01010000      ; RB6 & RB4 are I2C interface
        movwf   ODCONB
        clrf    ODCONC
```

```
; select I/O pin be input or output, 1=input, 0=output
        movlw   0b11001011
        movwf   TRISA
        movlw   0b01111111
        movwf   TRISB
        movlw   0b00011001
        movwf   TRISC
repeat:
        bsf     LATA2   ;
        call    delay_200ms ;
        bcf     LATA2   ;
        call    delay_200ms ;
        goto    repeat

delay_200ms:
        movlw   d_200
        movwf   c_200
loop_1:
        call    delay_1ms
        decfsz  c_200,f
        goto    loop_1
        return
```

Lab5：輸入訊號練習

- 實驗目標：學習MCU I/O pin偵測外部訊號
- 利用Lab4為基礎,偵測SW3按鍵的狀態,以改變LED ON-OFF時間, SW3連接至MCU RC0 pin
- 當按鍵沒有被押按時, LED ON時間為250ms, OFF時間為100ms
- 當按鍵被押按時, LED ON時間為100ms, OFF時間為250ms
- https://youtu.be/i6avLRb9lwQ

- 加碼練習：可任意修改LED ON or OFF時間 1 ~ 255ms

需要加程式至lab5

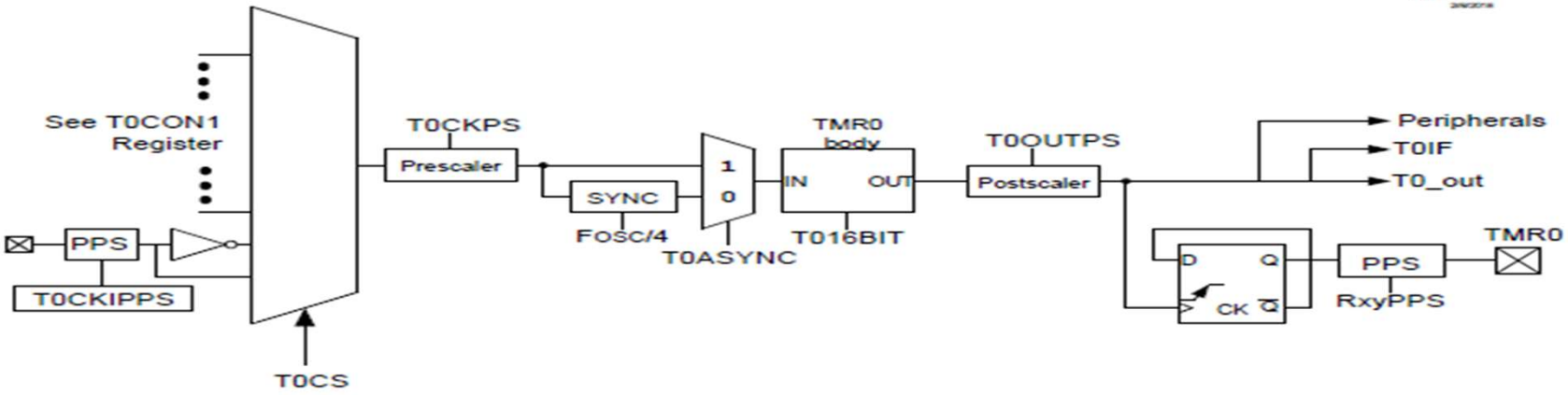
```
on_count      equ  100
off_count     equ  250
iv_on         equ  250
iv_off        equ  100

