dsPIC 組合語言

Getting Started Programming
on the dsPIC30F

課程目標

- 認識 dsPIC 基本架構與使用 ASM30 撰寫 dsPIC 的組合語言
- 經由實做與程式的修改，了解如何正確選擇內建的周邊與dsPIC 的功能

課程內容 (一)

- 架構說明
- 定址模式概述
- ASM30 語法
- 中斷控制
  - 實驗一
- 12-bit A/D 轉換模組
  - 實驗二
- DCI 資料轉換模組
  - 實驗二　(continued)

課程內容 (二)

- UART 非同步串列通訊模組
  - 實驗二　(continued)
- 系統可靠度、時脈系統與電源管理
  - 實驗三
- Capture, Compare, Timers
  - 實驗四
- PSV 程式記憶空間的映對
  - 實驗五

dsPIC30F 成員

- 三種 dsPIC 家族成員
  1. 功率轉換及馬達控制系列
  2. 泛用型訊號處理控制系列
  3. 感應器處理系列

dsPIC30F: 16-bit MCU with the power of DSP

dsPIC30F

功率轉換及馬達控制系列

Product dsPIC(R) DSC	Pins	Flash KB	SRAM Bytes	EE Bytes	Timer 16-bit	Input Cap	Output Comp/ Std PWM	Motor Cntrl PWM	A/D 10-bit 500 KSPS	Quad Enc	UART	SPI (TM)	I2C (TM)	CAN
dsPIC30F2010	28	12	512	1024	3	4	2	6	6 ch	Yes	1	1	1	
dsPIC30F3010	28	24	1024	1024	5	4	2	6	6 ch	Yes	1	1	1	
dsPIC30F4012	28	48	2048	1024	5	4	2	6	6 ch	Yes	1	1	1	1
dsPIC30F3011	40	24	1024	1024	5	4	4	6	9 ch	Yes	2	1	1	
dsPIC30F4011	40	48	2048	1024	5	4	4	6	9 ch	Yes	2	1	1	1
dsPIC30F5015	64	66	2048	1024	5	4	4	8	16 ch	Yes	1	2	1	1
dsPIC30F6010	80	144	8192	4096	5	8	8	8	16 ch	Yes	2	2	1	2

- 直流無刷馬達控制
- 交流感應馬達控制
- 開關是磁阻馬達控制
- UPS, 變頻器 及 電源供應器
- 家電
- 電動工具
- 汽車電裝
- 工業控制

dsPIC30F

泛用型訊號處理控制系列

Product dsPIC(R) DSC	Pins	Flash KB	SRAM Bytes	EE Bytes	Timer 16-bit	Input Capture	Output Compare Std PWM	A/D 12-bit 100 KSPS	UART	SPI (TM)	I2C (TM)	CAN	Codec Interface
dsPIC30F3014	40	24	2048	1024	3	2	2	13 ch	2	1	1		
dsPIC30F4013	40	48	2048	1024	5	4	4	13 ch	2	1	1	1	AC97, I2S
dsPIC30F5011	64	66	4096	1024	5	8	8	16 ch	2	2	1	2	AC97, I2S
dsPIC30F6011	64	132	6144	2048	5	8	8	16 ch	2	2	1	2	
dsPIC30F6012	64	144	8192	4096	5	8	8	16 ch	2	2	1	2	AC97, I2S
dsPIC30F5013	80	66	4096	1024	5	8	8	16 ch	2	2	1	2	AC97, I2S
dsPIC30F6013	80	132	6144	2048	5	8	8	16 ch	2	2	1	2	
dsPIC30F6014	80	144	8192	4096	5	8	8	16 ch	2	2	1	2	AC97, I2S

dsPIC30F
感應器處理系列

Product dsPIC(R) DSC	Pins	Flash KB	SRAM Bytes	EE Bytes	Timer 16-bit	Input Capture	Output Compare Std PWM	A/D	UART	SPI (TM)	I2C (TM)	CAN
dsPIC30F2011	18	12	1024		3	2	2	12-bit, 8 ch	1		1	
dsPIC30F3012	18	24	2048	1024	3	2	2	12-bit, 8 ch	1		1	
dsPIC30F2012	28	12	1024		3	2	2	12-bit, 10 ch	1		1	
dsPIC30F3013	28	24	2048	1024	3	2	2	12-bit, 10 ch	2		1	

- 玻璃破裂偵測
- 瓦斯漏氣偵測
- 扭力偵測
- 胎壓偵測
- 轉向角度偵測
- 電容式雨量感應
- 低電壓偵測，智慧型感應器
- 氣囊感應處理器
- 壓力感應器
- 震動的感應與量測

基本架構概述

dsPIC 架構概要 (一)

- 改良式的 Harvard 核心架構
- 4M x 24-bit 線性定址程式記憶空間 (PS)
  - 所有的 dsPIC 採用 Flash 的製程
- 64 KB 高定址能力的資料空間 (DS)
  - 資料存取寬度可爲 8 / 16-bit
- 所有的 dsPIC 都內建資料儲存 EEPROM
- 16 x 16-bit W 暫存器
- 增強型的指令集
  - MCU 及 DSP 等級的指令

dsPIC 架構概要(二)

- 彈性化的資料定址模式
  - 採用線性定址
  - Modulo 定址模式 (Circular buffers)
  - Bit Reversed 定址模式
- DSP 運算核心
  - 17-bit x 17-bit 單一執行週期的硬體乘法器
  - 兩組 40-bit 累積器(ACC)
  - 40-stage Barrel Shifter with +/-16-bit shift range
  - Rounding and Saturation Logic

dsPIC 架構概要 (三)

- 54 個中斷向量
  - 7 個可由使用者設定中斷優先權
  - 8 個非遮蓋式中斷 (Non-Maskable Traps)
- 低功耗的電源管理模式
  - 多重振盪器模式
  - 即時振盪時序來源的切換模式
  - 振盪時序失效功能監測與切換
  - 低功耗模式的選擇：SLEEP 和 IDLE 模式

dsPIC30F CPU 基本組成(一)

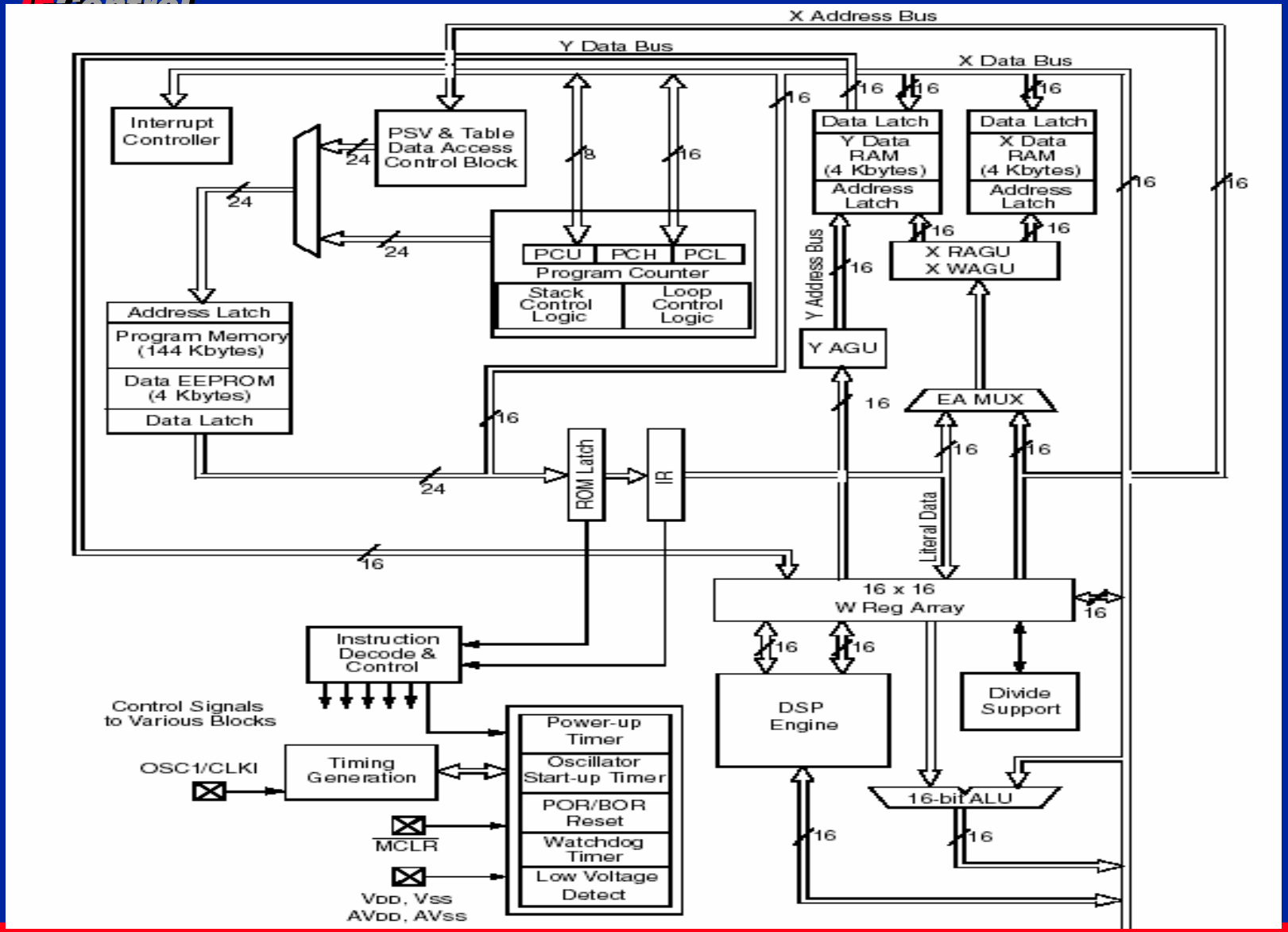
- 組成 CPU 主要的單元：
  - DSP 運算核心
    - ✓ 17x17-bit 整數與小數的硬體乘法器*
    - ✓ 40-bit 管狀式移位器*
    - ✓ 40-bit 加法器 / 減法器
    - ✓ 兩組 40-bit 累積器
    - ✓ 負號擴展及自動填零
    - ✓ 四捨五入及運算飽和邏輯

* 這些單元與 MCU 共用

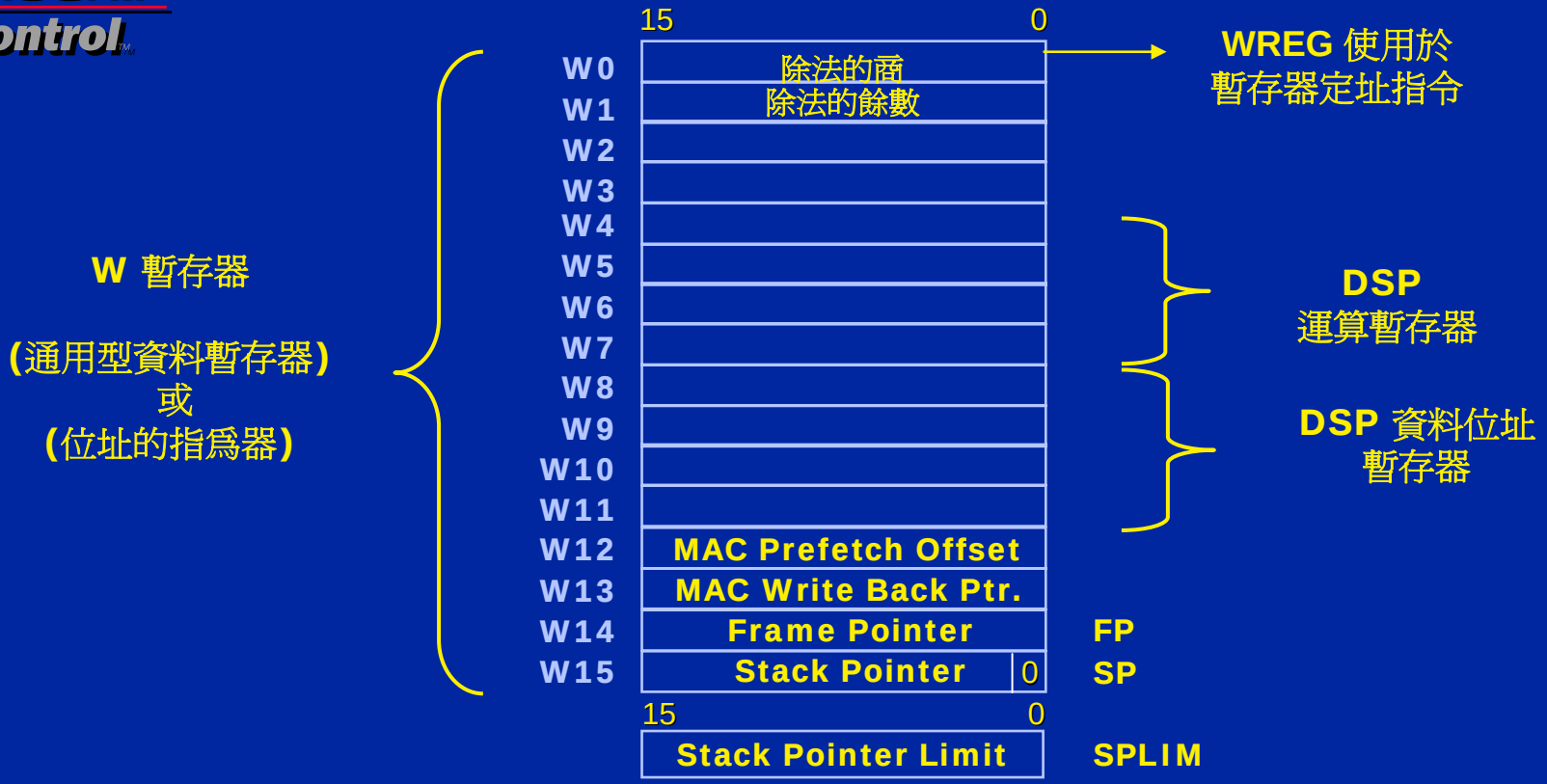
dsPIC30F CPU 基本組成(二)

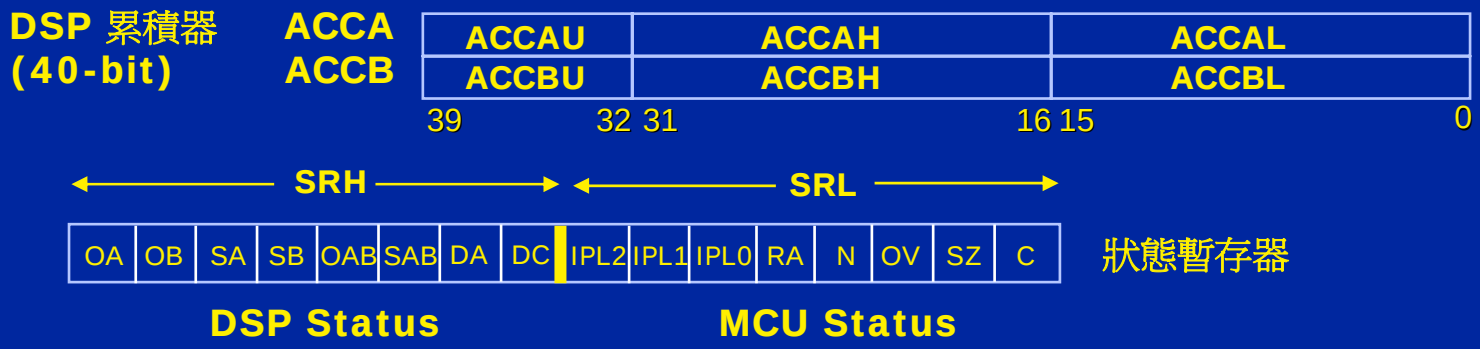
- 構成 MCU 的主要單元
  - 16-bit ALU
  - 除法運算邏輯
  - W 暫存器陣列
  - 位址產生器
- 硬體迴圈控制以支援 DO 及 REPEAT 迴圈指令

dsPIC CPU 方塊圖

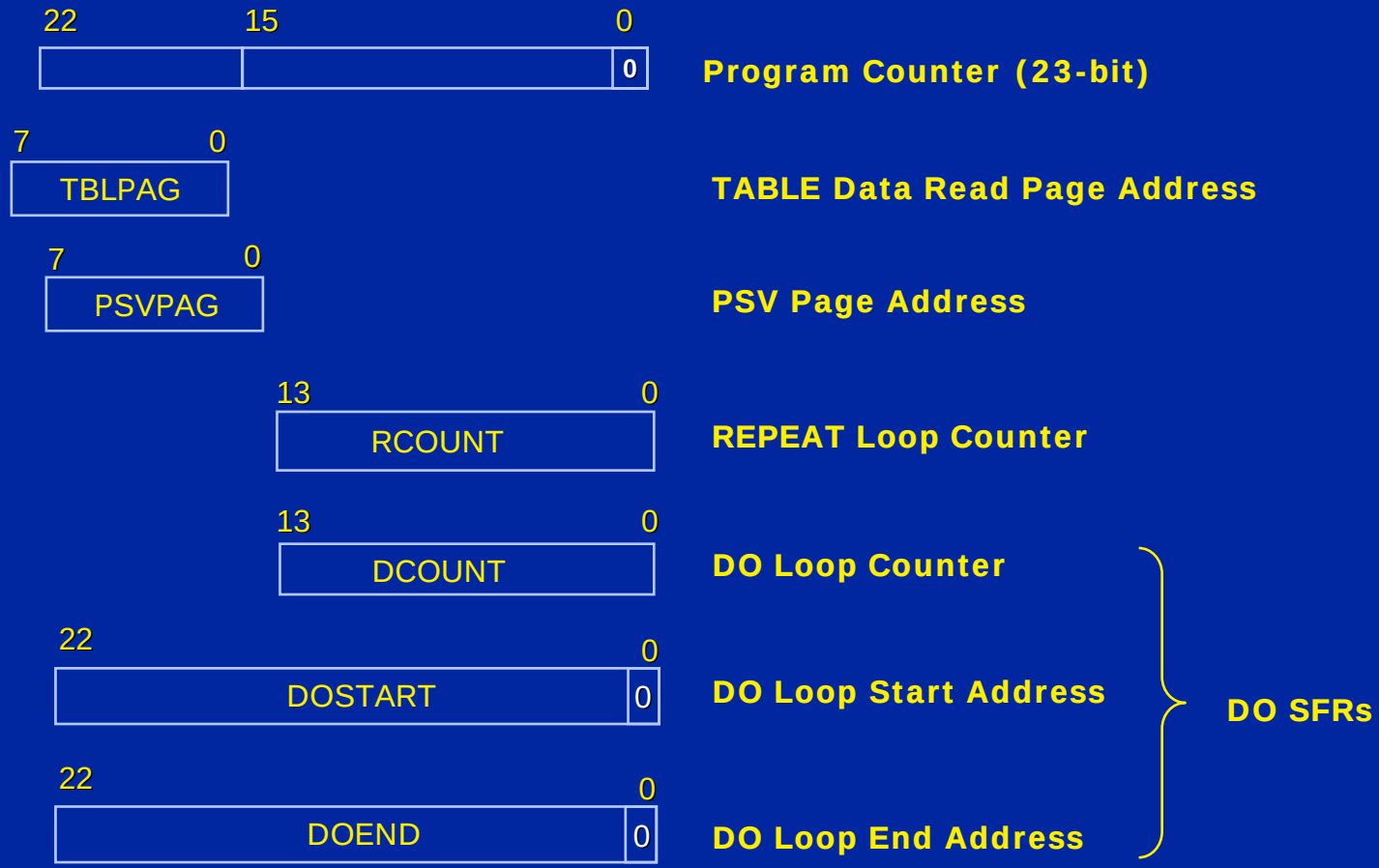


記憶體模型 (一)