on_time:   ds      1
off_time:     ds      1

repeat:
        bsf     LATA2   ;
        call    checkkey        ;
        movf    on_time,w
        call    delay_200ms ;
        bcf     LATA2   ;
        call    checkkey        ;
        movf    off_time,w
        call    delay_200ms ;
        goto    repeat
```

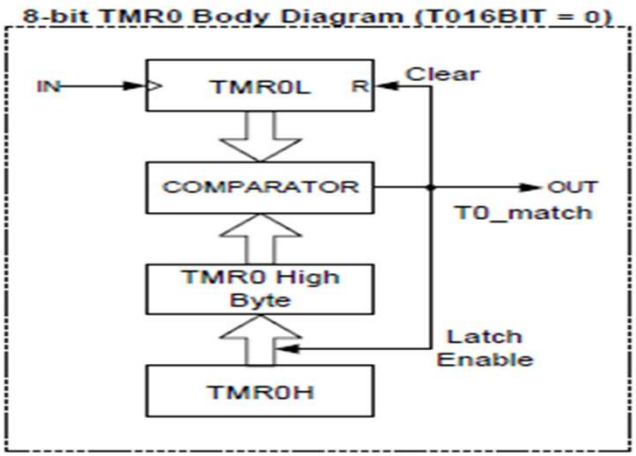
```
checkkey:
        btfss   SW3
        goto    pushkey
        movlw   on_count
        movwf   on_time
        movlw   off_count
        movwf   off_time
        return
pushkey:
        movlw   iv_on
        movwf   on_time
        movlw   iv_off
        movwf   off_time
        return
```

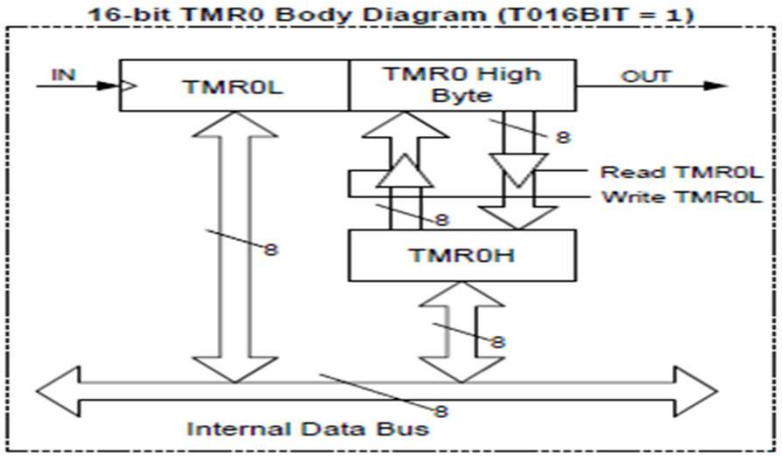
計時器(Timer)

- **MCU**內部的計時器
  - Timer 0 : 8/16 bit timer/counter
  - Timer1/3/5 : 16 bit timer/counter
  - Timer/2/4/6 : 8 bit timer
  - Signal Measurement Timer(SMT): 24 bit timer/counter

Timer 0方塊圖







Timer 0暫存器

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00 ... 0x059B	Reserved									
0x059C	TMR0L	7:0	TMR0L[7:0]							
0x059D	TMR0H	7:0	TMR0H[7:0]							
0x059E	T0CON0	7:0	T0EN		T0OUT	T016BIT	T0OUTPS[3:0]			
0x059F	T0CON1	7:0	T0CS[2:0]			T0ASYNC	T0CKPS[3:0]			

Bits 3:0 – T0OUTPS[3:0] TMR0 Output Postscaler (Divider) Select

Value	Description
1111	1:16 Postscaler
1110	1:15 Postscaler
1101	1:14 Postscaler
1100	1:13 Postscaler
1011	1:12 Postscaler
1010	1:11 Postscaler
1001	1:10 Postscaler
1000	1:9 Postscaler
0111	1:8 Postscaler
0110	1:7 Postscaler
0101	1:6 Postscaler
0100	1:5 Postscaler
0011	1:4 Postscaler
0010	1:3 Postscaler
0001	1:2 Postscaler
0000	1:1 Postscaler

Bits 7:5 – T0CS[2:0] Timer0 Clock Source Select

Table 18-1. Timer0 Clock Source Selections

T0CS	Clock Source
111	CLC1_out
110	SOSC
101	MFINTOSC(500 kHz)
100	LFINTOSC
011	HFINTOSC
010	$F_{OSC}/4$
001	Pin selected by T0CKIPPS (Inverted)
000	Pin selected by T0CKIPPS (Noninverted)

Bit 4 – T0ASYNC TMR0 Input Asynchronization Enable

Value	Description
1	The input to the TMR0 counter is not synchronized to system clocks
0	The input to the TMR0 counter is synchronized to $F_{OSC}/4$

Bits 3:0 – T0CKPS[3:0] Prescaler Rate Select

Value	Description
1111	1:32768
1110	1:16384
1101	1:8192
1100	1:4096
1011	1:2048
1010	1:1024
1001	1:512
1000	1:256
0111	1:128
0110	1:64
0101	1:32
0100	1:16
0011	1:8
0010	1:4
0001	1:2
0000	1:1

lab6

- 實驗目標 : 因為MCU內部都有Timer, 為了讓CPU有多餘的時間去執行其它事情,所以讓計時的工作就交給Timer去做
- Timer 0 clock來源選MFINTOSC(500KHz),所以每個clock週期為2us.
- 為了取代1ms軟體延遲,所以prescaler設為1:2, postscaler設為1:1, 且為8 bit 模式,TMR0H設250,即可滿足所需
- 為了知道Timer 0計時是否已經到了設定時間,請檢查PIR0 : TMR0IF bit https://youtu.be/nmQfRyu0sQ8

需要加程式至lab6

```
; setting Timer 0 operation mode
        banksel T0CON0
        movlw   0b10100001
        movwf   T0CON1
        movlw   0b00000000
        movwf   T0CON0
        movlw   time_base
        movwf   TMR0H
        clrf    TMR0L
        banksel PIR0
        bcf     TMR0IF
```

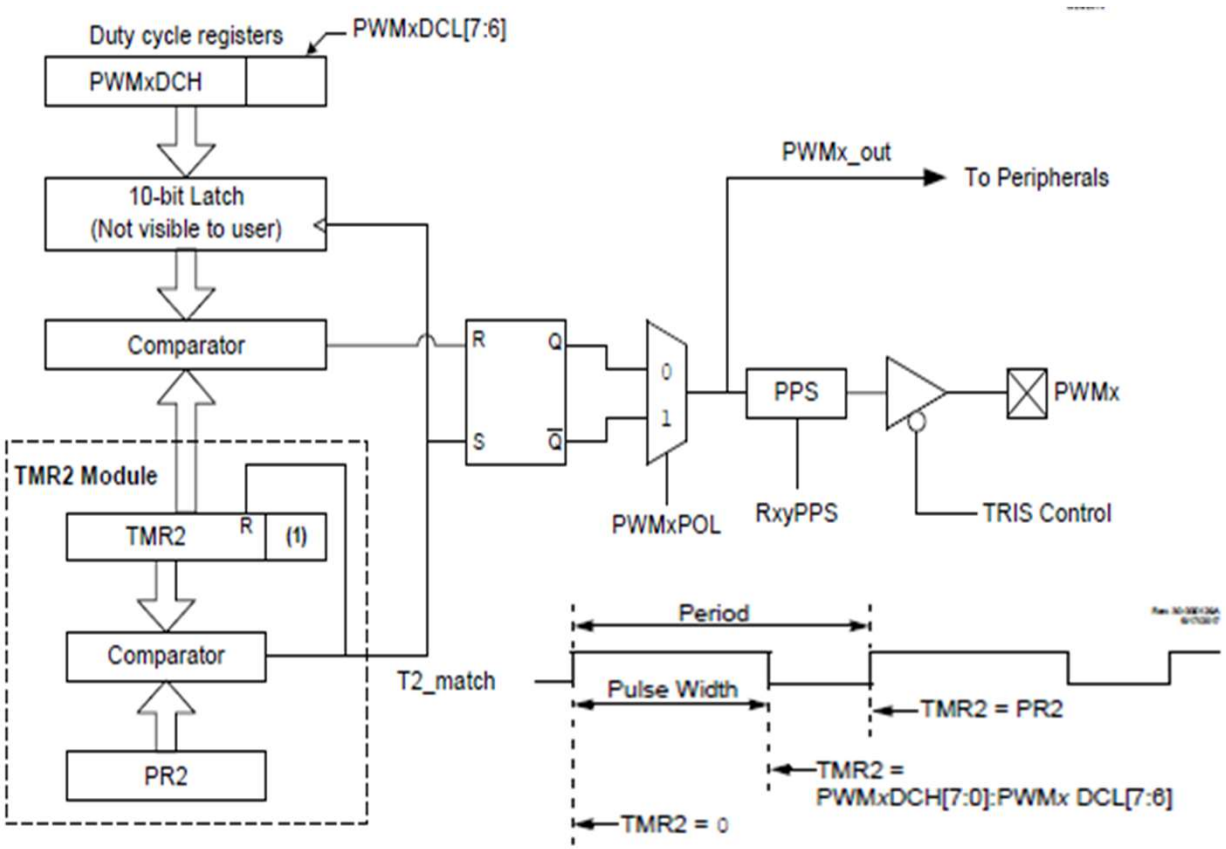
```
repeat:
        banksel LATA
        bsf     LATA2   ;
        call    checkkey
        movf    on_time,w
        call    delay_200ms ;
        banksel LATA
        bcf     LATA2   ;
        call    checkkey
        movf    off_time,w
        call    delay_200ms ;
        goto    repeat
```