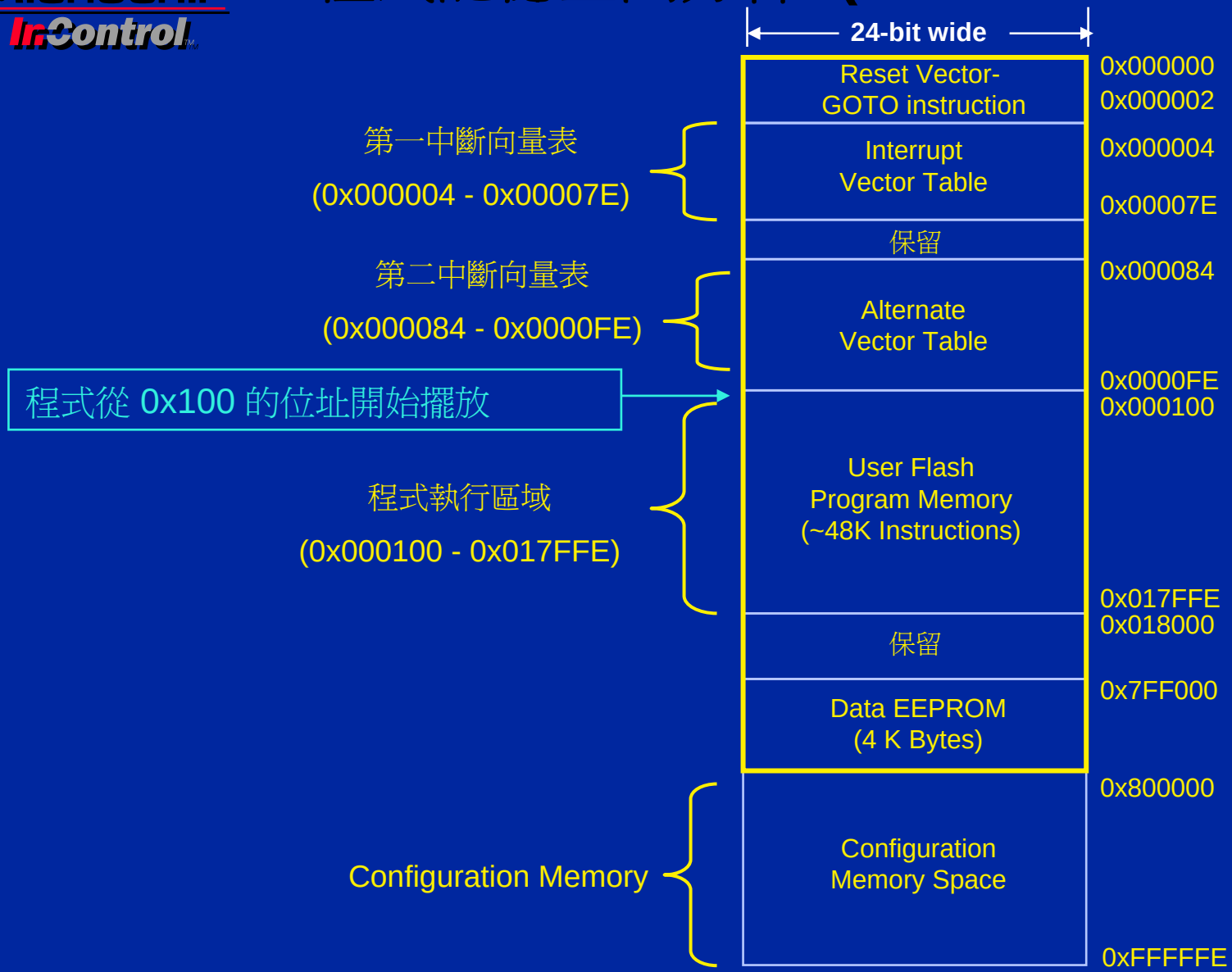
記憶體模型 (二)



程式記憶體的架構

- Linear Program Space (PS)
  - 最多可以有4M x 24-bit 的程式定址空間
  - 指令提取的寬度爲 24-bit
- 程式計數器爲 23-bit (PC<22:0>)
  - PC 自動加二，以提取下一個指令
    - PC<0> 永遠爲 ‘0’，偶數位址的提取

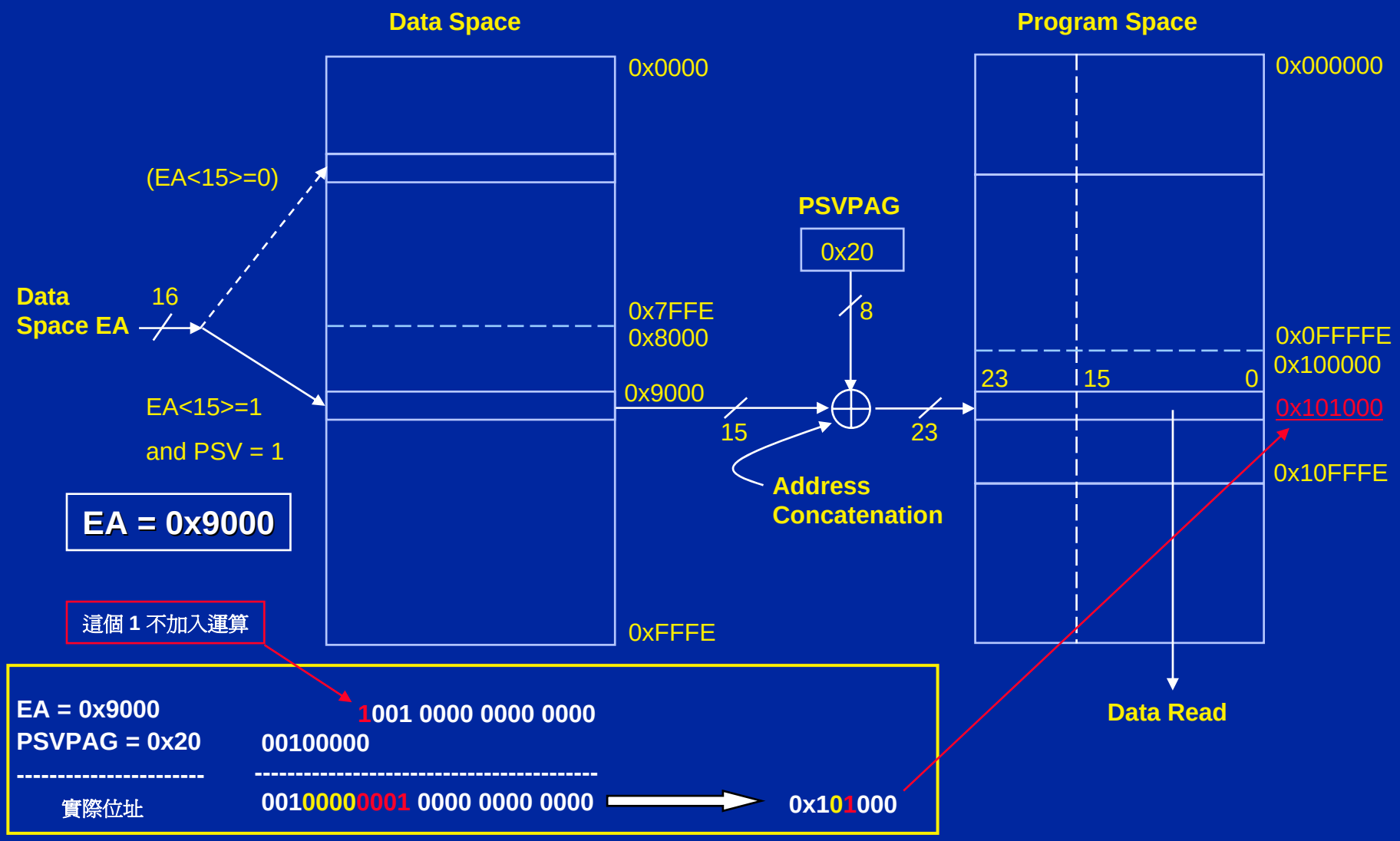
程式記憶空間分佈 (dsPIC30F6014)



程式記憶體讀取的方式

- 三種方式可以讀取程式記憶體
  - 一般是透過 23-bit 程式計數器
    - (PC , 24-bit wide Access)
  - 透過 Table 讀寫指令 (TBLRD and TBLWT)
    - 可以對 Byte 或 Word 操作
  - 採用程式記憶體映對到 32 KB RAM 的操作模式 (Program Space Visibility, PSV)
    - 可以對 Byte 或 Word 操作

PSV 定址模式 – 範例


Data Space
Program Space
0x0000
0x000000
(EA<15>=0)
PSVPAG
0x20
Data
Space EA
16
8
0x7FFE
0x8000
0x0FFFFE
0x100000
23
15
0
0x9000
EA<15>=1
and PSV = 1
15
23
0x101000
Address
Concatenation
0x10FFFE
EA = 0x9000
這個 1 不加入運算
0xFFFE
EA = 0x9000
1001 0000 0000 0000
PSVPAG = 0x20
00100000
實際位址
0010000 0001 0000 0000 0000
0x101000
Data Read

Data Memory 架構 (一)

- Linear Data Memory
  - 最高可以定址到 64 KB Data Memory
  - Data memory 可以用 Byte 方式定址
  - 基本上 dsPIC 的 RAM 是以16-bit (word 型態) 爲主
  - 大部分的指令允許以 Byte 或 Word 兩種方式存取資料
- 資料擺放的方式是以 little-endian 的格式
  - LSB 資料存放在較低位址 (偶數)
  - MSB 資料存放在較高位址 (奇數)

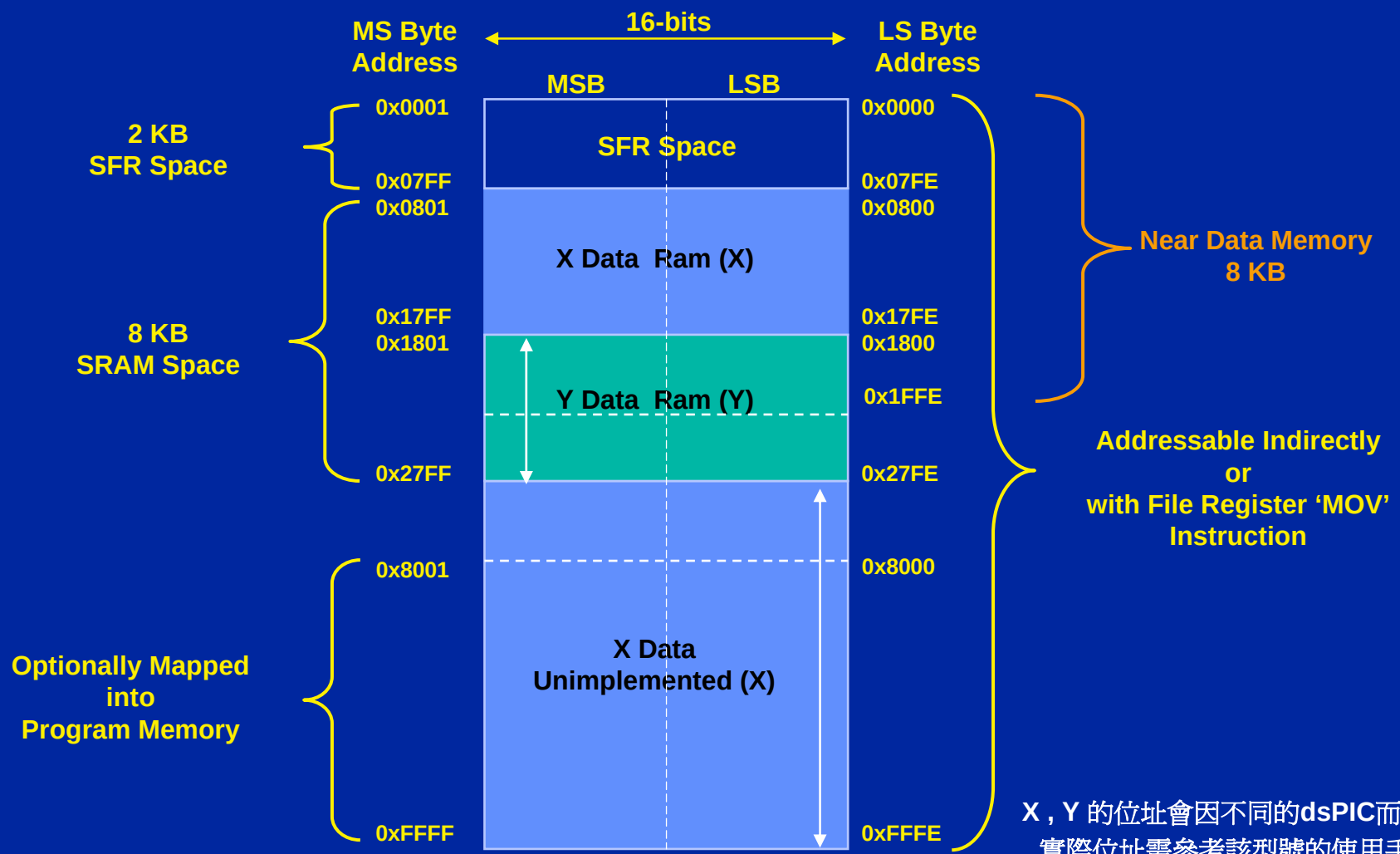
Data Memory 架構 (二)

- 線性定址資料空間的視野
  - 可以爲單一個 X Space 的空間
    - 給MCU級的指令及 DSP級的指令 (除了MAC指令以外)

或 ….

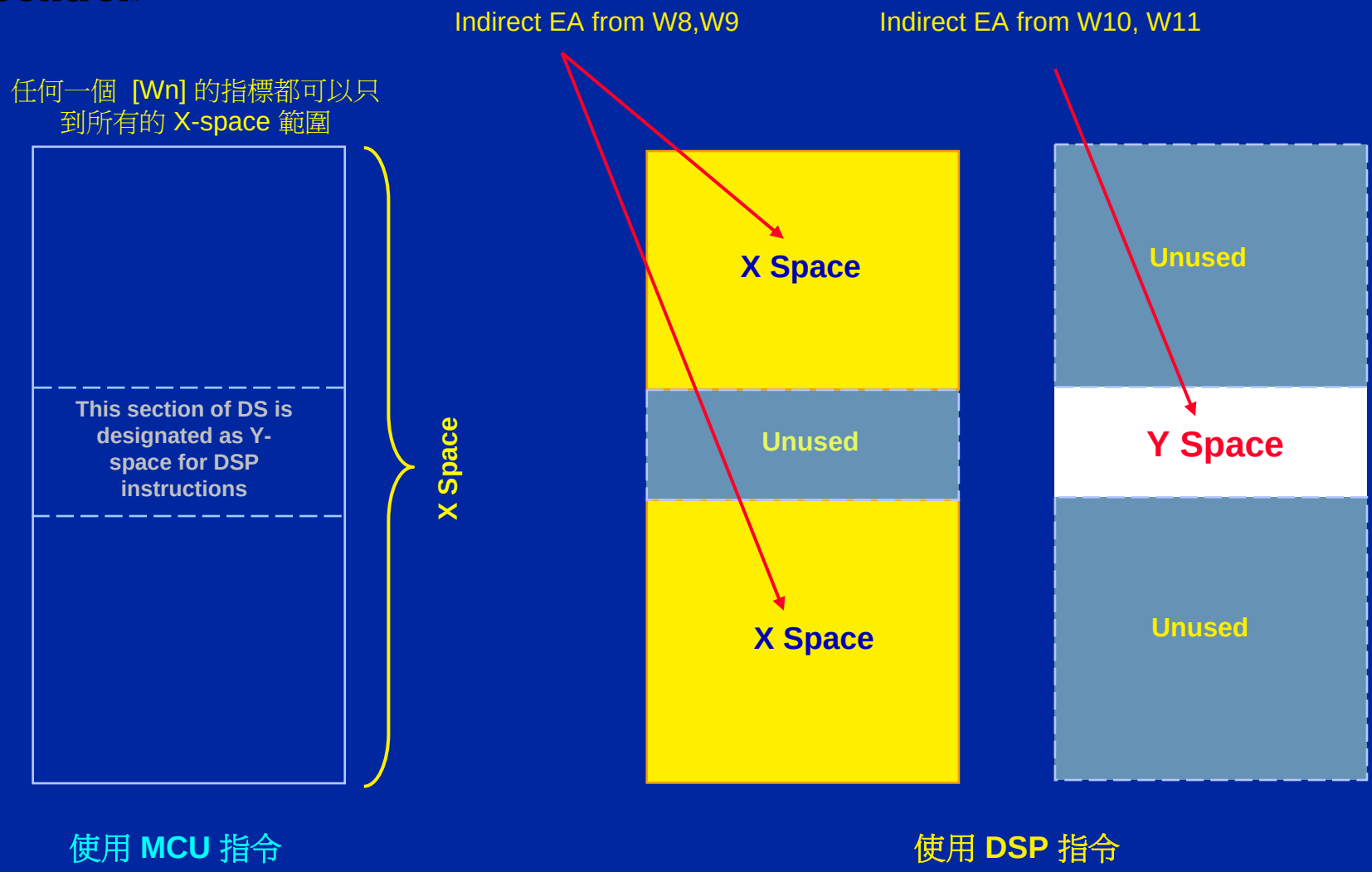
  - 兩個分別獨立空間 (X & Y) 給使用 DSP 的 MAC 指令
    - 允許在同一週期裡同時對 X，Y 兩個資料區同時存取，以提高效率

dsPIC30F6014 資料架構



X , Y 的位址會因不同的dsPIC而變化
實際位址需參考該型號的使用手冊

Data Memory 架構 (三)



Note: 32 KB Section of PS mapped to DS using PSV, is actually mapped to X-DS

資料記憶體
鄰界位址的考慮

- DS word 的存取必須是偶數位址
  - 不合理 DS 存取會產生“位址錯誤”的軟體中斷，如下例所示:
    - MOV 0x1001, W2 ；嘗試讀取奇數位址的資料；產生中斷
    - 指令是完整的，但寫入動作會被禁止(像執行一個 NOP)
  - 資料存取必須是一個有效空間，例如：PSV 是關閉的此時讀取PSV就會錯誤：
    - MOV 0xFEFE, W2 ;嘗試讀取PSV的資料；產生中斷
    - {Note: MOV 0xFEFE, W2 是合理的指令，但PSV沒打開所以錯誤}
    - 指令是完整的，但寫入動作會被禁止(像執行一個 NOP)

定址模式

Addressing Modes

dsPIC30F 基本定址模式

- 基本定址模式：
  - 暫存器定址 / 直接定址
    - 定址範圍在 8 KB 以內，又稱之爲 “Near”
  - W 暫存器直接定址
  - 暫存器索引(間接)定址
  - 立即定址
  - Register indirect with register or literal offset (supported in some instructions)