```
delay_200ms:
        movwf   c_200
        banksel T0CON0
        bsf     T0EN
loop_1:
        banksel PIR0
        btfss   TMR0IF
        goto    loop_1
        bcf     TMR0IF
        banksel c_200
        decfsz  c_200,f
        goto    loop_1
        banksel T0CON0
        bcf     T0EN
        return
```

PWM(一)

- **Capture/Compare/PWM module : CCP1 ~ 4**
  - Capture/Compare mode clock來源可以是 Timer1,3 or 5
  - PWM mode clock來源可以是 Timer2, 4 or 6
- **PWM6 & 7 : clock來源可以是 Timer2, 4 or 6**
- 為了簡化選用**PWM6**來控制**LED1**的亮度, **clock**來源選用**Timer 2**
- 並將**PWM**輸出到**RA2,**所以也要用到**Peripheral Pin Select(PPS)**功能
- 所以此實驗總計會使用到**PWM6, Timer 2 & PPS**三種**chip**內部功能整合使用

PWM方塊圖 &輸出波形(二)



- 使用**PWM**時關注的問題
  - PWM工作頻率
  - PWM解析度, ?bit
  - PWM輸出的工作模式
    - 互補式輸出
    - 同步輸出
    - 半橋式輸出
    - 全橋式輸出
    - 相位偏移…

MICROCHIP

PWM相關計算公式(三)

Equation 24-1. PWM Period

$$PWMPeriod = [(T2PR) + 1] \bullet 4 \bullet Tosc \bullet (TMR2\ \mathrm{Prescale}Value)$$

Note: $T_{OSC} = 1/F_{OSC}$

Equation 24-2. Pulse Width

$$PulseWidth = (PWMxDCH{:}PWMxDCL[7{:}6]) \bullet Tosc \bullet (TMR2\mathrm{Prescale}Value)$$

Note: $T_{OSC} = 1/F_{OSC}$

Equation 24-3. Duty Cycle Ratio

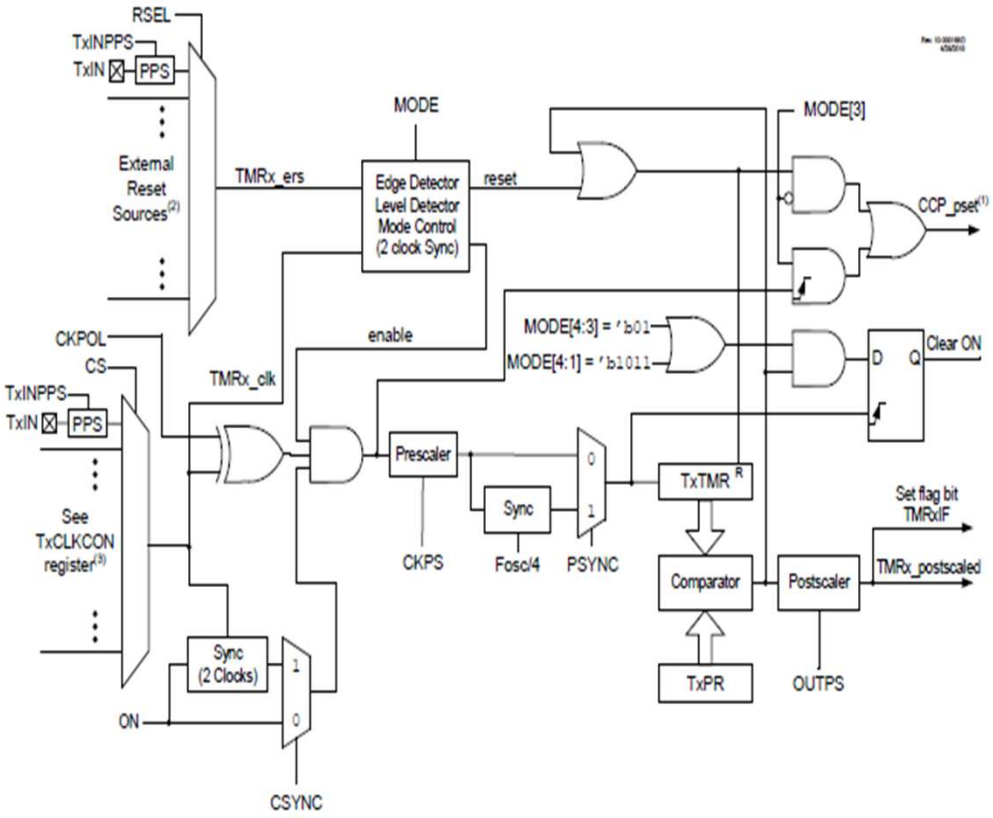
$$DutyCycleRatio = \frac{(PWMxDCH{:}PWMxDCL[7{:}6])}{4(T2PR + 1)}$$

Equation 24-4. PWM Resolution

$$Resolution = \frac{\log[4(T2PR + 1)]}{\log(2)} bits$$

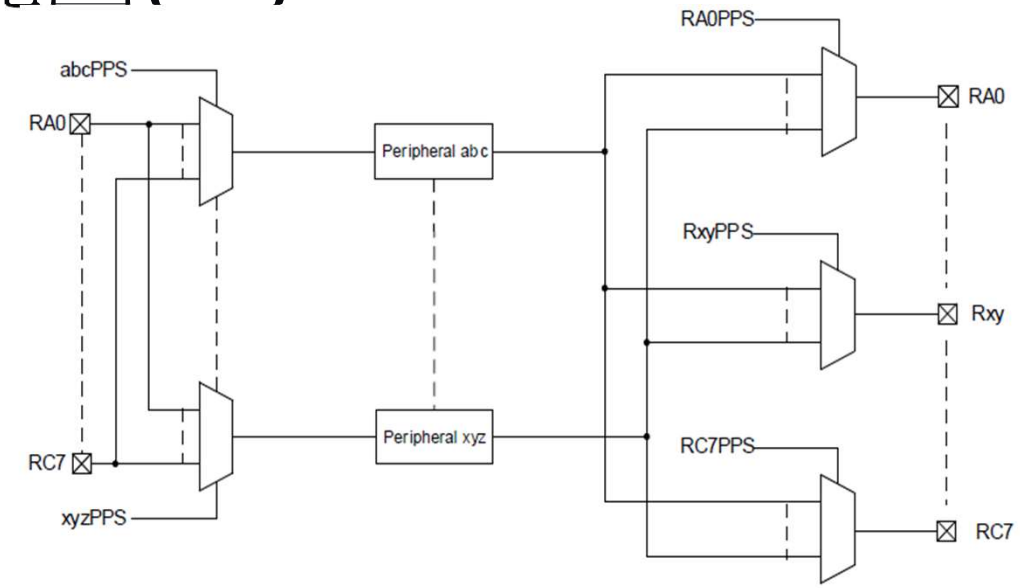
Timer 2方塊圖

- 選擇clock source : Fosc/4
- 工作模式 : Period Pulse software gate, mode[4:0] = 00000
- Prescale & Postscale = 1:1
- 若TxPR =249, 解析度~10 bit, 在此條件下PWM的頻率可達4KHz



PPS方塊圖(一)

- 只有數位訊號的輸入&輸出才可使用此功能
- 意即數位訊號的輸入or輸出腳可以任意設定要從MCU的那一個I/O pin當成輸入or輸出pin,此功能可以讓硬體設計更有彈性
- 每個周邊功能都有一個輸入PPS register, 以供設定輸入的IO pin



Input Signal Name	Input Register Name	Default Location at POR (14/16-pin devices)	Reset Value (xxxPPS[4:0]) 14/16-pin devices	Default Location at POR (20-pin devices)	Reset Value (xxxPPS[4:0]) 20-pin devices	PORT From Which Input Is Available		
INT	INTPPS	RA2	00010	RA2	00010	A	B	—
T0CKI	T0CKIPPS	RA2	00010	RA2	00010	A	B	—

PPS方塊圖(二)

- 每個輸入腳都有一個固定編碼,將此編碼寫入各周邊PPS register即可完成輸入PPS設定
- 每個周邊輸出也同樣有一個編碼,將此編碼填入各IO的PPS register即可完成輸出的設定
- 例如PWM6要輸出到RA2, 所以將001101填入RA2PPS register即可

Desired Input Pin	Value to Write to Register
RC7	01 0111
RC6	01 0110
RC5	01 0101

RxyPPS Register Value	Output Signal Name	Remappable to Any PORT (14-pin devices)	Remappable to Any PORT (20-pin devices)
10 0000	ADCGRDB	X	X
01 1111	ADCGRDA	X	X
01 1110	CWG2D	X	X
01 1101	CWG2C	X	X
01 1100	CWG2B	X	X
01 1011	CWG2A	X	X
01 1010	DSM1OUT	X	X
01 1001	CLKR	X	X
01 1000	NCO1OUT	X	X
01 0111	TMR0OUT	X	X
01 0100	SDO1/SDA1	X	X
01 0011	SCK1/SCL1	X	X
01 0010	C2OUT	X	X
01 0001	C1OUT	X	X
01 0000	DT1	X	X
00 1111	CK1/TX1	X	X
00 1110	PWM7OUT	X	X
00 1101	PWM6OUT	X	X

lab7

- 實驗目標：此實驗是本課程中最複雜的一個實驗,會用到中斷, Timer0, Timer 2 & PPS, 利用PWM6輸出到LED1做一個呼吸燈
- 需要定時去更新PWM輸出Duty, 所以可以利用前面實驗的Timer 0為time base. 但前面的實驗用polling的方式知道時間的更新,還是需要CPU不斷去看旗標,這樣還是浪費CPU資源,此實驗改用中斷(interrupt)的方式.
- 中斷向量位於0x04, Timer 0的中斷服務程式必須從0x04的位址開始
- 從前面的實驗開始我們均將Timer 0的時間週期設為1ms,以此時間週期去調整PWM的輸出速度太快,人眼無法辨別,故需再加一個count延遲時間,設為延遲7ms.
- 為了簡化程式, PWM duty控制只修改PWM6DCH
- 嘗試修改delay counter,看LED顯示的效果

https://youtu.be/NEPIfj-CIlw

需要加程式至lab7