暫存器定址模式

直接指定的運算位址
範圍在 **8 Kb** 以內 **(Near DM)**

特別指出是
byte 的運算

ADD{.b} 0x900 , {WREG}

或其它
MCU 的指令

假如特別指定是 **WREG** ，
則運算結果就存入 **WREG**，
否則存回原暫存器中

- **{ }** 裡爲 **option** 選項

暫存器定址模式：使用 MOV 指令的定址方式

只能對 **Word** 的運算

直接指定 **64KB** 的偶數位址

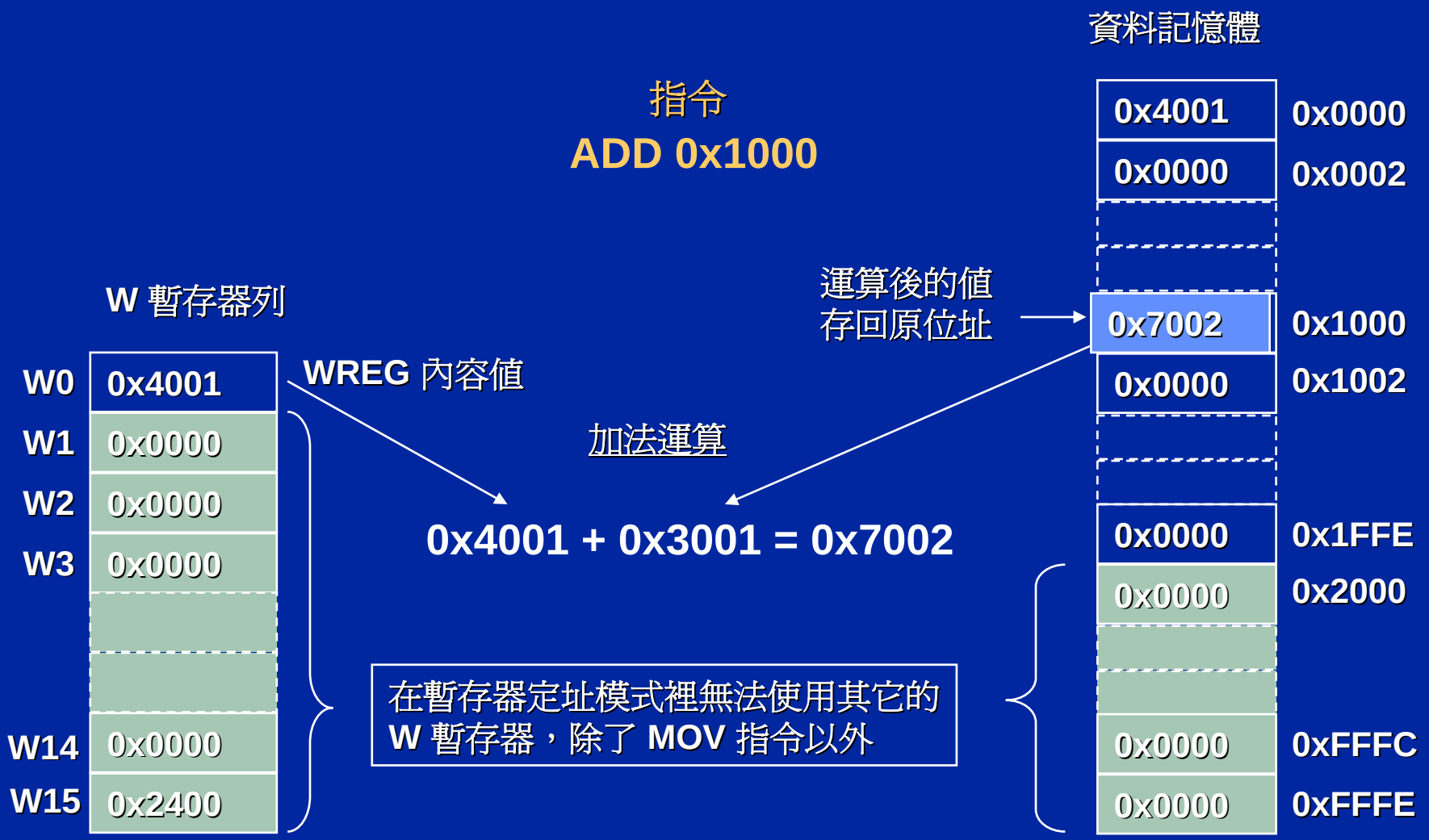
```
MOV    0x2500    , W7
```

只有使用於 **MOV** 指令

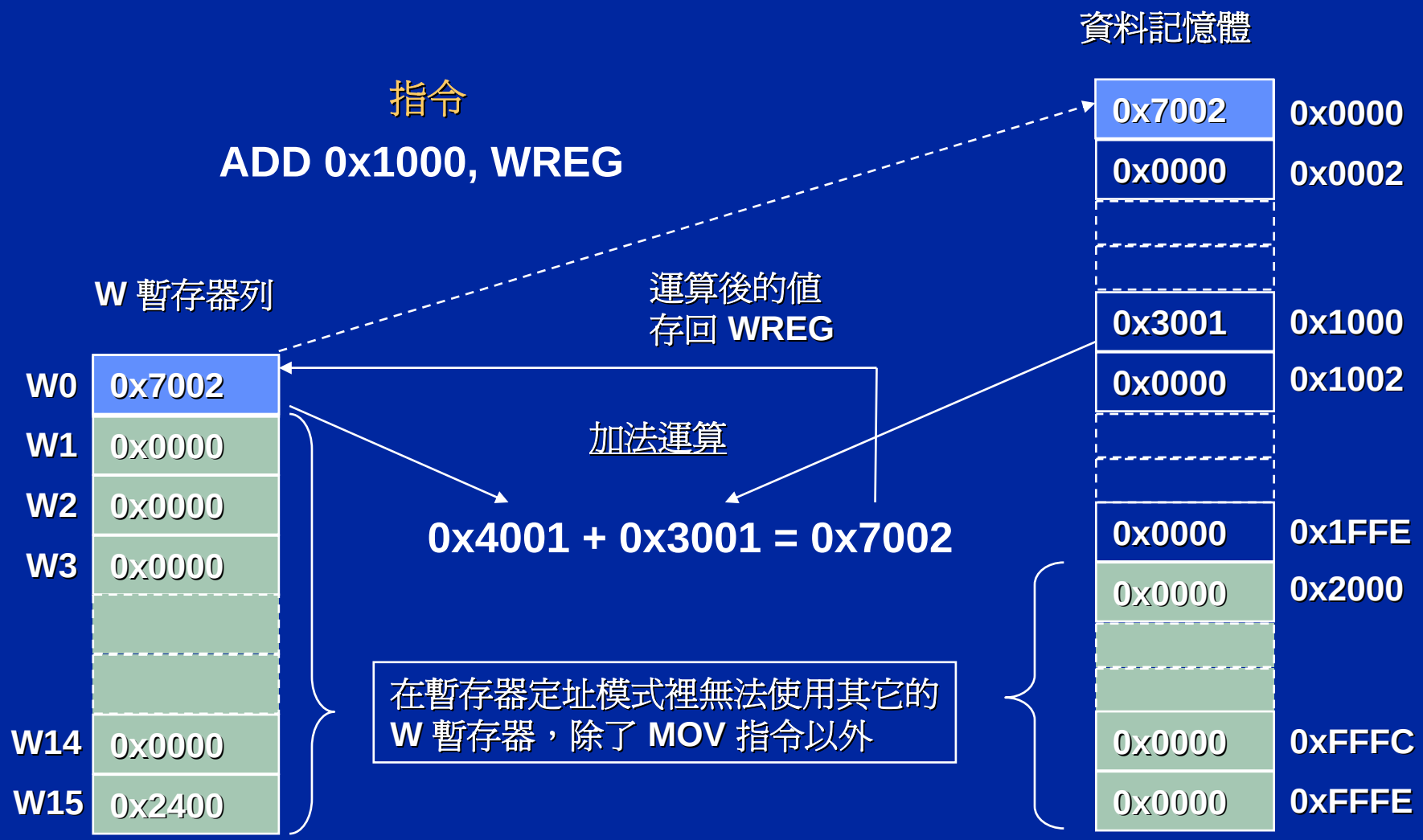
可以指定任何一個 **W** 暫存器

說明：暫存器定址模式 **byte** 的 **MOV** 運算使用 **WREG** 暫存器時定址範圍只有 **8Kb** 以內。
要直接存取**64KB**的範圍只有用間接定址模式或以**word**的模式存取方式**(**如本例**)**

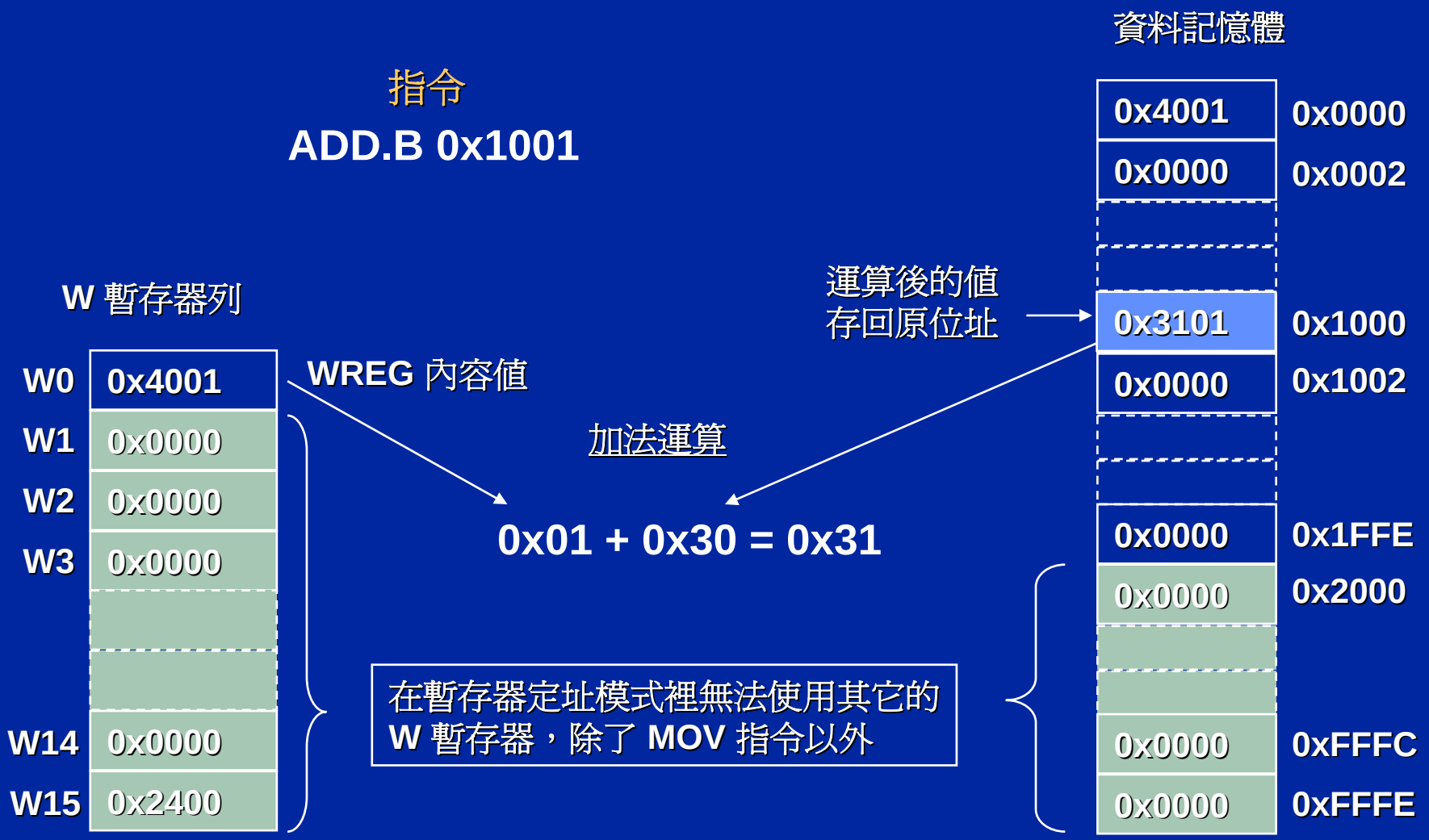
暫存器定址模式 - Near Data 存取範例 (Word)



暫存器定址模式 - Near Data 存取範例 (Word)



暫存器定址模式 - Near Data 存取範例 (Byte)



暫存器定址模式 - 64KB 範圍存取範例 (Word)

指令

MOV W3, 0x2000

暫存器定址模式的語法範例

- 定義兩個 RAM 的變數在 “.nbss” 及 “.ndata” 的節區裡

```
        .section .nbss
MYBUF:  .space 2                ; Word Format
        .section .ndata
MYFLAG: .byte 0x0A
```

- 暫存器定址模式

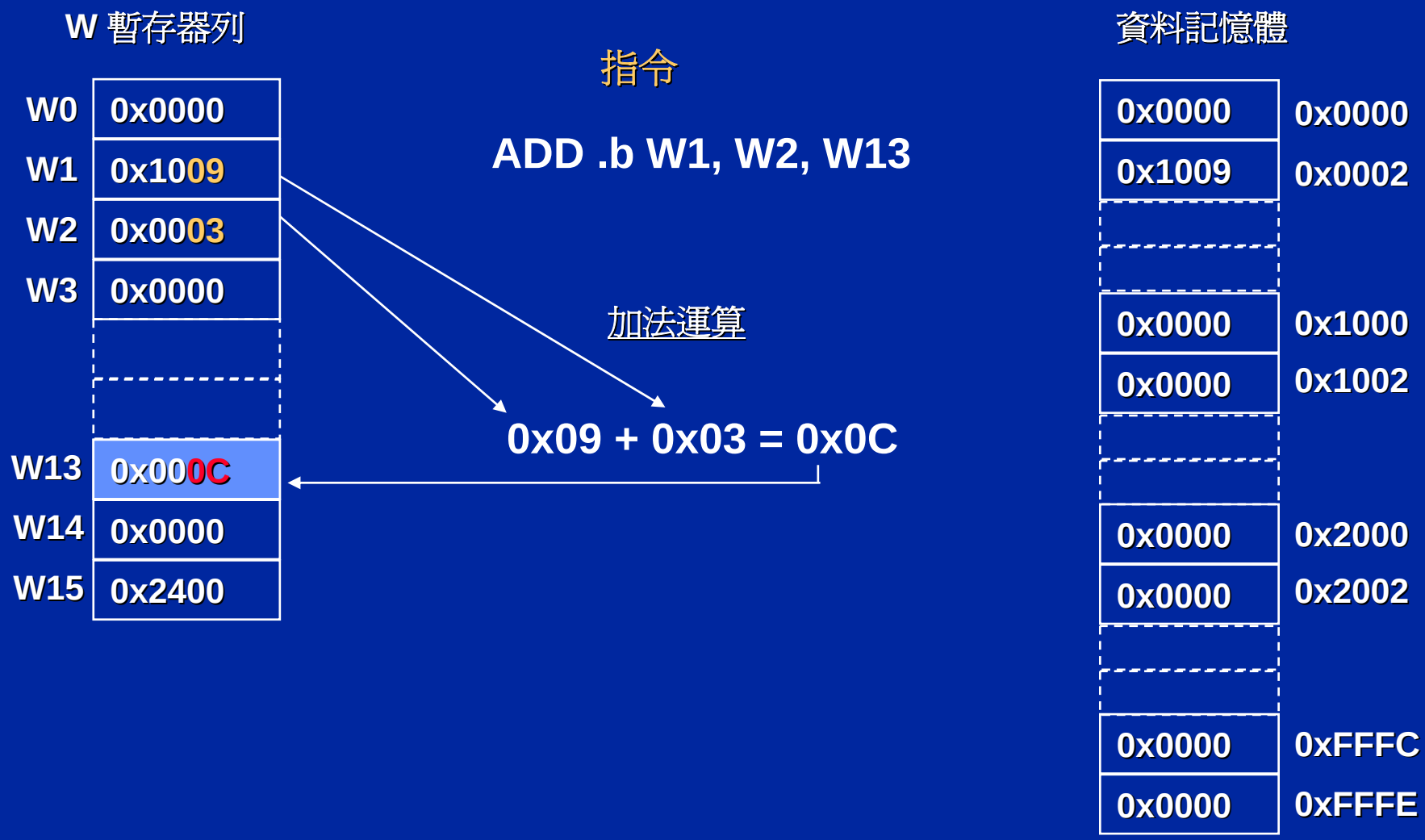
```
        MOV    0x900, WREG      ;Copy data at 0x900 to W0
        MOV    WREG, MYBUF      ;Copy W0 to MYBUF
```

暫存器直接定址模式

- 資料存取直接透過 W 暫存器列
  - W0 - W15 (0x0000 - 0x001F 在 RAM 的位址)
  - 支援 byte 和 word 的操作格式
- 範例：
  - IOR W2, W4, W6
    - ✓ 將 W2 和 W4 做 OR 運算其結果存入W6

暫存器直接定址 - Word 存取

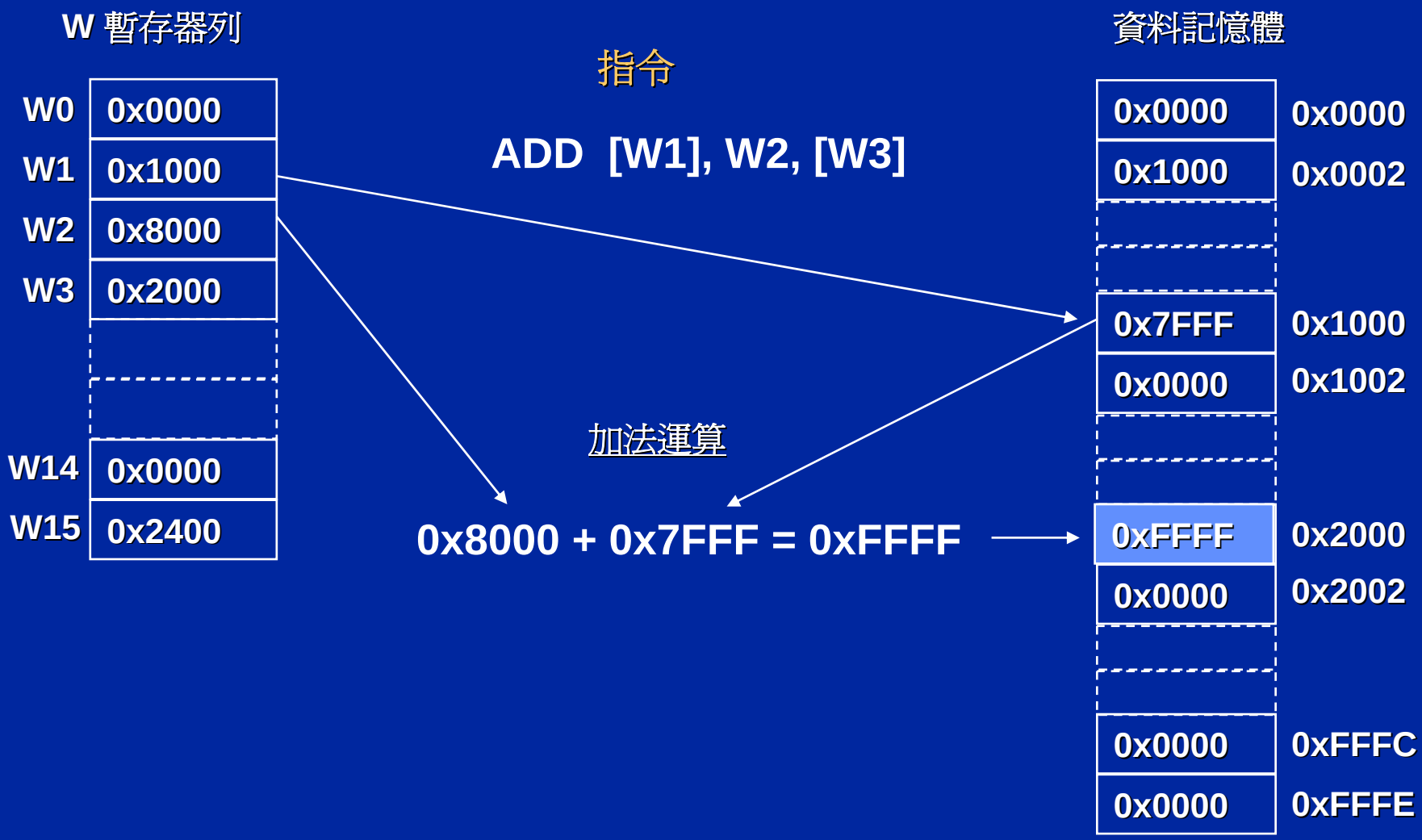
暫存器直接定址 - Byte 存取



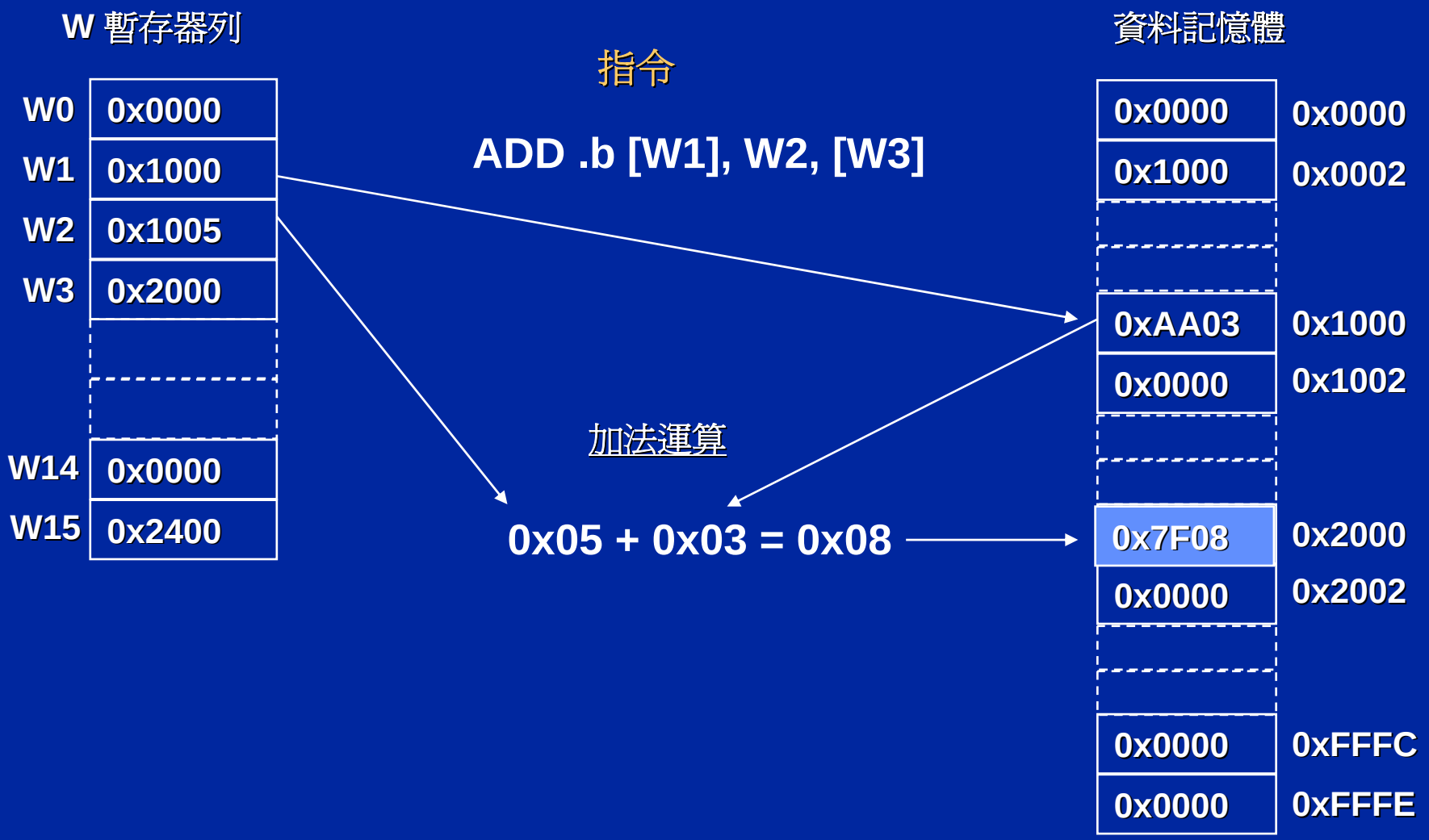
暫存器間接定址模式

- 使用 W 暫存器當作資料的指位器
  - W 暫存器索指向的位址必須是一個有效資料位址 (EA)
  - 間接定址範圍可達 64KB 資料空間
  - 支援大部分的指令
    - 可以存取到 byte 的單位
  - 間接定址模式可以同時使用於 “來源”及“目地”(Source and Destination)
- 間接定址的標記符號
  - 使用 “[ ]” 包住 Wn 暫存器，例 : [W0]

暫存器間接定址模式 - Word 的存取



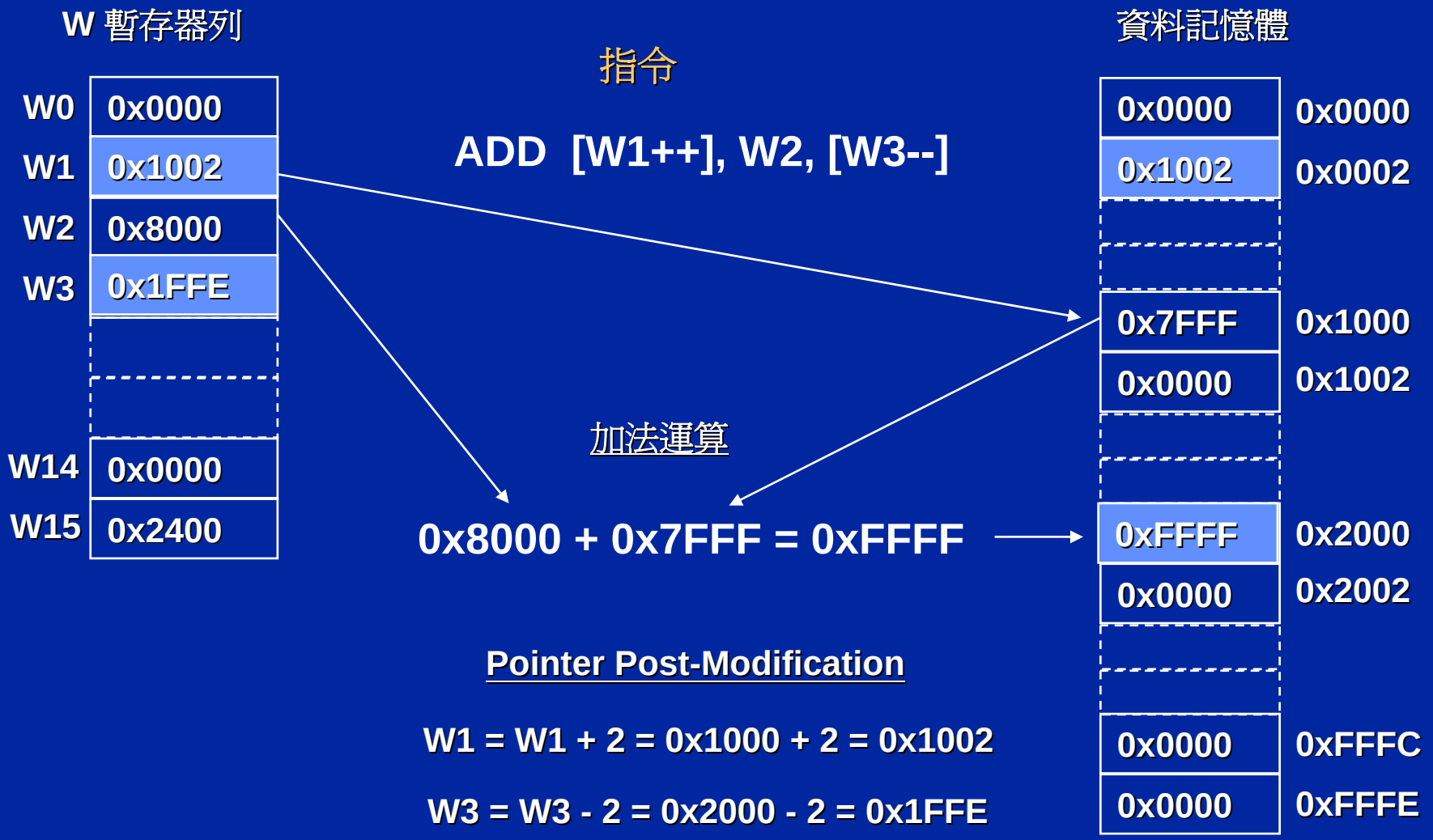
暫存器間接定址模式 - Byte 的存取



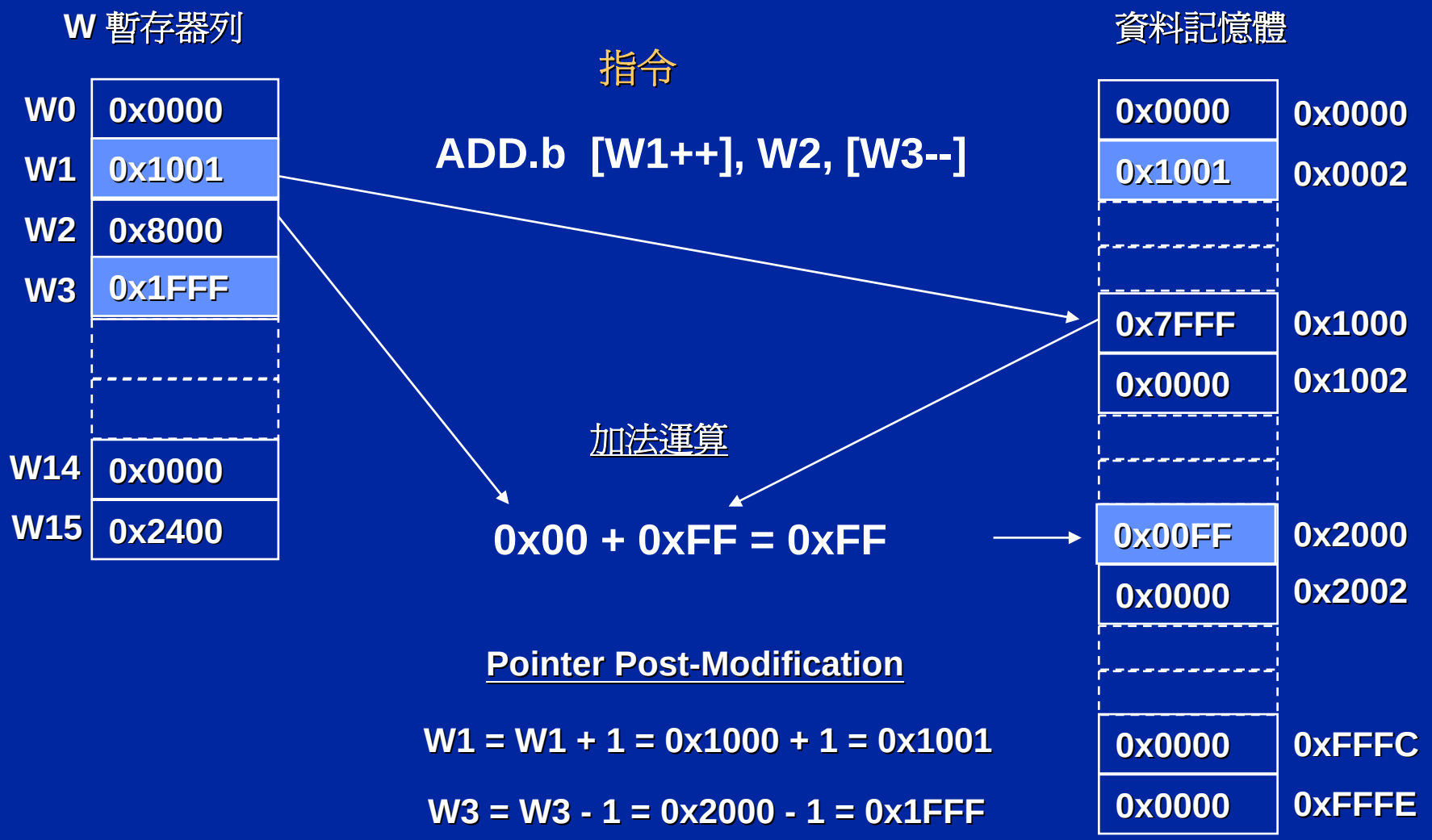
暫存器間接定址模式 – Post-Modification

- 指令先行處理資料後再進行有效位址 (EA) 的更新
  - EA = Wn 暫存器所指到的位址
  - 在語法上類似 C 型態的指標修改方式，例如：‘[W1++]’及 ‘[W2--]’
  - Post-modifies 修改指標
    - 在 byte 模式底下，指標會加 / 減 1
    - 在 word 模式底下，指標會加 / 減 2
  - Post-modification 可以使用於“來源”及“目地”暫存器

間接定址 Post-Modification: Word Access



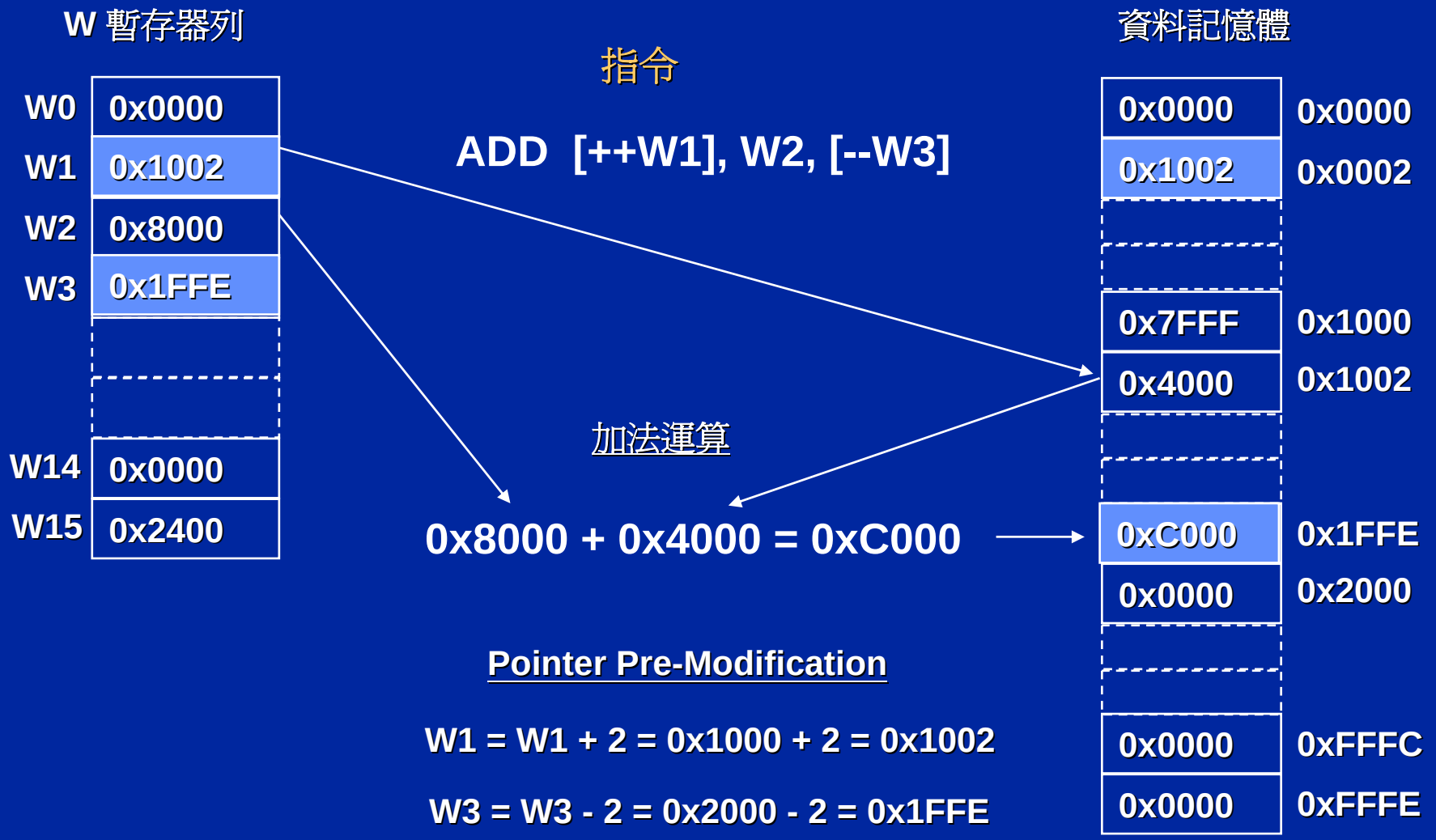
間接定址 Post-Modification: Byte Access



暫存器間接定址模式 – Pre-modification

- 指令先行更新有效位址 (EA)後再進行資料的處理
  - EA = Wn 暫存器所指到的位址
  - 在語法上類似 C 型態的指標修改方式，例如：‘[++W1]’ 及 ‘[--W2]’
  - Pre-modifies修改指標
    - 在 byte 模式底下，指標會加 / 減 1
    - 在 word 模式底下，指標會加 / 減 2
  - Pre-modification 可以使用於“來源”及“目地”暫存器

間接定址 Pre-Modification: Word Access



暫存器間接定址模式 – 暫存器偏移量

(ED, EDAC, LAC, MAC, MOV, MOVSAC, MPY, MPY.N MSC, PUSH, POP, SAC, SAC.R)

- 有效位址的指標爲兩個 W 暫存器之和
  - 內容值不可以改變

執行前	指令	執行後
W4 = 0x2000 W5 = 0x0004 W6 = 0x1000 [0x2004] = 0x000A [0x1000] = 0xEEEE [0x1002] = 0xFFFF	MOV [W4+W5], [W6++]	W4 = 0x2000 W5 = 0x0004 W6 = 0x1002 [0x2004] = 0x000A [0x1000] = 0x000A [0x1002] = 0xFFFF

暫存器間接定址模式 – 暫存器偏移量

W 暫存器列

W0	0x0000
W1	0x1000
W2	0x0020
W3	0x2002
W14	0x0000
W15	0x2400

指令

MOV [W1+W2], [W3++]

操作

有效位址計算 = 0x1000 + 0x20
複製位址 0x1020 的內容到位址 0x2000

Pointer Post-Modification

W1, W2 Unaffected

W3 = W3 + 2 = 0x2000 + 2 = 0x2002

資料記憶體

0x0000	0x0000
0x7FFF	0x1000
0xF354	0x1020
0xF354	0x2000
0x0000	0x2002
0x0000	0xFFFC
0x0000	0xFFFE

間接定址 – 暫存器偏移量 查表使用範例

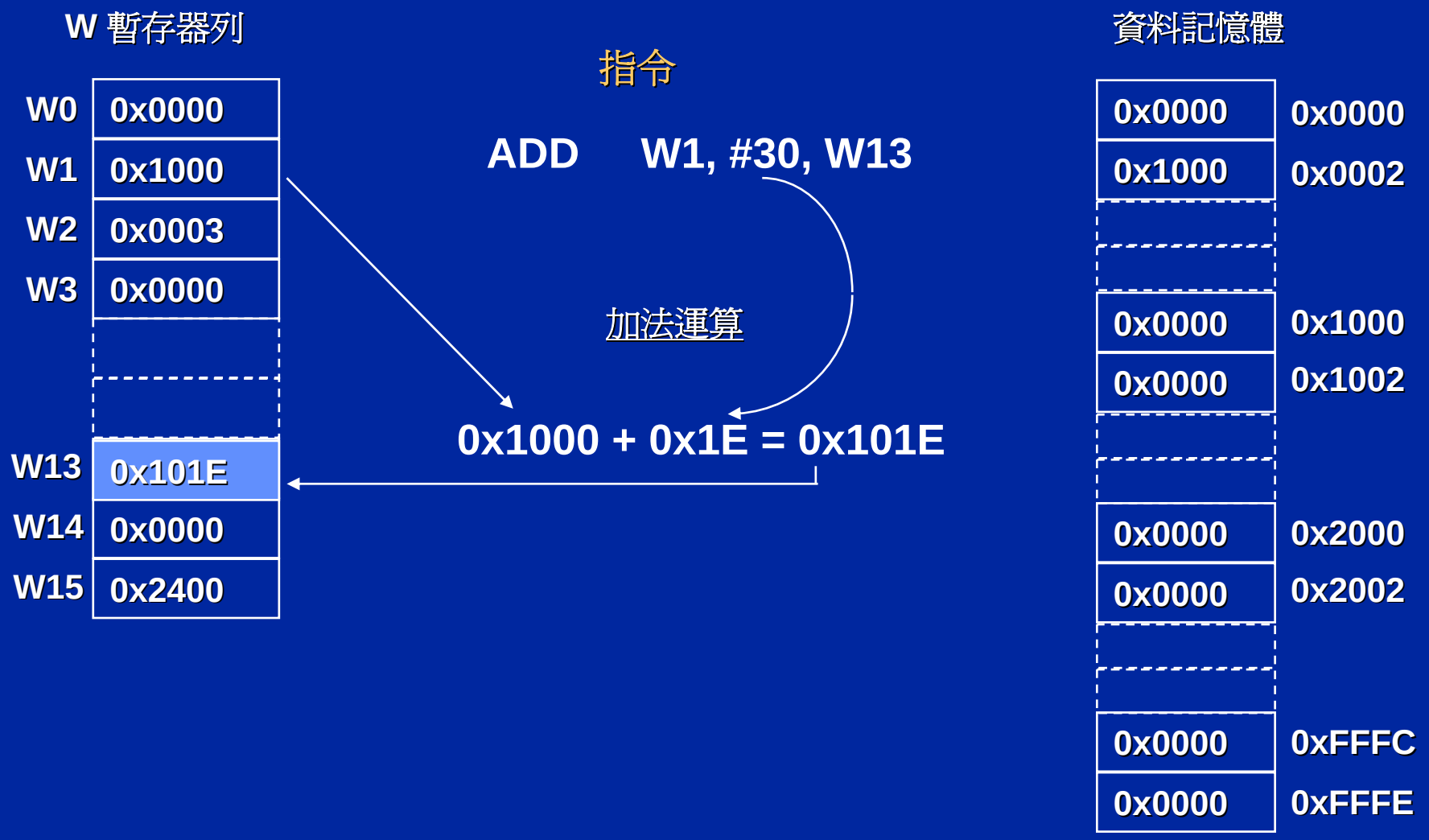
- 有效位址來自兩個 W 暫存器
  - 查表資料存放在資料記憶空間

```
                MOV    #MyTableBaseAdr, W0
                MOV    #Offset, W1
NextWordRead:   MOV    [W0 + W1], W2 ;Process contents of W2
                ADD    W1, #Offset, W1
                BRA    NextWordRead
```

立即定址模式

- 在指令裡指定一特定的數值以運算
  - 指定的立即數最多可以有 16 bits，主要還是取決於使用何種指令
    - MOV 指令可以有 16-bit 的立即數輸入
  - 有些指令可以接受 Byte 或 Word 型態的立即數
  - 立即數的操作可以支援眾多的指令

立即定址模式 – 範例



暫存器間接定址模式 – 立即偏移量值

- 有效位址的指標爲 W 暫存器和立即數之和
  - 只能用於 MOV 指令
  - 立即數的範圍：
    - 偶數值範圍從 [-1024, 1022]，使用於 **MOV** 指令於 word 模式下
    - 使用於 **MOV** 指令於 Byte 模式下，輸入的範圍從 [-512, 511]

指令功能集

- MCU 指令集
  - ✓ Move Instructions
  - ✓ Math and Logic Instructions
  - ✓ Bit Instructions
  - ✓ Stack Control Instructions
  - ✓ Program Flow Control Instructions
  - ✓ CPU Control Instructions
- DSP 指令集
  - ✓ MAC class of Instructions
  - ✓ Other Accumulator-based instructions

使用 MOV 指令群

- MOV 指令支援所有的定址模式
- 可以存取 64 KB 的範圍
- 一些範例如下：

```
MOV #0x1234,W4              ;Immediate operand
MOV W4,W6                   ;Register-direct
MOV 0x1F00,W4               ;File-register
MOV [W4++],[--W6]           ;Indirect with pre
                            ;and post-modifier
MOV [W4+W5],[W8]            ;Indirect with
                            ;Register Offset
MOV [W4+#768],[W8]          ;Indirect with
                            ;Literal offset
```

Math 和 Logic 指令群

- 暫存器 / 直接定址
- 暫存器直接定址
- 立即數的操作
- 間接定址

範例：

```
MUL.SU W2, #31, W2          ;W3:W2 = W2 * 31
AND W3, [W5++], [W8++]
ADD 0x900, WREG             ;W0 = W0+ Mem[0x900]
IOR #0x3FF, W4              ;W4 = W4 OR 0x03FF
REPEAT #17                  ;W0 = Quot.(W8/W2)
DIV.S W8, W2                ;W1 = Remder.(W8/W2)
```

位元操作指令

- 暫存器 / 直接定址
- 立即數的操作 (3 or 4-bit 立即數操作)
- 暫存器直接定址
- 間接定址

範例：

```
BSET.b    INTCON1H, #7    ;Sets bit 15 of INTCON1
BTG       INTCON1, #15    ;Toggle bit 15 of INTCON1
BCLR      [W1++], #9      ;Clears bit 9 of [W1]
BSET      W0, #4          ;Sets bit 4 of W0
BTSS      W0, #4          ;Test bit 4 of W0 and skip
                          ;next instruction if set
```

比較 / 跳躍 指令

- 暫存器 / 直接定址
- 立即數的操作
- 間接定址

範例：

```
CP 0x1200          ;Compare Mem[0x1200]
                   ;with W0
CP W2, #13         ;Compare W2 with 13
CPSGT W3, W5       ;If W3>W5 skip next
                   ;instruction
```