```
#define     LEDstaus    status,0
time_base   equ 250
d_count     equ 7
period      equ 249
max_duty    equ period+1

PSECT   udata
count:  ds      1
status: ds      1


dec_pwm:
        banksel PWM6DCH
        decf    PWM6DCH,f
        btfss   ZERO
        retfie
        banksel status
        bcf     LEDstaus
        retfie
```

```
PSECT isrVec, class=CODE,delta=2
isr:
        pagesel $
        banksel PIR0
        btfss   TMR0IF
        retfie
        bcf     TMR0IF
        banksel count
        decfsz  count,f
        retfie
        movlw   d_count
        movwf   count
        banksel status
        btfsc   LEDstaus
        goto    dec_pwm
        banksel PWM6DCH
        incf    PWM6DCH,f
        movlw   max_duty
        subwf   PWM6DCH,w
        btfss   ZERO
        retfie
        banksel status
        bsf     LEDstaus
        retfie
```

需要加程式至lab7

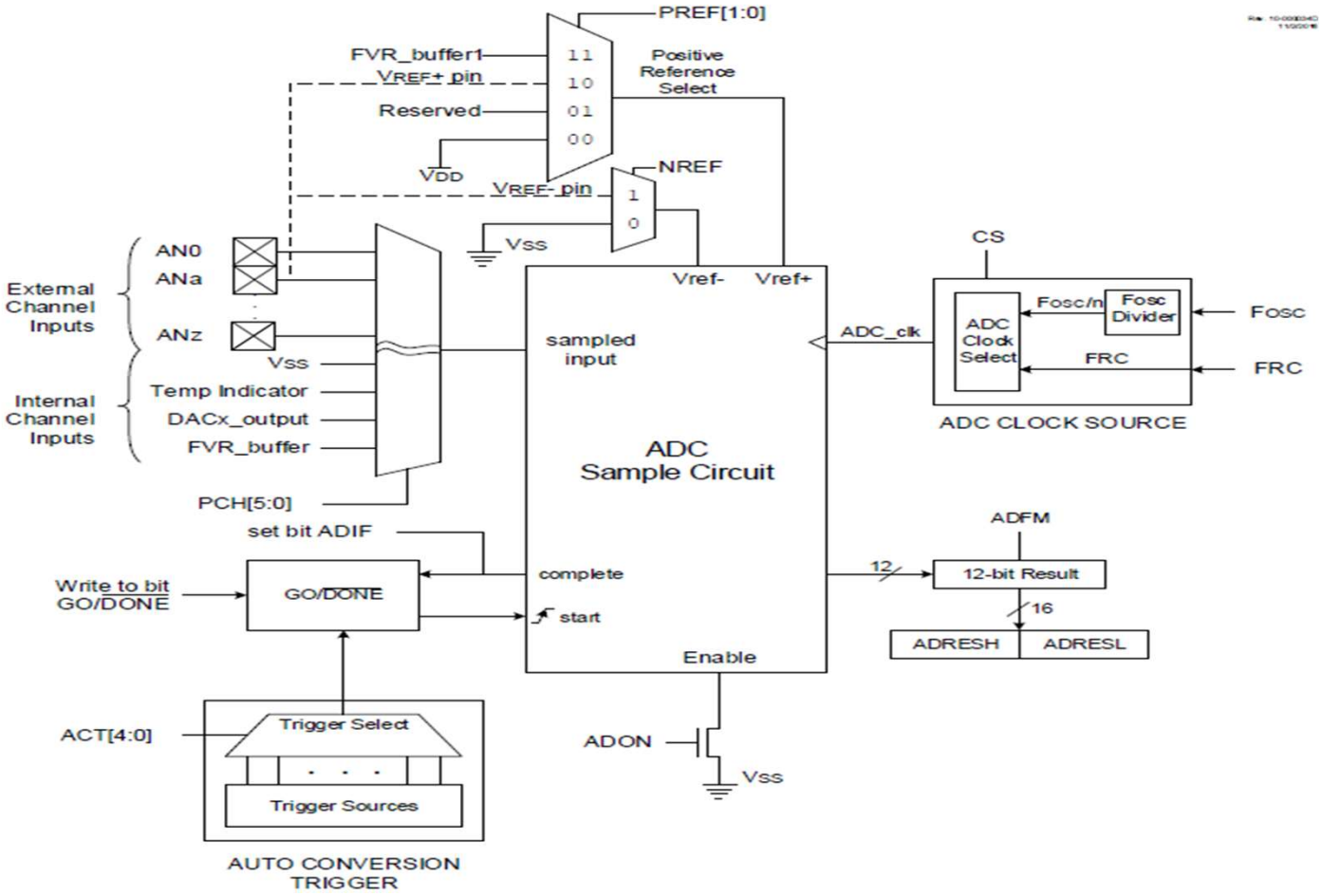
```
; setting PWM6 clock source from Timer2
        banksel CCPTMRS1
        movlw   0b00000100  ; set PWM6 clock source from Timer2
        movwf   CCPTMRS1