程式流程控制

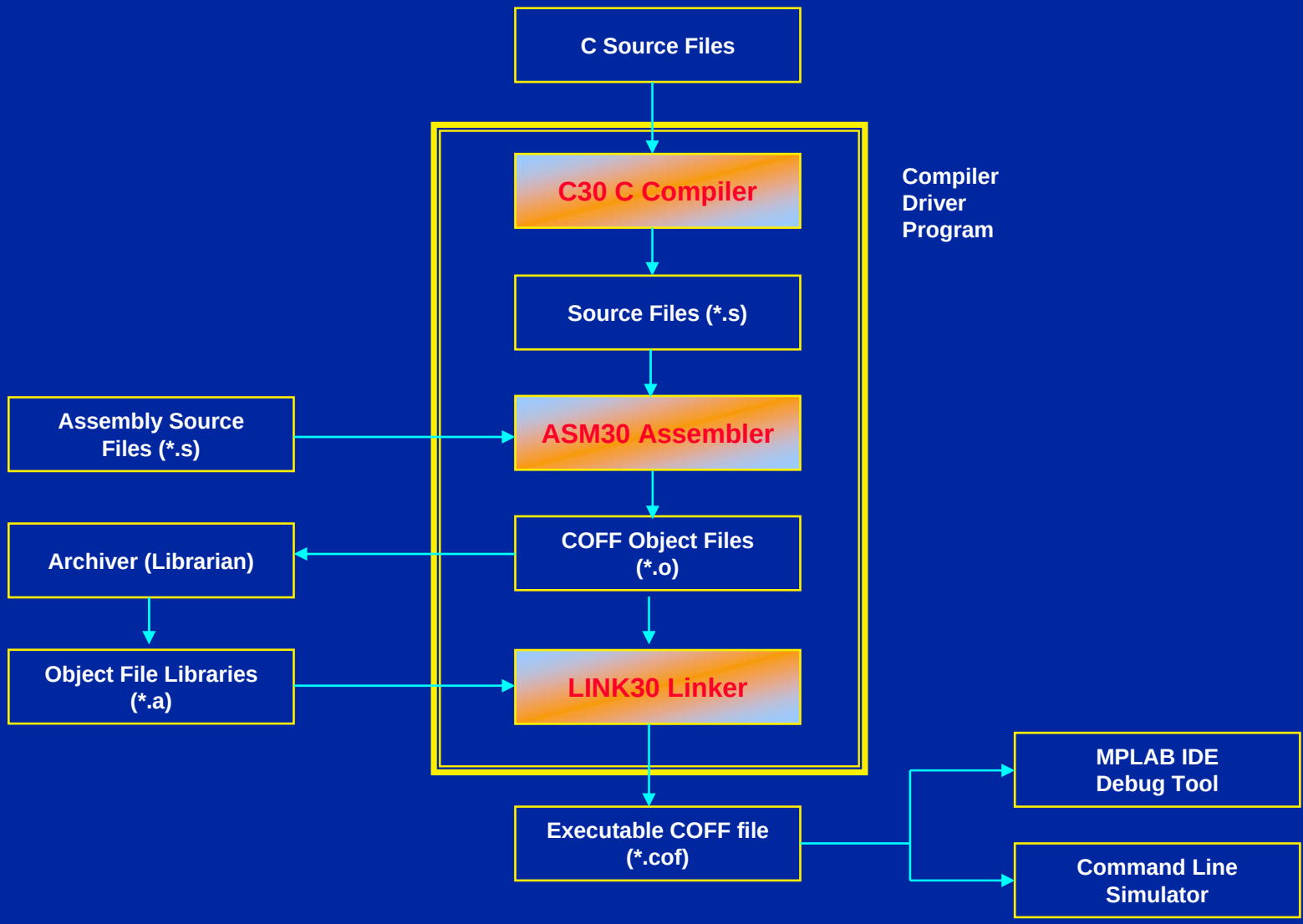
- 暫存器直接定址
- 立即數的操作

範例：

```
REPEAT W2              ;Repeat next instruction
                       ;W2+1 times
DO  #15, Label1        ;Execute a block of code
                       ;16 times
BRA W3                 ;PC<15:0> = W3, PC<23:16> = 0x00
GOTO W1                ;PC<15:0> = W1, PC<23:16> = 0x00
CALL W5                ;PC<15:0> = W5, PC<23:16> = 0x00
RCALL Subroutine1      ;1-word instruction
GOTO Label2            ;2-word instruction
CALL Subroutine2       ;2-word instruction
```

使用 dsPIC ASM30 & LINK30

組譯工具處理流程



ASM30 常用的虛指令 .include

- .include
  - 加入 source file 到程式裡
  - 組合語言的格式需加小數點，指到的是 *.inc 的檔案
    - 路徑需用“ ”的符號指定， MPASM 的語法 < > 在此不適用
    - .include “c:\pic30_tools\support\inc\p30f6014.inc“
  - 注意程式的路徑，不可指錯路徑
  - C 語言需加#，指到的是 *.h 的檔案格式
    - #include “p30f6014.h”

ASM30 常用的虛指令 .equ .equiv .set

- 定義常數的名稱
- 一般習慣上常使用大寫定義常數
- 語法上與 MPASM 有所不同
  - dsPIC 語法
    - .equ CORCONH , 0x45
  - MPASM 語法
    - CORCONH equ 0x45
- .equ 可重複定義常數名稱，以上一定義的名稱爲參考
- .equiv 不可重複定義常數名稱，重複組譯時會產生錯誤
- .set 使用上與 .equ 一樣

ASM30 常用的虛指令
節區名稱 .section

- 節區名稱宣告語法： **.section** *name* [,"Flags]
- 最基本的節區名稱有三種：
  - .bss : 未設定初始值的變數區域 (Uninitialized Data)
  - .data : 已設定初始值的變數區域 (Initialized Data)
  - .text : 程式區域 (Executable Code)
- 屬性 "Flags" 則有五種
  - "b" : bss section (未指定初始值變數)
  - "n" : Section is not load
  - "d" : Data section (指定初始值變數)
  - "r" : Read-Only data section (PSV window)
  - "x" : Execttable section

ASM30 常用的虛指令
節區名稱 .section

- 當節區名稱被宣告時，內定的屬性旗號會跟隨著所屬的宣告

Section Name	Default Flag
.bss	“b”
.data	“d”
.text	“x”

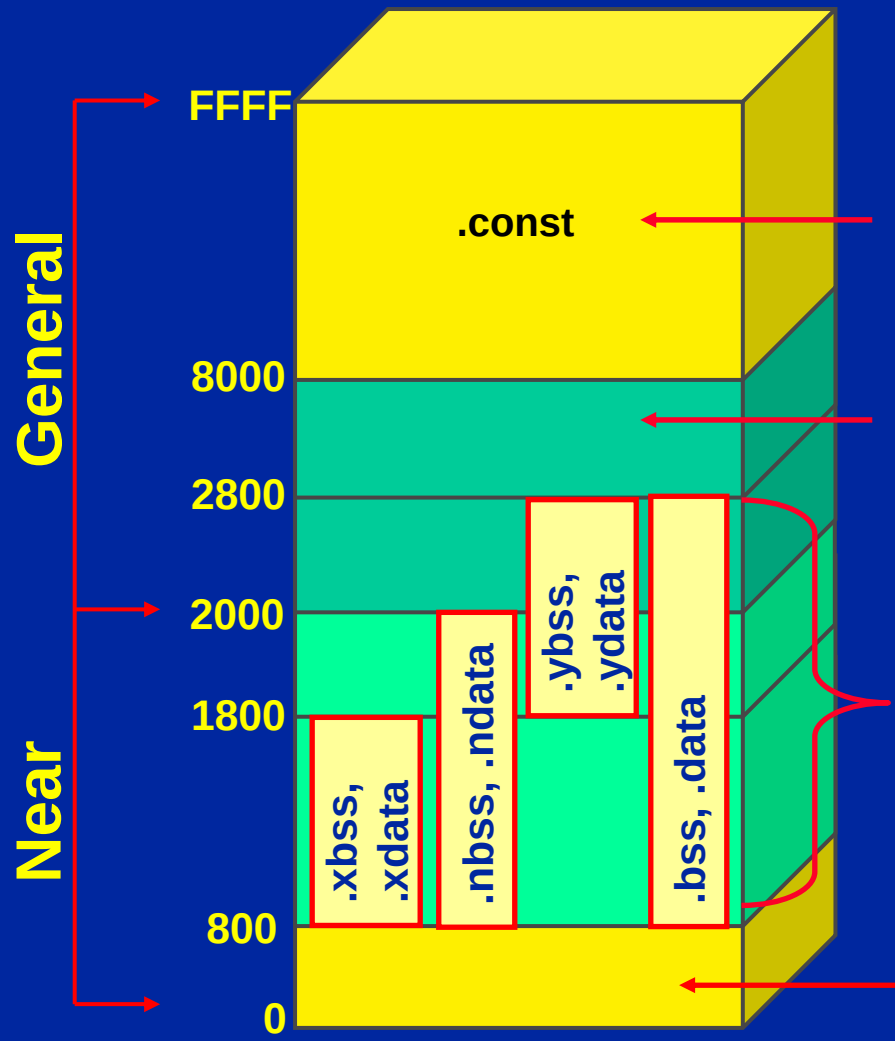
範例：
將下列 **Var1 & Var2** 的變數定義到 **.bss (uninitialized data)** 的節區

```
        .section .bss, "b"
Var1:   .space 4   ; 保留 4 個 bytes
Var2:   .space 1   ; 保留 1 個 byte
```

資料節區名稱

dsPIC Section Name	Description	Section Flag
.bss	General Memory is not uninitialized	“b”
.nbss	Near Memory (8kB) is not uninitialized	“b”
.xbss	X Memory is not uninitialized	“b”
.ybss	Y Memory is not uninitialized	“b”
.pbss	Persistent Data Memory	“b”
.data	General Memory (initial values)	“d”
.ndata	Near Memory (initial values)	“d”
.xdata	X Memory (initial values)	“d”
.ydata	Y Memory (initial values)	“d”
.dconst	Constants in the General Memory	“d”
.ndconst	Constants in the Near Memory	“d”

資料記憶空間分配



Program Space Visibility Data Window (PSV)

目前未使用區域

.bss, .nbss, .xbss, .ybss:
無初始值設定的變數區, 堆疊區

.data, .ndata, .xdata, .ydata:
有初始值設定的變數區

(SFR) 特殊暫存器空間

使用 .bss 定義變數位址

- 語法：.section　.bss，“b”
- 宣告以下的資料爲 Uninitialized Data，並放置在RAM (0x800) 以後的位址。
- 位址的排定是在 *.gld 宣告，由Link30安排

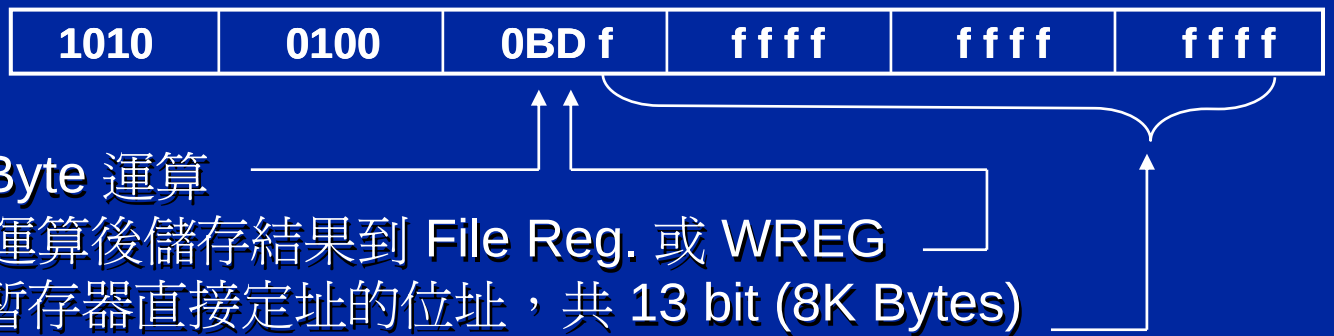
範例：

```
;.....................................................................
;          Uninitialized variables in general data memory
;.....................................................................

                .section .bss, "b"
                .align      2
A_Var:          .space 4                    ; 保留 4 個 Bytes 的變數給 A_Var
B_Var:          .space 20                   ; 保留 20 個 Bytes 的變數給 B_Var
C_Var:          .space 4                    ; 保留 4 個 Bytes 的變數給 C_Var
```

暫存器定址模式 .nbss

- 基本上在 FR 定址模式下，與 WREG 的相互操作，其暫存器的視野只有 8K Bytes。
  - MOV 0x1000,WREG
  - ADD.B 0x17FF,WREG



- 但在 FR 定址模式下，與 Wn 的相互操作，其暫存器的視野有 32K Words。
  - MOV 0xF000,W0 ；注意：一定要使用偶數位址，word 型態

定義變數位址在 Near Data .nbss

- 語法：.section　.nbss，“b”
- 宣告以下的資料爲 Uninitialized Data，並放置在RAM (0x800 – 0x1FFF) 的區間。

範例：

```
;.....................................................................
;          Uninitialized variables in near data memory
;.....................................................................

                .section .nbss, "b"
                .align      2
A_Var:          .space 4                    ; 保留 4 個 Bytes 的變數給 A_Var
B_Var:          .space 20                   ; 保留 20 個 Bytes 的變數給 B_Var
C_Var:          .space 4                    ; 保留 4 個 Bytes 的變數給 C_Var
```

定義變數位址 – X Space .xbss

- 語法：.section .xbss , “b”
- 宣告以下的資料爲 Uninitialized Data，並放置在X Data RAM 的位址。
- 儲存一些 DSP 運算的資料 (DSP 指令可以同時存取 X 及 Y Space)，例如：A/D 取樣，FIR 、IIR 的係數，Modulo Buffer …

範例：

```
                .equ        SAMPLES, 64          ; A/D number of samples
;.....................................................................
;Uninitialized variables in X-space in data memory
;.....................................................................

                .section    .xbss, "b"
x_input:        .space      4*SAMPLES            ;Allocating space (in bytes) to variable.
```

定義變數位址 – Y Space .ybss

- 語法：.section　.ybss，“b”
- 宣告以下的資料爲 Uninitialized Data，並放置在Y Data RAM 的位址。
- 僅適用於 DSP 指令，可讓 dsPIC 同時提取 X-Space 及 Y-Space

範例：

```
            .equ        SAMPLES, 64         ; A/D number of samples
;.....................................................................
;Uninitialized variables in Y-space in data memory
;.....................................................................

            .section    .ybss, "b"
y_input:    .space      4*SAMPLES           ;Allocating space (in bytes) to variable.
```

保留記憶體給變數 .space

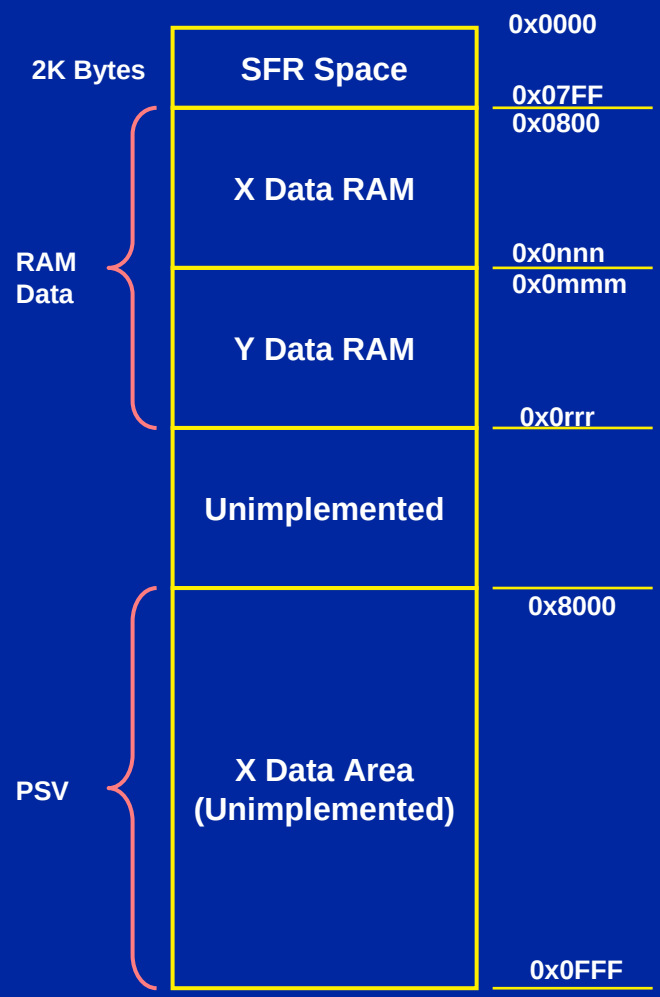
- 語法：.space *size* [,fill]
  - 保留“size”數量的 Bytes 給變數，如有給特定值 [fill]則會被填入此保留空間

範例:

```
                .equ SAMPLES, 64                ;Number of samples
;
                .section    .xbss, "b"
x_input:        .space      2*SAMPLES           ;Allocating space (in bytes) to variable.
;.....................................................................
;Uninitialized variables in Y-space in data memory
;.....................................................................
                .section    .ybss, "b"
y_input:        .space      2*SAMPLES
;.....................................................................
;Uninitialized variables in Near data memory (Lower 8Kb of RAM)
;.....................................................................
                .section    .nbss, "b"
Var1:           .space 4                        ; 2 words of space for variable "var1".
Var2:           .space 2
```

RAM 資料的擺放

- dsPIC SFR 暫存器的位址是固定從 0x0000 ~ 0x07FF
- RAM 資料會從 0x0800 的位址開始擺放
- 注意：RAM 的大小會依所使用的 dsPIC 元件而有所不同，X Data & Y Data RAM Size 也會隨著所使用的元件而變動
- PSV 共32K Bytes，此位址是固定不變的



RAM 資料的擺放順序

1. .xbss -- 從 0x0800 開始
2. .nbss -- 緊跟著 .xbss 後面
3. .bss -- 緊跟著 .nbss 後面
4. .ybss -- 0x1800 (dsPIC30F6014)
   -- 0x0900 (dsPIC30F2010)

.bss 範例

```
                .section .bss, "b"
                .align      2
A_Var:          .space 4
B_Var:          .space 20
C_Var:          .space 4

                .section .bss, "b"
                .align      2
D_Var:          .space 4
E_Var:          .space 2
F_Var:          .space 2
;.....................................................................
;
                .section .xbss, "b"
x_input:        .space 4*SAMPLES        ;Allocating space (in bytes) to variable.
;.....................................................................
;
.section        .ybss, "b"
y_input:        .space 2*SAMPLES
;.....................................................................
;
                .section .nbss, "b"
var1:           .space 2                ;Example of allocating 1 word of space for variable "var1"
```

觀察變數位址配置
dsPIC30F6014

Data Memory Usage

section	address	alignment gaps	total length	(dec)
.xbss	0x800	0	0x100	(256)
.nbss	0x900	0	0x2	(2)
.bss	0x902	0	0x24	(36)
.ybss	0x1800	0	0x80	(128)
Total data memory used (bytes):			0x1a6	(422)

Dynamic Memory Usage

region	address	maximum length	(dec)
heap	0x1880	0	(0)
stack	0x1880	0xf18	(3864)
Maximum dynamic memory (bytes):		0xf18	(3864)

觀察變數位址配置
dsPIC30F2010

Data Memory Usage

section	address	alignment gaps	total length	(dec)
.xbss	0x800	0	0x100	(256)
.nbss	0x900	0	0x2	(2)
.bss	0x902	0	0x24	(36)
.ybss	0x926	0	0x80	(128)
Total data memory used (bytes):			0x1a6	(422)

Dynamic Memory Usage

region	address	maximum length	(dec)
heap	0x9a6	0	(0)
stack	0x9a6	0x52	(82)
Maximum dynamic memory (bytes):		0x52	(82)

Stack Request : .ybss (0x926) + Length (0x80) = 0x0926

程式的起始動作
Program Start-Up

- dsPIC 的程式起始動作有兩種
  - 不需設定初始變數的啓動方式
  - 需設定初始變數的啓動方式
- 套用範例程式
  - 路徑　C:\Program Files\Microchip\MPLAB ASM30 Suite\Support\templates \assembly
  - tmp6010.s & tmp6014.s -- 不設定初始變數
  - Tmp6010_srt.s & tmp6014_srt.s -- 設定初始變數

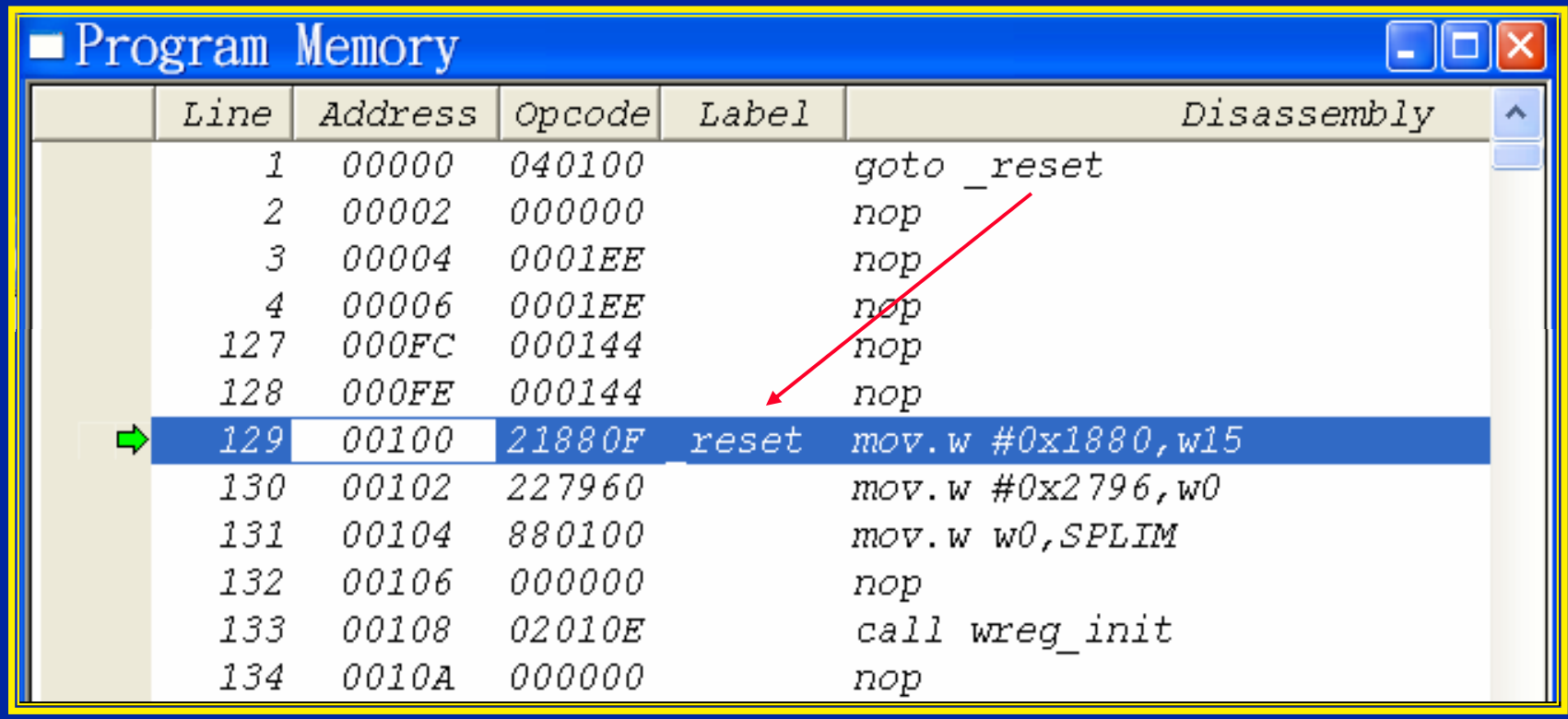
套用 tmp6010.s & tmp6014.s

- 單純使用在 .bss，.nbss，.xbss，.ybss 的宣告
- 沒有使用到 “啓動模組”，dsPIC reset 後直接將控制權交給 “ _ _ reset: “ 的標記 (Label)
- Reset Vector 0x000000 自動填入 goto _reset
- 如果沒有特殊的指定 _reset: 的執行位址會被編排到 0x000100

```
.text                       ;Start of Code section
__reset:
        MOV #__SP_init, W15       ;Initalize the Stack Pointer
        MOV #__SPLIM_init, W0     ;Initialize the Stack Pointer Limit Register
        MOV W0, SPLIM
```

程式的位址安排
未使用啓動模組

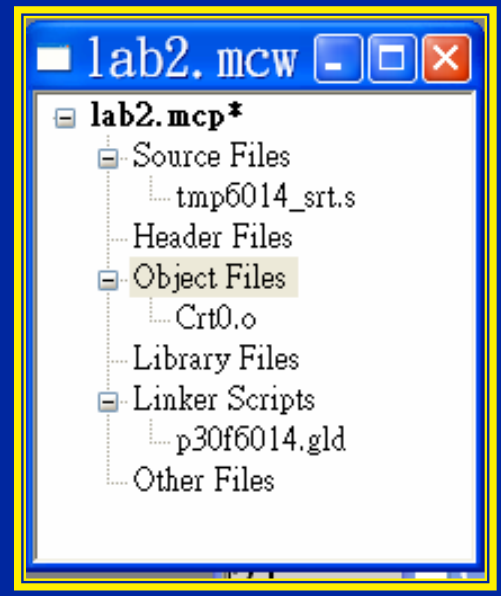
- 由下列的反組譯 (disassembly) 的顯示即可清楚的知道程式的擺放 (以上一頁程式爲例)



使用啓動模組 crt0.o 或 crt1.o

- 參考套用的範例程式的書寫格式
  - \dsPIC_Tools\support\templates\assembly\tmp6014_srt.s
- Project 的設定裡，加入 Object File “Crt0.o” 如下圖所示
- 內定是使用 crt0.o，如果沒有 Initialized Data 則可使用 Crt1.0

1. **Reset 後，控制權會先交給 Crt0.o 裡的 _reset**
2. **設定堆疊指標 (W15 & SPLIM)**
3. **設定 PSVPAG 及 CORCON (.const)**
4. **處理初始值的設定**
5. **將控制權交給 User’s 程式裡的 main 標記**



套用 tmp6010_srt.s & tmp6014_srt.s

```
            .equ __30F6014, 1
            .include   "\pic30_tools\support\inc\p30f6014.inc“
;.....................................................................
;Initialized variables in X,Y-space in data memory
;.....................................................................
            .section   .xdata, "d"
            .align 32                ;Aligns the next word to be stored  to a multiple of 32
x_in:       .hword     0x1111, 0x2222, 0x3333, 0x4444, 0x5555
;
            .section   .ydata, "d"
y_in:       .hword 0x1234, 0x5678, 0x9abc, 0xdef0, 0xabab
;
.text                                ;Start of Code section
_main:
            CALL _wreg_init          ;Call _wreg_init subroutine
```

程式的位址安排
使用啓動模組 - - crt0.o

Reset Vector

_reset

Main

Program Memory

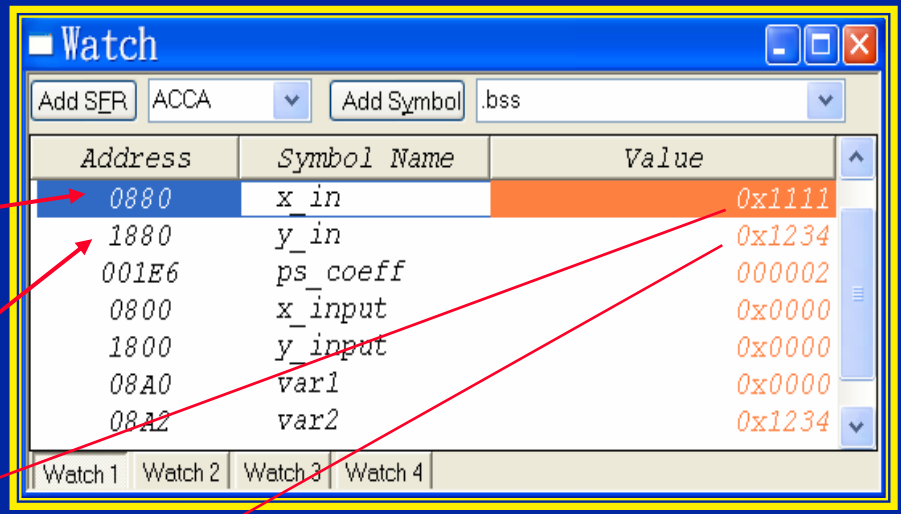
Line	Address	Opcode	Label	Disassembly
1	00000	040100		goto _reset
2	00002	000000		nop
3	00004	0001EE		nop
128	000FE	0001EE		nop
129	00100	2188AF	_reset	mov.w #0x188a,w15
130	00102	227960		mov.w #0x2796,w0
131	00104	880100		mov.w w0,SPLIM
132	00106	000000		nop
133	00108	070005		rcall _psv_init
134	0010A	07000C		rcall _data_init
135	0010C	020180		call main
136	0010E	000000		nop
193	00180	020186	main	call wreg_init
194	00182	000000		nop
195	00184	37FFFF	done	bra done
196	00186	EB0000	wreg_ini	clr.w w0
197	00188	780700		mov.w w0,w14
198	0018A	09000C		repeat #12
199	0018C	782F00		mov.w w0,[++w14]
200	0018E	EB0700		clr.w w14
201	00190	060000		return

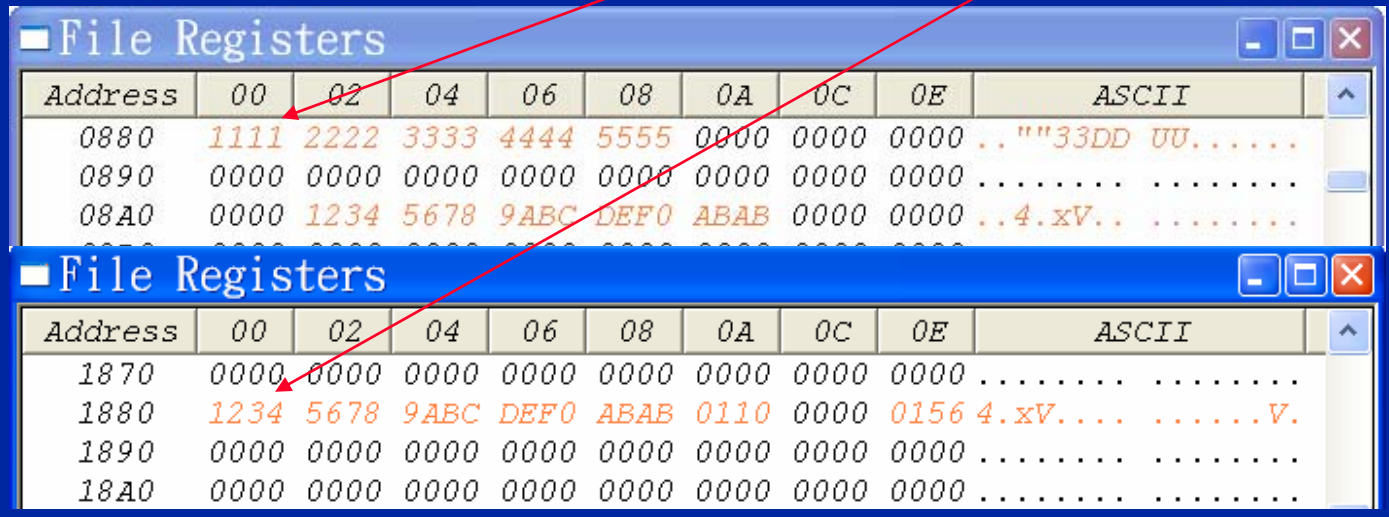
初始設定的資料安排
crt0.o

Data Memory Usage

section	address	total length (dec)
.xbss	0x800	0x80 (128)
.xdata	0x880	0x20 (32)
.nbss	0x8a0	0x2 (2)
.ndata	0x8a2	0xa (10)
.ybss	0x1800	0x80 (128)
.ydata	0x1880	0xa (10)

Total data memory used (bytes): 0x136 (310)

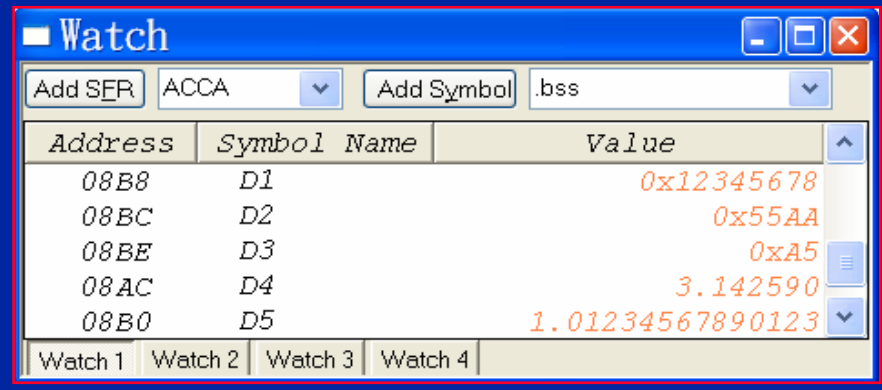




Define Initialized Data .data & .ndata

- .section .data , “d”
  - Define the Initialized Data in the RAM area

```
            .section    .data , “d”
D1:         .long       0x12345678      ; 4 bytes
D2:         .word       0x55AA          ; 2 bytes
D3:         .byte       0xA5
```



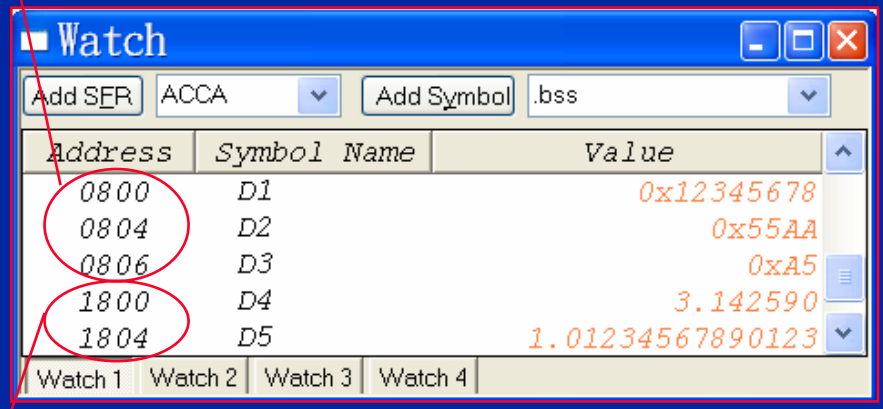
- .section .ndata , “d”
  - Initialized Data in near RAM area

```
            .section    .ndata,"d"
D4:         .float      3.14259             ; 4 bytes
D5:         .double     1.012345678901 23   ; 8 bytes
```

Define Initialized Data .xdata & .ydata

- .section .xdata , “d”
  - Define the Initialized Data in the X Data RAM

```
                .section    .xdata , “d”
D1:             .long       0x12345678      ; 4 bytes
D2:             .word       0x55AA          ; 2 bytes
D3:             .byte       0xA5
```



- .section .ydata , “d”
  - Initialized Data in the Y Data RAM

```
                .section    .ndata,"d"
D4:             .float      3.14259         ; 4 bytes
D5:             .double     1.01234567890123 ; 8 bytes
```

數字的進制語法

- 二進制
  - 0b01011010 , 0B01011010
- 十進制
  - 01234 , 05000
- 十六進制
  - 0x55AA , 0X00FF
- Floating
  - IEEE-754 format
- Fixed-Point Number
  - Q15 format

資料格式設定

- .byte – 8-bit data format
- .word – 16-bit data format
- .hword – 16-bit data format
- .long – 32-bit data format
- .int -- 32-bit data format
- .fixed – 16-bit Q15 data format
- .float – 32-bit IEEE -754 float format
- .single – 32-bit IEEEE-754 float format
- .double – 64-bit IEEE-754 float format
- .ascii – 填入字串，不自動補 null byte (0x00)
- .assiz --填入字串，自動補 null byte (0x00)

標記 (Label)

- 標記必須以冒號“:“ 做爲結束
  - 標記可以使用英文字母、數字、“_” 及“.“
  - 標記必須在第一列

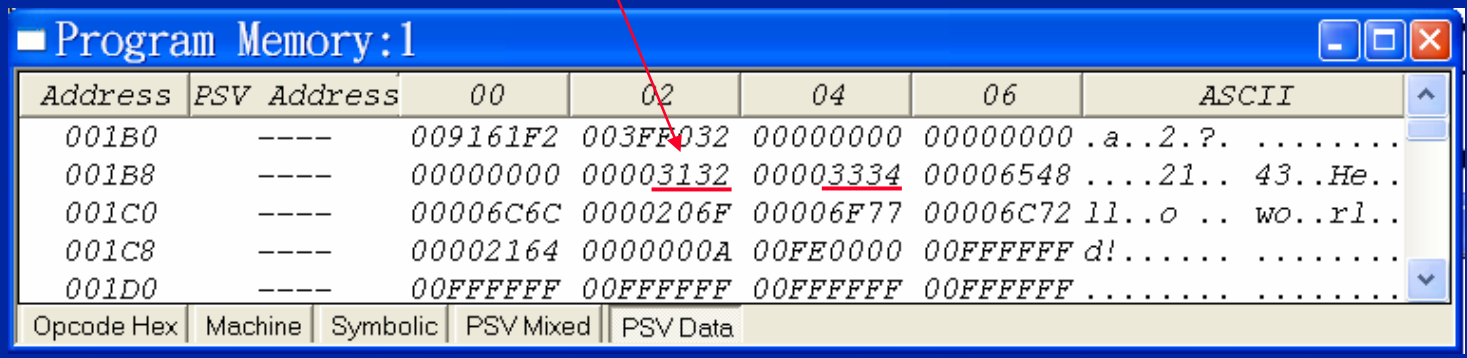
- 區域標記 (Local Label)
  - 定義近程的區域跳躍標記
  - 共10個標記可供使用
    - ✓ 0 ~ 9
  - 往前跳躍要加 b
  - 往後跳躍則要加 f

```
print_string:
            mov w0,w1
1:
            cp0.b [w1]
            bra z,9f
            mov.b [w1++],w0
            call print_char
            bra 1b
9:
            return
```

定義 ROM Data

- .section .const , “r”
  - 定義常數資料到 Program Memory

```
        .section .const, "r"
; The following symbols (C1 and C2) will be placed
; in the named section ".const".
C1:     .word   0x3132
C2:     .word   0x3334
hello:  .ascii "Hello world!\n\0"
```



自 Program Memory 存取資料

操作指令	說明
tblpage (name)	Get page for Table Read/write operations
tbloffset (name)	Get pointer for Table Read/write operations
psvpage (name)	Get page from PSV data window operations
psvoffset (name)	Get pointer from PSV data window operations
paddr (label)	Get 24-bit address of *label* in Program Memory
handle (label)	Get 16-bit reference of *label* in Program Memory
.sizeof. (name)	Get size of section *name* in address units
.startof. (name)	Get starting address of section *name*

Table Read/Write

- Table Read 指令
  - TBLRDL.B , TBLRDL.W , TBLRDH.B , TBLRDH.W
  - Table 指令是使用 [Wn] 來索引定址，故其視野範圍只有 16-bit
  - 利用 Page 的觀念擴展視野到 24-bit 的範圍
  - 利用 #tblpage (label name) 設定 TBLPAG 暫存器
  - #tbloffset (label name) 設定 [Wn] 的 64KW 的位址

Table Read 範例

```
;-------------------------------------------------------------------------------
;Tone table is placed as a loopup table in program memory

        .section .const,"r"
        .palign 4
ToneTable:
        .hword   0x1370,0x1398,0x13B0,0x13C6,0x13D9,0x13E9,0x13F5,0x13FC
        .hword   0x13FF,0x13FC,0x13F5,0x13E9,0x13D9,0x13C6,0x13B0,0x1398
        .hword   0x137F,0x1366,0x134E,0x1338,0x1325,0x1315,0x1309,0x1302
        .hword   0x1300,0x1302,0x1309,0x1315,0x1325,0x1338,0x134E,0x1356;
;
        .section  .text, “x”
;
        :
        :
        :
;
        mov      #tblpage(ToneTable),W0      ;Get upper address (page)
        mov      W0,TBLPAG                   ;Load address into PSVPAG
        mov      #tbloffset(ToneTable),W0    ;Get lower address (offset) of text
                                             ;in program memory
        tblrdl   [W0++],W1                   ; Get ToneTable into the W1
```

使用 PSV 功能

- Program Space Visibility
  - 需要將 PSV bit = 1 (CORCON<2>)
  - 被指到的 Program Memory (16 K Word) 會映對 RAM 在 0x8000 – 0xFFFF 共 32K Byte 的位置
  - PSV 模式下，Program Memory 24-bit 的資料中，只有低16-bit 的資料會被映對，8-bit 的 MSB byte 是無法被映對到 RAM 的
    - 只有 Table 指令才可以讀到最高的 8-bit 資料
  - PSV 常用讀取連續的 EEPROM 資料，FIR , IIR 的係數，常與 REPEAT , DO 指令合用

PSV 設定範例

```
        .equ    PSV, 2
;-------------------------------------------------------------------------
;Tone table is placed as a loopup table in program memory

        .section .const,"r"
        .palign 4
ToneTable:
        .hword  0x1370,0x1398,0x13B0,0x13C6,0x13D9,0x13E9,0x13F5,0x13FC
        .hword  0x13FF,0x13FC,0x13F5,0x13E9,0x13D9,0x13C6,0x13B0,0x1398
        .hword  0x137F,0x1366,0x134E,0x1338,0x1325,0x1315,0x1309,0x1302
        .hword  0x1300,0x1302,0x1309,0x1315,0x1325,0x1338,0x134E,0x1356;
;
        .section  .text, “x”
;
        :
        :
        bset.b  CORCON,#PSV                 ; Enable PSV function
;
        mov     #psvpage(ToneTable),W0      ;Get upper address (page)
        mov     W0,PSVPAG                   ;Load address into PSVPAG
        mov     #psvoffset(ToneTable),W0    ;Get lower address (offset) of text
                                            ;in program memory
```

堆疊的設定

- 堆疊的大小會隨著所使用變數的多寡自動調整
- 堆疊的設定
  - 使用 crt0.o 的啓動模組 – 自動設定 W15 及 W14
  - 沒有使用啓動模組 – 需自行設定 W15 及 W14

自行設定堆疊的範例：

```
                .section   .text , “x”              ;Start of Code section

__reset:
                mov        #__SP_init, W15         ;Initalize the Stack Pointer
                mov        #__SPLIM_init, W0
                mov        W0,SPLIM                ;Initialize the Stack Pointer Limit Register
                nop                                ;Add NOP to follow SPLIM initialization
```