;Setting PWM6 working register
        banksel PWM6DCL
        clrf    PWM6DCL     ; clear PWM6 output low duty register
        clrf    PWM6DCH     ; clear PWM6 output high duty register
        movlw   0x80
        movwf   PWM6CON     ; enable PWM6 output, output polarity as normal
; setting Timer 2 working register
        banksel T2TMR
        clrf    T2TMR       ; clear Timer2 counter register
        movlw   period
        movwf   T2PR        ; setting PWM6 period
        clrf    T2HLT       ; setting Timer 2 working mode
        movlw   0b00000001  ; Fosc/4 is Timer 2 clock source
        movwf   T2CLKCON    ;
        movlw   0b00001000  ;
        movwf   T2RST
        movlw   0b10000000  ; enable Timer 2, Prescale & Postscale 1:1
        movwf   T2CON
```

```
; PWM output set to RA2
        banksel RA2PPS
        movlw   0b00001101
        movwf   RA2PPS


        banksel status
        bcf     LEDstaus
; setting delay counter
        movlw   d_count
        movwf   count
; enable Timer 0 interrupt
        banksel PIE0
        bsf     TMR0IE
; enable Timer 0
        banksel T0CON0
        bsf     T0EN
; enable global interrupt
        banksel INTCON
        bsf     GIE
```

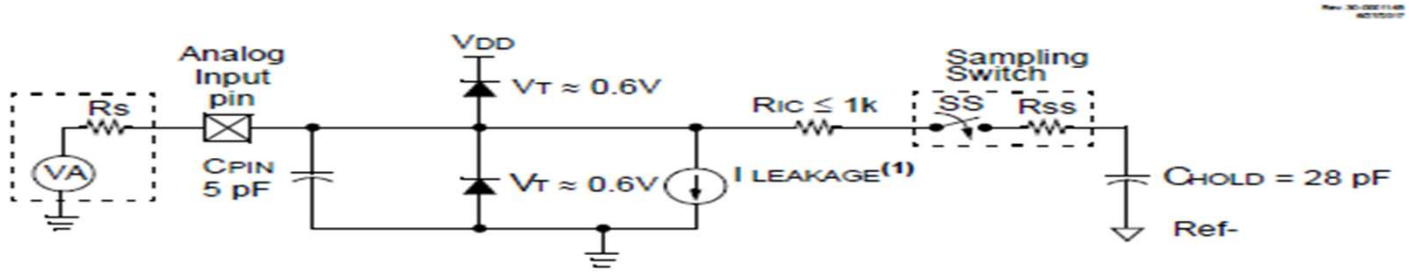

MICROCHIP

類比 / 數位 轉換器 (ADC)關注事項&方塊圖

- 解析度
- 轉換速度
- 量測電壓範圍
- 量測誤差
- 輸入訊號的組數

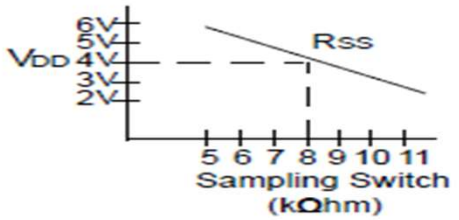


類比 / 數位 轉換器 (ADC)等效電路(一)



Legend:

CHOLD	=	Sample/Hold Capacitance
CPIN	=	Input Capacitance
I LEAKAGE	=	Leakage current at the pin due to various junctions
RIC	=	Interconnect Resistance
RSS	=	Resistance of Sampling Switch
SS	=	Sampling Switch
VT	=	Threshold Voltage



Equation 32-1. Acquisition Time Example

Assumptions: Temperature = 50°C and external impedance pf 10 kΩ, 5.0V V_{DD}

T_{ACQ} = Amplifier Settling Time + Hold Capacitor Charging Time + Temperature Coefficient

= $T_{AMP} + T_C + T_{COFF}$

= 2 µs + T_C + [(Temperature - 25°C)(0.05 µs/°C)]

類比 / 數位 轉換器 (ADC)等效電路(二)

$$V_{APPLIED}\left(1 - \frac{1}{(2^{n+1}) - 1}\right) = V_{CHOLD} \quad ;[1]\ V_{CHOLD}\text{charged to within 1/2 lsb}$$

$$V_{APPLIED}\left(1 - e^{\frac{-T_C}{R_C}}\right) = V_{CHOLD} \quad ;[2]\ V_{CHOLD}\text{charge response to}\ V_{APPLIED}$$

$$V_{APPLIED}\left(1 - e^{\frac{-T_C}{R_C}}\right) = V_{APPLIED}\left(1 - \frac{1}{(2^{n+1}) - 1}\right) \quad ;\text{combining [1] and [2]}$$

Note: Where n = number of bits of the ADC.

Solving for T_C:

$$T_C = -C_{HOLD}(R_{IC} + R_{SS} + R_S)\ \ln(1/8191)$$

$$T_C = -28\,pF(1\,k\Omega + 7\,k\Omega + 10\,k\Omega)\ \ln(0.0001221)$$

$$T_C = 4.54\,us$$

Therefore:

$$T_{ACQ} = 2\,us + 4.54\,us + [(50°C - 25°C)(0.05\,us/°C)]$$

$$T_{ACQ} = 7.79\,us$$

MICROCHIP

lab8

- 實驗目標 :在自然界中存在的訊號都是類比訊號,故此實驗嘗試用ADC將自然界的訊號轉換為數位訊號,以利CPU做資料的處理,此實驗版上的可變電阻就是模擬類比訊號,以訊號的大小值調整LED1輸出亮度
- 可變電阻VR1的輸出連接至CON2的AD0, 而Mikro_AN的輸入連接至MCU RC3, 兩者之間並沒有直接連線,故須使用跳線將AD0 & Mikro_AN連接在一起
- 請檢查ANSELC 3 & TRISC3均要設為1
- ADC正端輸入pin為RC3, 故填0b00010011至ADPCH register
- Disable連續轉換,clock source : Fosc 4MHz, 左移,因為只取high byte的AD結果. 故清除ADCON0為0
- 參考電壓為Avdd & Vss,故清除ADREF為0
- 因Tad需為0.5 ~ 9us, = Fosc/(2*(CS+1)), 故寫1至ADCLK, Tad=1us
- 利用Timer 0中斷每隔一段時間trigger ADC開始做轉換
- 因取樣時間為25us,故設ADACQ = 25,也可依前頁計算公式所得設為8us

https://youtu.be/hi4RLYKqoyo

需要加程式至lab8

```
; setting ADC working register
        banksel ADPCH
        movlw   0b00010011  ; select RC3 as analog input
        movwf   ADPCH
        clrf    ADCNT       ; clear repeat counter

        banksel ADACQL
        movlw   time_acq
        movwf   ADACQL      ; setting acquisition time
        clrf    ADACQH
        clrf    ADCAP       ; setting additional sample capacitor
        clrf    ADPREH      ; clear precharge time control register
        clrf    ADPREL
        clrf    ADCON0      ; disable continue mode, clock : Fosc, left-justified
        clrf    ADCON1
        clrf    ADCON2
        movlw   0x70
        movwf   ADCON3
        clrf    ADREF       ; negation ref : Vss, Positive ref : Vdd
        movlw   0b00011011
        movwf   ADACT       ; disable auto-conversion trigger source
        movlw   ad_clk
        movwf   ADCLK       ; setting ADC clock frequence, 1Mhz, request Tad= 0.5 ~ 9us
        clrf    ADCPCON0
        bsf     ADON
```

```
;add trigger ADC converter in interrupt routine
        banksel ADCON0
        bsf ADGO
        retfie
```

```
; in main routine, check conversion complete
repeat:
        banksel ADCON0
        btfss   ADGO
        goto    repeat
repeat1:
        btfsc   ADGO
        goto    repeat1
        banksel ADRESH
        movf    ADRESH,w
        banksel PWM6DCH
        movwf   PWM6DCH
        goto    repeat
```

總結

- 當要求精準時間控制時可以使用組合語言
- 在C語言中也可以使用in-line assembler, 以達到精簡程式
- 因軟體從來沒有bug free,所以當用C寫程式時,有時需要到listing file才能找出bug,這時就需要有assembler語言的能力

Thank You!

各位辛苦了