中斷功能

中斷基本功能

- dsPIC30F 中斷功能
  - 每一個中斷源有自己獨立的中斷向量表 (IVT)
    - 8 個非遮罩式中斷向量 (Non-Maskable)
    - 54 個遮罩式中斷向量
  - 向量表內的內容是中斷副程式進入點的位址
  - 每一格中斷源使用著可以設定 7 種不同的中斷優先權
  - 第二組中斷向量表 (IVT)，可用於除錯功能
  - 固定 5 個指令周期的中斷響應時間
  - 固定 3 個指令周期的中斷返回時間

中斷向量表 (IVT)

中斷的優先權 (一)

- CPU 有 16 種中斷優先權的設定，級數愈高等級愈高
- 等級 8 - 15 保留給 trap 中斷使用
- 狀態旗號的 IPL 位元指出目前 CPU 的中斷優先權設定等級
  - IPL<3> bit (CORCON<3>)
  - IPL<2:0> bits (SRL<7:5)
- 使用著可設定每個中斷源 0~7 等級的中斷優先權
- 藉由設定 IPC 暫存器可以改變各個中斷源的優先權等級
- 中斷等級 = 0，該中斷源禁能

中斷的優先權 (二)

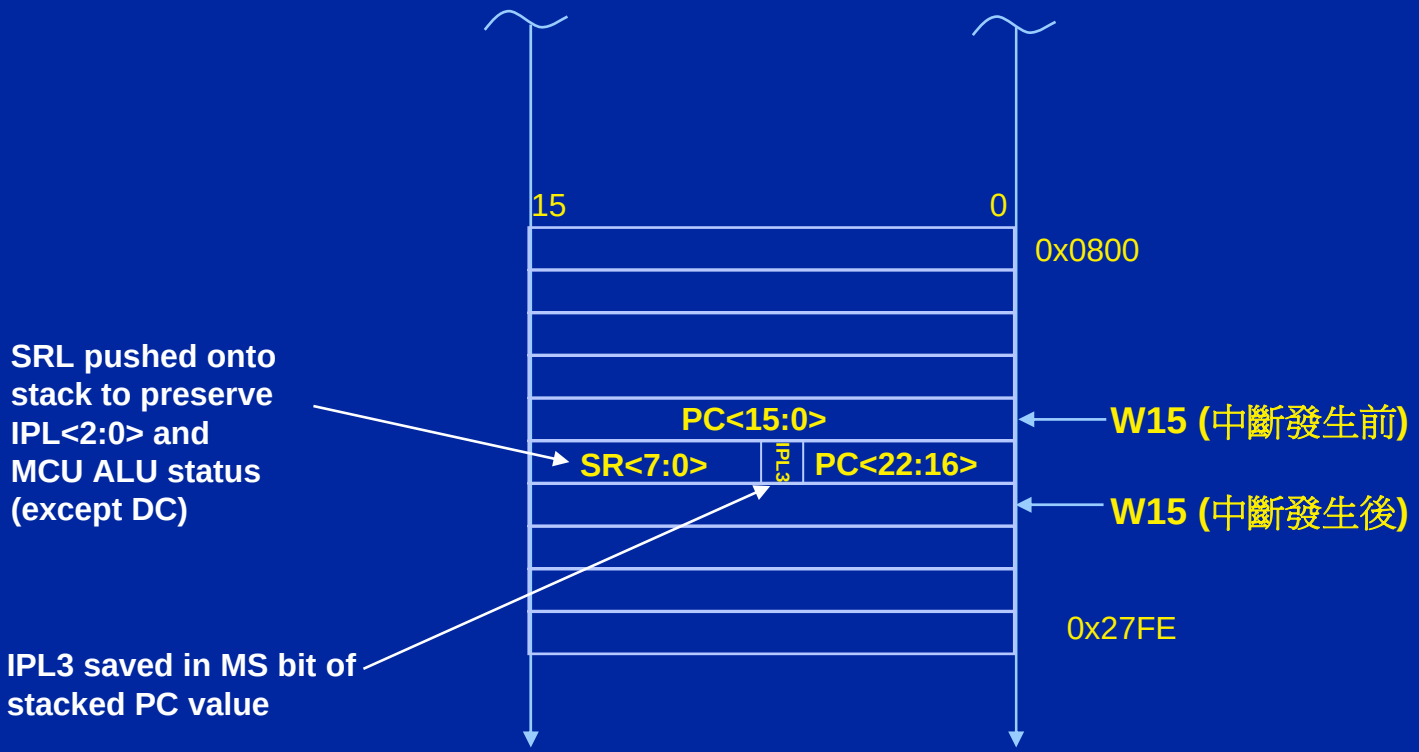
- 中斷源的優先權等級必須大於 CPU 的等級設定 (IPL<3:0>) 才有能力中斷 CPU
- IPL<2:0> 可以用軟體設定以變更 CPU 優先權
- IPL<3> 僅能被軟體讀取，它無法被禁能
- IPL 位元在中斷程式裡是可以被變更的
- 中斷發生時，原先 IPL<3:0> 的值會被推入軟體堆疊裡
- 中斷向量表有內定的先後順序設定以解決中斷相互衝突事件
- 內定的先後順序設定可以重新設定其優先權

中斷的 Context 儲存 / 取回

- 中斷的內容儲存
  - SRL 及 PC 會自動儲存到堆疊裡
  - 可以使用 PUSH(.D) 和 POP(.D) 指令存取暫存器
  - 隨時可以使用 PUSH.S 和 POP.S 指令
    - 允許快速的將 W0...W3 和 MCU 的狀態旗號 (DC, N, OV, Z, C) 內容儲存
    - 僅提供一層的 shadow registers

中斷的堆疊

- 堆疊的動作會發生在進入中斷副程式之前



開始動手做實驗！

General Notes for labs

- For all labs we will utilize supplied projects
- Refer to the supplemental hand-outs for assistance.
  - 1 supplemental hand-out per each lab
- Lab solutions are provided in “solutions” folder with MPLAB® project and workspace files
- No right or wrong here, just write the code!
- Ask questions!!

Hands-on Lab #1

Code Requirements - Lab 1, Part 1

- Configure PORTD pins as outputs
- Configure INT1 - INT4 pins for interrupts
  - Clear respective interrupt flag
  - Enable respective interrupts
- Define multiple Interrupt Service Routines
  - INT1 - INT4 pin Interrupts
  - Clear interrupt flag
  - Return from Interrupt
- Provide global scope for 2 Interrupts
- In all there are 22 lines of code to write

Objectives - Lab 1, Part 1

- Understand Natural Order Priority Behavior
- Understand Interrupt Nesting Behavior
  - Nesting is enabled on device Reset
- How to identify IVT names from Linker script
- Implement several instruction types:
  - Logic Instructions
  - Bit Instructions
  - Program Flow Instructions

Notes - Lab 1, Part 1

- Upon device running:
  - Press and release switch S1 and observe LED1
  - Press and hold switch S4 and then press S1
    - ✓ What happens?
    - ✓ What should happen?
- Natural Priority Level
  - INT1 ⇒ Vector 24, (S1 on board maps to INT1)
  - INT2 ⇒ Vector 31, (S2 on board maps to INT2)
  - INT3 ⇒ Vector 44, (S3 on board maps to INT3)
  - INT4 ⇒ Vector 45, (S4 on board maps to INT4)

Code Requirements - Lab 1, Part 2

- Configure Peripheral Interrupt Priority Levels
  - Interrupt Priority Control Registers
    - ✓ IPC4[2:0] ⇒ INT1 (set to level 4, done for you)
    - ✓ IPC5[14:12] ⇒ INT2 (set to level 5)
    - ✓ IPC9[2:0] ⇒ INT3 (set to level 6)
    - ✓ IPC9[6:4] ⇒ INT4 (set to level 7)

- In all there are 9 lines of code to write

Objectives - Lab 1, Part 2

- Understand interrupt behavior when peripheral priority levels are modified
- Natural order of interrupts are not affected

- Implement more BIT instruction types

Notes - Lab 1, Part 2

- Upon device running:
  - Press and release switch S1 and observe LED1
  - Press and hold switch S4 and then press S1
    - What happens?
    - What should happen?
    - What changed for Lab1, Part1 ?
- Observe Interrupt nesting behavior
  - LED behavior should be different from Part 1 when all peripherals had the same priority level

Code Requirements - Lab 1, Part 3

- Implement Alternate Interrupt Vector Table
  - ALTIVT bit ⇒ INTCON2[15]
- Define multiple Alternate Interrupt Service Routines
  - INT1 - INT4 pin Interrupts
  - Clear interrupt flag
- In all there are 9 lines of code to write

Objectives - Lab 1, Part 3

- Understand Alternate Interrupt Handler Behavior
- How to identify ISR names from Linker script
- Implement more BIT instruction types

Notes - Lab 1, Part 3

- Upon device running:
  - Press and release switch S1 and observe LED1
    - Press and hold switch S4 and then press S1
    - What happens?
    - What should happen?
- Notice LED1 - LED4 behavior is opposite of LAB1, Part 1
- Alternate interrupts useful for creating dual ISRs for same source
  - May be helpful for system diagnostics

Code Requirements - Lab 1, Part 4

- Use DISI instruction to disable level 1-6 interrupts
  - DISI #N
- Only 1 line of code to add!

Notes - Lab 1, Part 4

- Vary the DISI count by small amounts and observe the behavior
  - Is there a difference? Why not?
- Upon device running:
  - Why does S4 always toggle LED1?
  - What happens with the other switches and associated LEDs?
- How do you prevent level 7 interrupts?
  - Set CPU Priority level $\Rightarrow$ IPL[3:0] = 7

Review - Lab 1

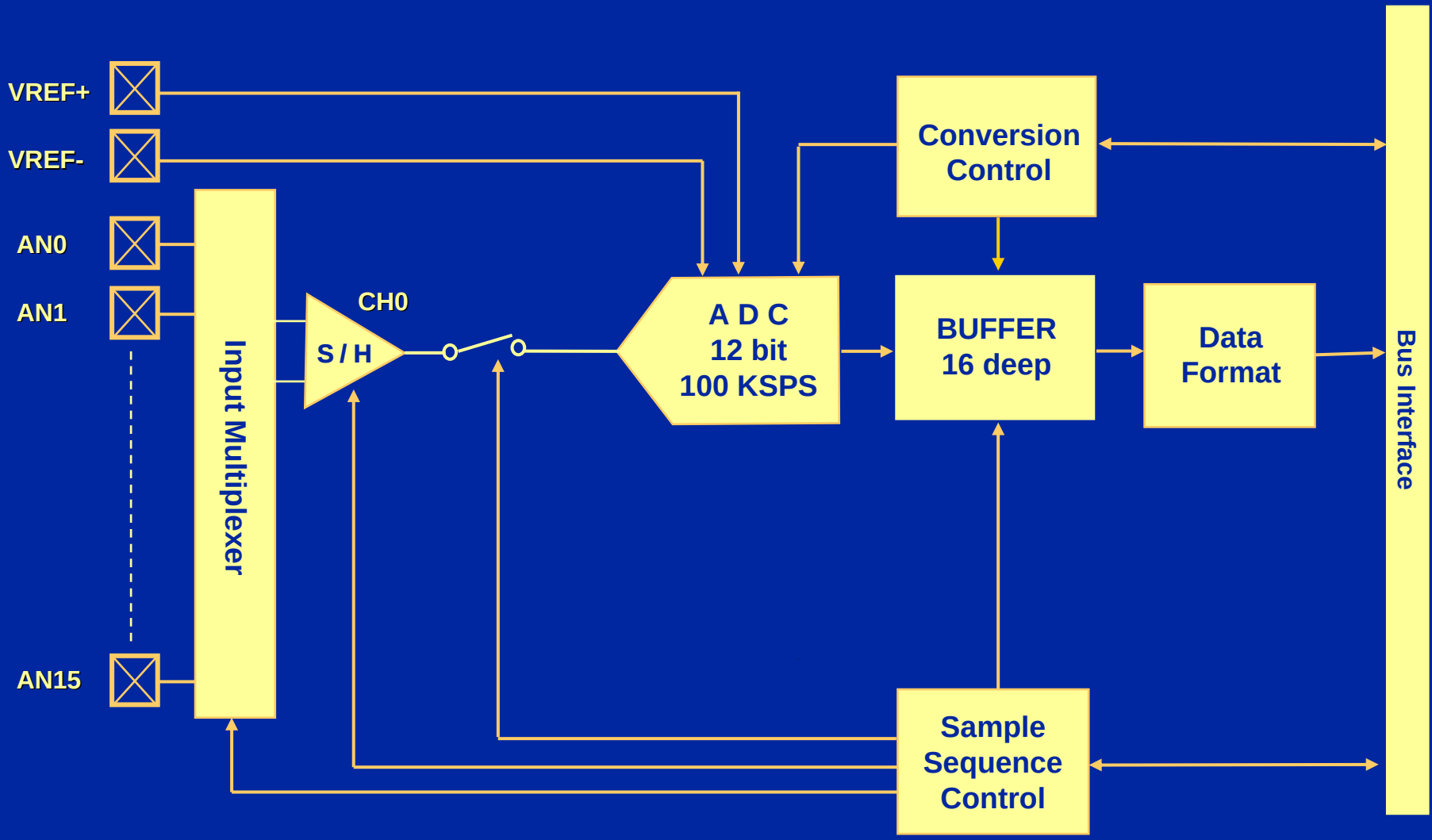
- Defining Interrupts
- Modifying Peripheral Interrupt Priorities
- Interrupt Nesting
- Defining Alternate Vector Interrupts
- Disabling Interrupts
- Interrupt Latency considerations

12-bit ADC Module

12-bit A/D Converter

- 12 bit Resolution, +/- 1 bit accuracy
- 100 K Samples / Sec Sampling Rate
- Up to 16 input channels
- External V_{REF}+ and V_{REF}-
- Analog Input Range: 0 to 5V (or V_{REF}- to V_{REF}+)
- 16 word buffer for A/D conversion results
- Automatic scheduled conversion options
- Formatting options for reading conversion results
- No missing codes

12-bit A/D 轉換器



12-bit A/D Converter

- Control Register Summary
  - ADCON1
  - ADCON2
  - ADCON3 - A/D Clock Control Register
  - ADCHS - A/D Input Select Register
  - ADCSSL - A/D Input Scan Select Register
  - ADPCFG - A/D Port Configuration Register
    - All A/D port pins configured as analog at Reset
    - Digital - port read enabled, A/D input to AVss
    - Analog - port read disabled

ADCON1 暫存器 (MSB)

ADCON1

ADON – 設爲 1 時，將 ADC 轉換功能致能(Enable)

ADSIDL – 設爲 1 時，ADC 在 IDLE 模式下會關閉

FORM<1:0> - AD轉換後資料的輸出格式選擇

- 11 = 有號小數 (Signed fractional)
- 10 = 無號小數 (Fractional)
- 01 = 有號整數 (Signed integer)
- 00 = 整數 (Integer)

ADC 輸出格式
16-bit

	b15 b14 ← ---------- → b1 b0
有號小數	$\overline{d11}$,d10,d09,d08,d07,d06,d05,d04,d03,d02,d01,d00, 0 , 0 , 0 , 0
小數	d11,d10,d09,d08,d07,d06,d05,d04,d03,d02,d01,d00, 0 , 0 , 0 , 0
有號整數	$\overline{d11}$,$\overline{d11}$,$\overline{d11}$,$\overline{d11}$,$\overline{d11}$,d10,d09,d08,d07,d06,d05,d04,d03,d02,d01,d00
整數	0 , 0 , 0 , 0 ,d11,d10,d09,d08,d07,d06,d05,d04,d03,d02,d01,d00

ADCON1 暫存器 (LSB)

ADCON1

SSRC<2:0> - 啓動AD轉換的觸發信號來源選擇

111 = 使用內部時序設定取樣時間及轉換時間(自動轉換)

(需參考到 ADCON3 暫存器的設定)

011 = 馬達控制 PWM 間隔結束時，結束取樣ADC開始轉換

(需參考到 SEVTCMP 暫存器的設定) 馬達專用的dsPIC才有此功能

010 = Timer 3 計時比較完成後，結束取樣ADC開始轉換

001 = INT0 腳位電位轉態時，結束取樣ADC開始轉換

(需參考到 INTCON2 <INT0EP> 位元的設定)

000 = 手動轉換， SAMP=1 時取樣，清除 SAMP 位元後轉換

ADCON1 暫存器 (LSB)

ADCON1

ASAM :

- 設爲 1 時，當上次AD轉完後成後，立即自動取樣 (SAMP位元會自動設爲 1，但AD不會轉換，只要將SAMP清爲 0 後，AD 就會開始轉換)
- 設爲 0 時，採手動取樣模式，將 SAMP位元設定爲 1 時取樣

SAMP :

- 條件；當ASAM = 0 時， SAMP 寫入 1 時就開始取樣；將 SAMP 清爲 0 時，採樣工作結束AD開始轉換(當DONE=1或ADIF=1 時轉換完成)
- 當 SSRC = 000 (上一頁所講的手動轉換模式)， 只要將 SAMP 位元清爲 0， AD 就開始轉換

ADCON1 暫存器 (LSB)

ADCON1

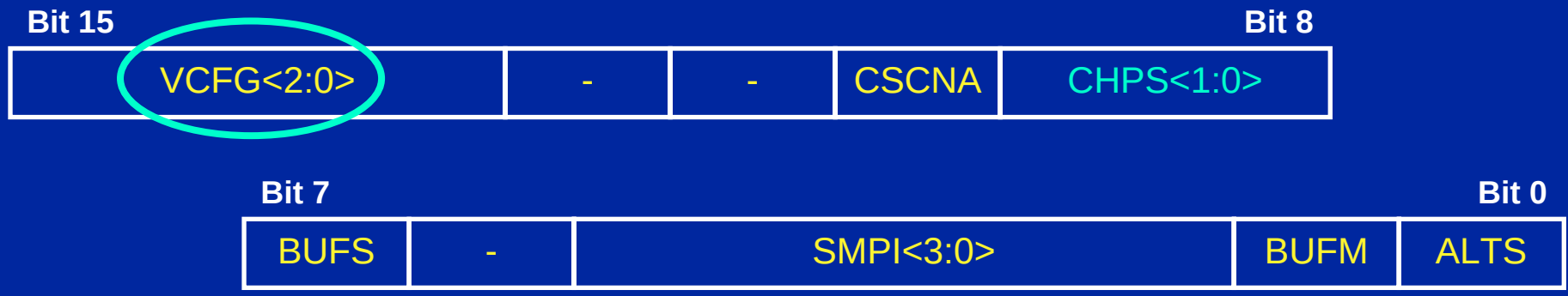
Bit 7					Bit 0
SSRC<2:0>	-	SIMSAM	ASAM	SAMP	DONE

DONE：AD 轉換狀態指示位元

等於 1 時，A/D 轉換完成，該位元可以用軟體清除 或新的轉換動作啓動時也會被清爲 0

ADCON2 暫存器

ADCON2



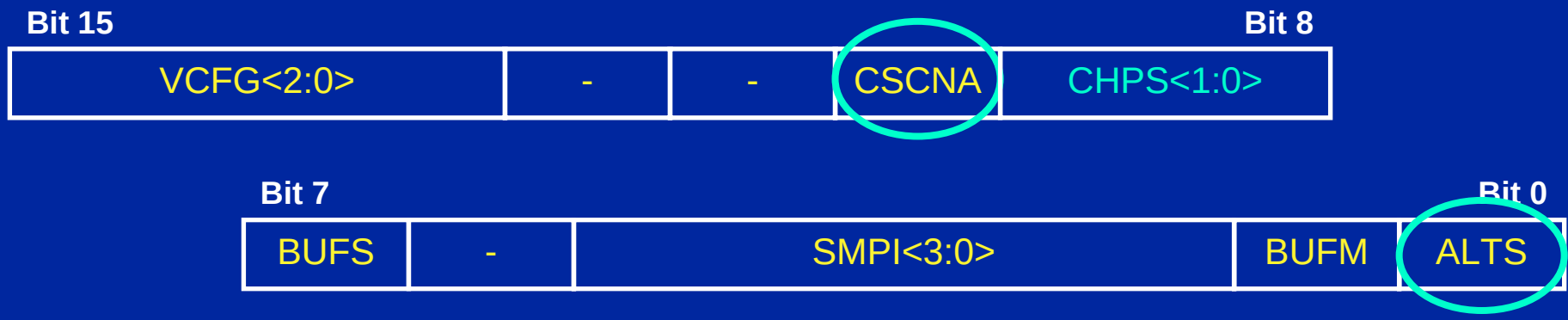
VCFG<2:0> - 參考電壓源的選擇

- 000 – AVDD ， AVSS
- 001 – 外接 VREF+ ， AVSS
- 010 – AVDD ， 外接 VREF-
- 011 – 外接 VREF+ ， 外接 VREF-
- 1xx – AVDD ， AVSS

轉換的輸入電壓範圍被限制在參考電壓範圍之間，在此範圍之外**(12-bit)**的電壓會以最大值**(0x7FF)** 或以最小值**(0x000)** 表示

ADCON2 暫存器

ADCON2



CSCNA : =1時，採用自動掃描方式自A組多工器輸入

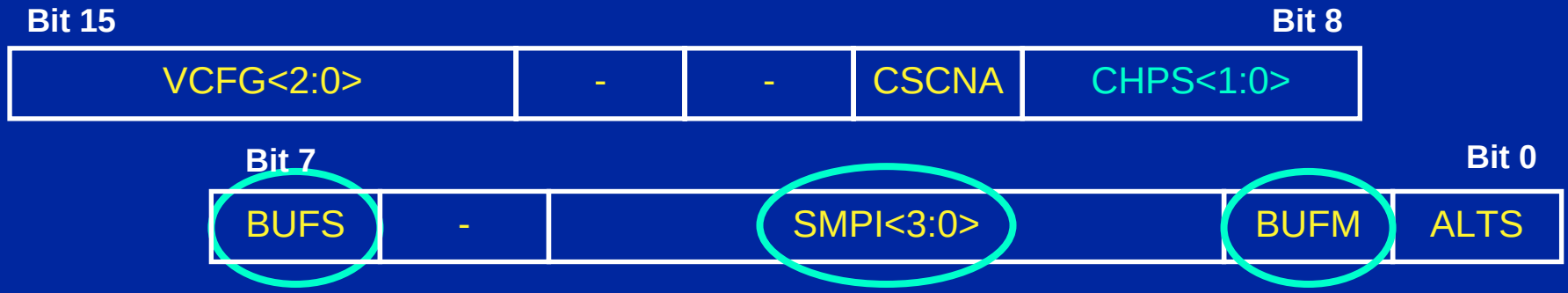
ALTS : = 1時，先選 A 組輸入再選 B 組輸入相互交換著選擇多工器 (A→B→A→B→A→B ….)

= 0時，輸入只選 A 組多工器

關於此項功能會在後面詳細說明使用方式

ADCON2 暫存器

ADCON2



SMPI<3:0> - 設定 AD 要轉換幾次後才產生一次中斷
(這些轉換後的資料會被存到 ADCBUFx 的暫存器列裡)

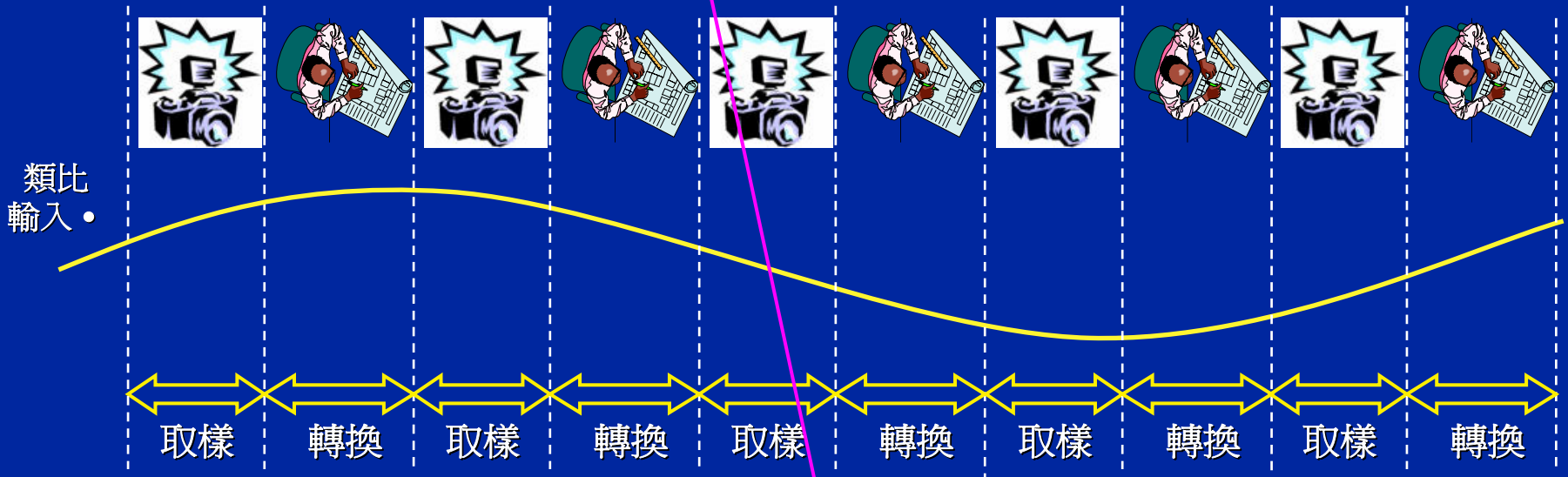
BUFM – ADCBUFx 暫存器列設定成單組或兩組模式

BUFS – ADCBUFx 採兩組模式時的狀態指示位元

轉換結果儲存規則

- AD 轉換的結果儲存到哪裡?
  - 當 BUFM 位元 = 0
    - ✓ 結果存入單一組16個緩衝器裡 - ADCBUF0,1,2…E,F
    - ✓ 每次 ADIF 中斷產生後，指標歸零指向 ADCBUF0
    - ✓ 每次 ADIF 中斷產生後，新的轉換資料會蓋掉上一次的 存在 ADCBUF0 的資料
    - ✓ 考慮資料到下一次資料被覆蓋時，軟體是否有足夠的處理時間?
  - 當 BUFM 位元 = 1
    - ✓ 結果會分組自動存入兩組的8個緩衝器裡
    - ✓ BUFS 位元會指出目前 AD 轉換使用那組緩衝器
    - ✓ BUFS = 1, AD 目前填入第二組緩衝器 ADCBUF8 - F
    - ✓ 每次中斷，最多可儲存 8 個資料

A/D 轉換器
轉換三次後產生中斷的說明



第一次結果 ADCBUF0
第二次結果 ADCBUF1
第三次結果 ADCBUF2
第四次結果 ADCBUF0
第五次結果 ADCBUF1

ADIF

用軟體讀取緩衝器資料

軟體清除 ADIF 旗號

ADCON2

VCFG<2:0>	-	-	CSCNA	-	-

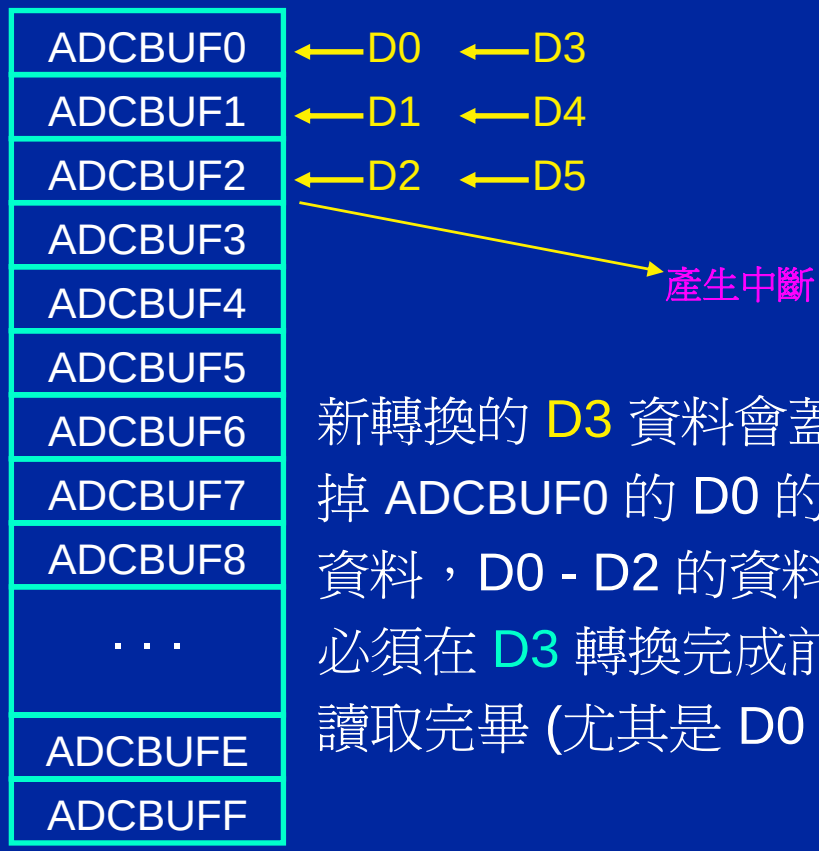
Bit 0

BUFS	-	SMPI = 0010	BUFM=0	ALTS

A/D 轉換器
轉換儲存模式 & 轉換次數與中斷

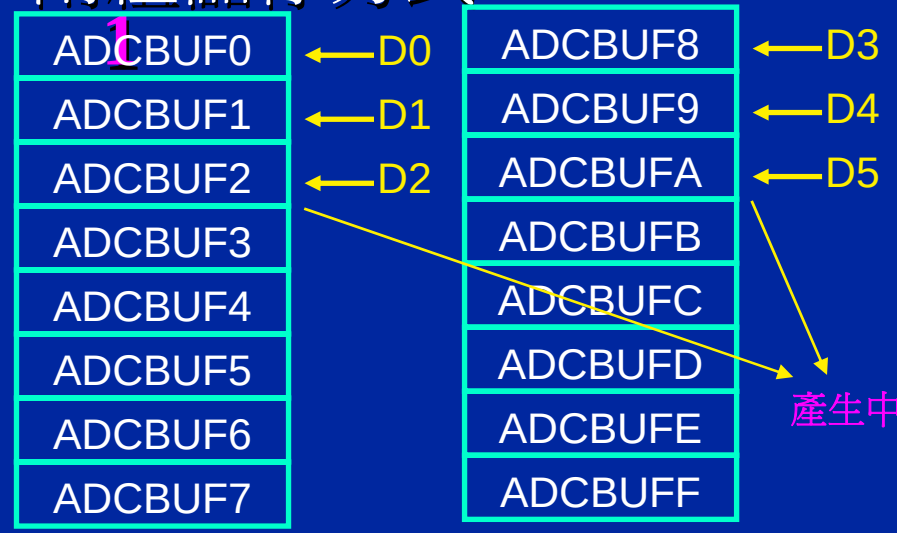
設定爲三次轉換後產生一次中斷 - **SMPI = 0010**

單組儲存方式 - BUFM = 0



新轉換的 D3 資料會蓋掉 ADCBUF0 的 D0 的資料，D0 - D2 的資料必須在 D3 轉換完成前讀取完畢 (尤其是 D0)

兩組儲存方式 - BUFM = 1



D6 的資料會蓋定 ADCBUF0 的 D0，所以 D0 - D2 的資料必須在 D6 轉換完成前讀取完畢

ADCON3 暫存器

ADCON3

SAMC<4:0>：設定自動取樣時間 (0 T_{AD} to 31 T_{AD})

ADRC：=1時，使用內建RC振盪時脈；=0時，使用系統時脈

ADCS<5:0> - AD 轉換時脈的時間 (T_{AD})

111111= (Tcy / 2) (ADCS<5:0>+1) = 32 Tcy

.....

000000= Tcy/2

自動取樣轉換的時間計算

- 必需先設定爲自動取樣與轉換模式
  - 假設石英晶體用7.3728MHz，並使用16 倍的倍頻電路使工作頻率爲 117.9648MHz = 8.477nS (Fosc)
  - Tcy = (1 / Fosc) x 4 = 33.91nS
- 在規格書裡有規定
  - 最小的 TAD 需大於 713nS，最小的 轉換時間=10uS
  - 若 ADCON3 = 0x1F3F (SAMC=11111, ADCS=111111)

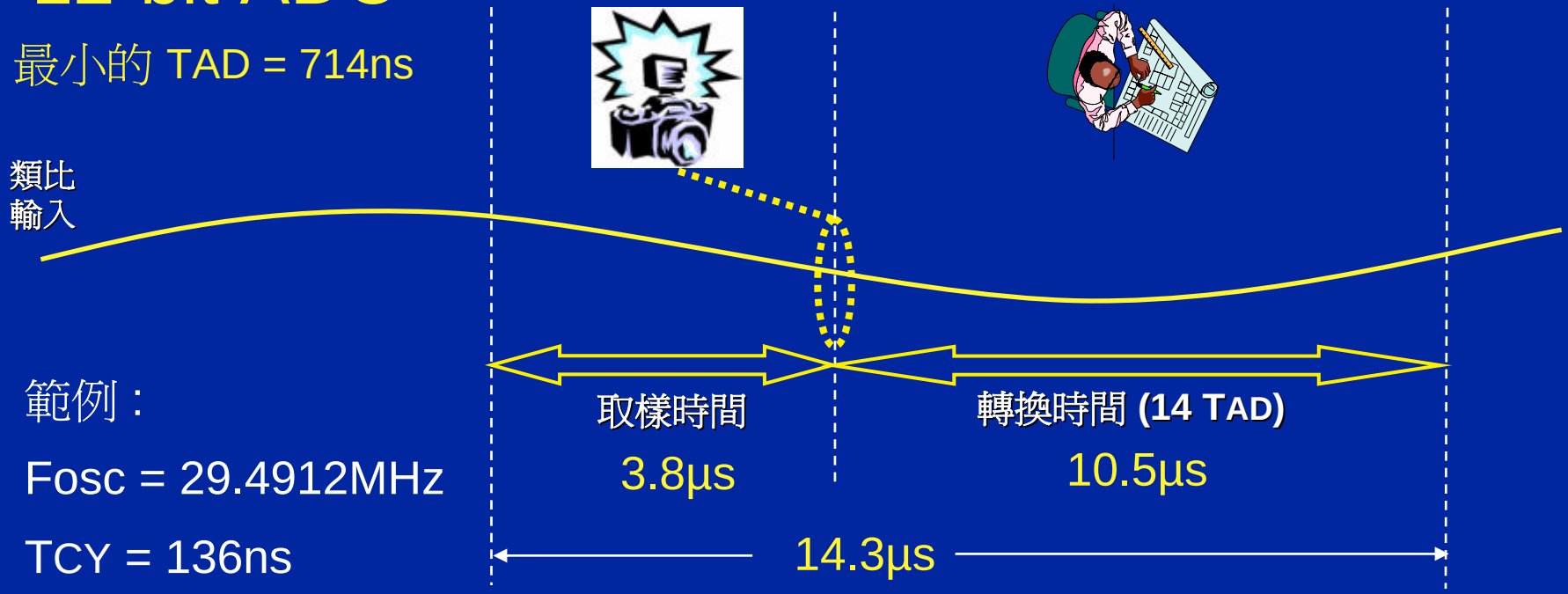
Tad = (ADCS<5:0>+1) Tcy / 2 = (63+1) Tcy / 2= Tcy * 32 = 33.91nS x 32 = 1.08512uS ,
轉換時間 Tconv = 14Tad ，取樣時間 = 31 Tad

A/D = 取樣時間 + 轉換時間 = 31 Tad + 14 Tad = (31 + 14) x 1.08512uS= 48.83uS

轉換時間計算範例

12-bit ADC

最小的 TAD = 714ns



範例：

Fosc = 29.4912MHz

TCY = 136ns

TAD = TCY(ADCS+1)/2 = 748ns (Tconv. = 14 TAD = 10.5 µs)

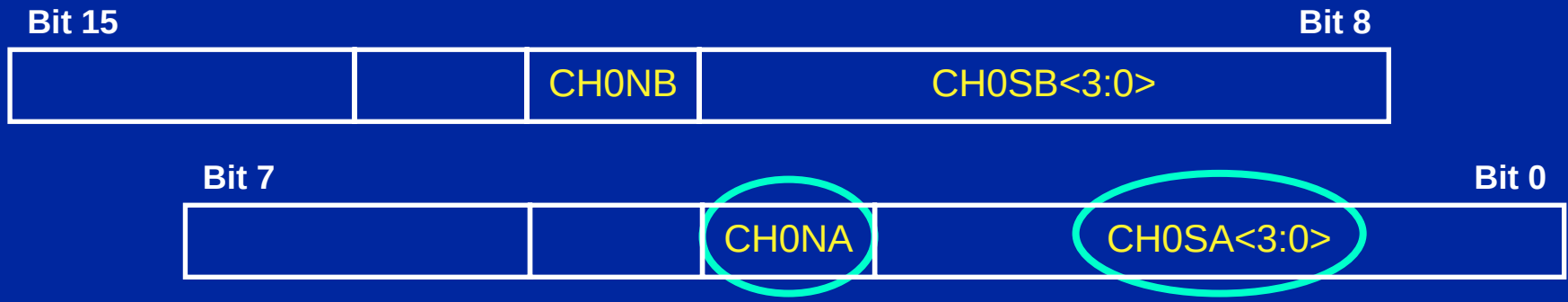
TSAMP = TAD x SAMC = 3.8µs

ADCON3

-	-	-	SAMC = 00101	ADRC = 0	-	ADCS = 001010

ADCHS 暫存器

ADCHS



CH0NA : A 組多工器負端輸入選擇

1 = 輸入選 AN1 , 0 = 輸入選 Vref-

CH0SA<3:0> : A 組多工器正端輸入選擇

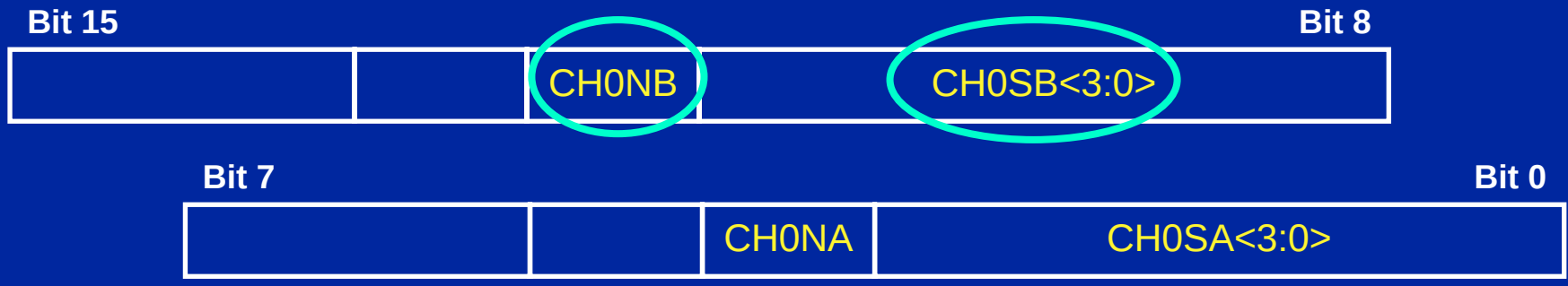
1111 : 選擇 AN15爲輸入端

……

0000 : 選擇 AN0爲輸入端

ADCHS 暫存器

ADCHS



CH0NB : B 組多工器負端輸入選擇

1 = 輸入選 AN1 , 0 = 輸入選 Vref-

CH0SB<3:0> : B 組多工器正端輸入選擇

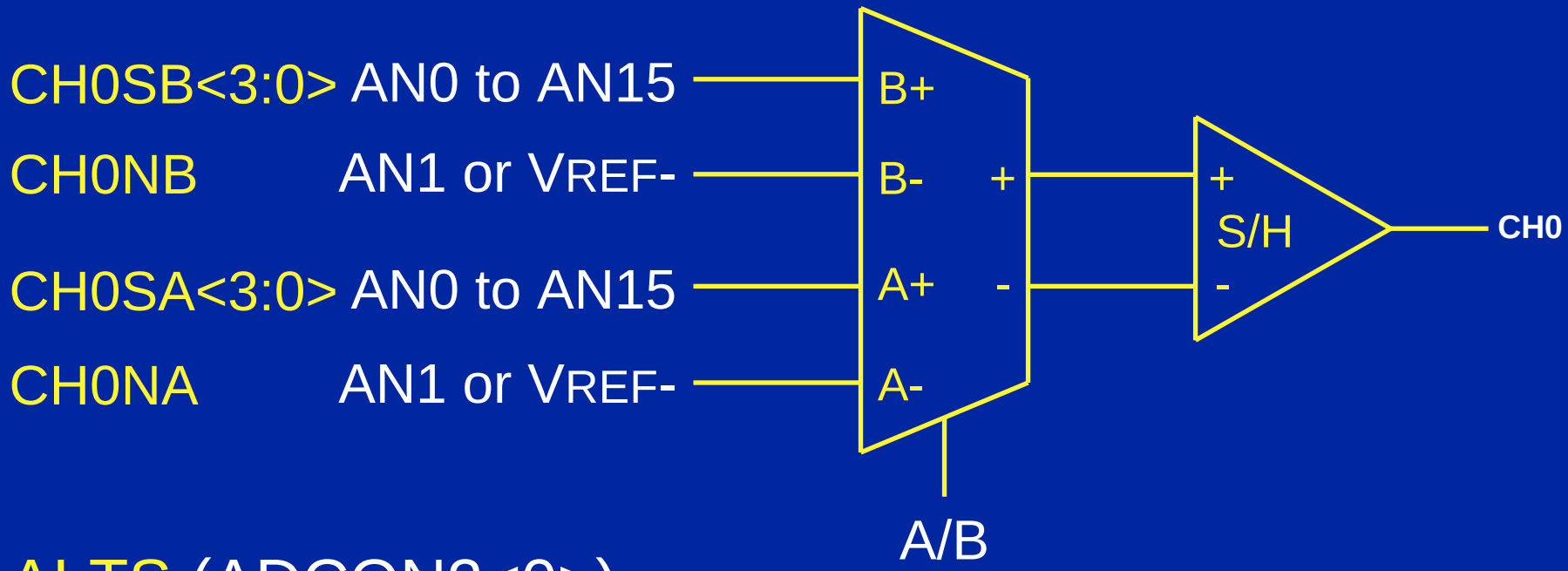
1111 : 選擇 AN15爲輸入端

……

0000 : 選擇 AN0爲輸入端

A/D 轉換器
多工 (交錯式) 輸入

多工器的取樣輸出 CH0



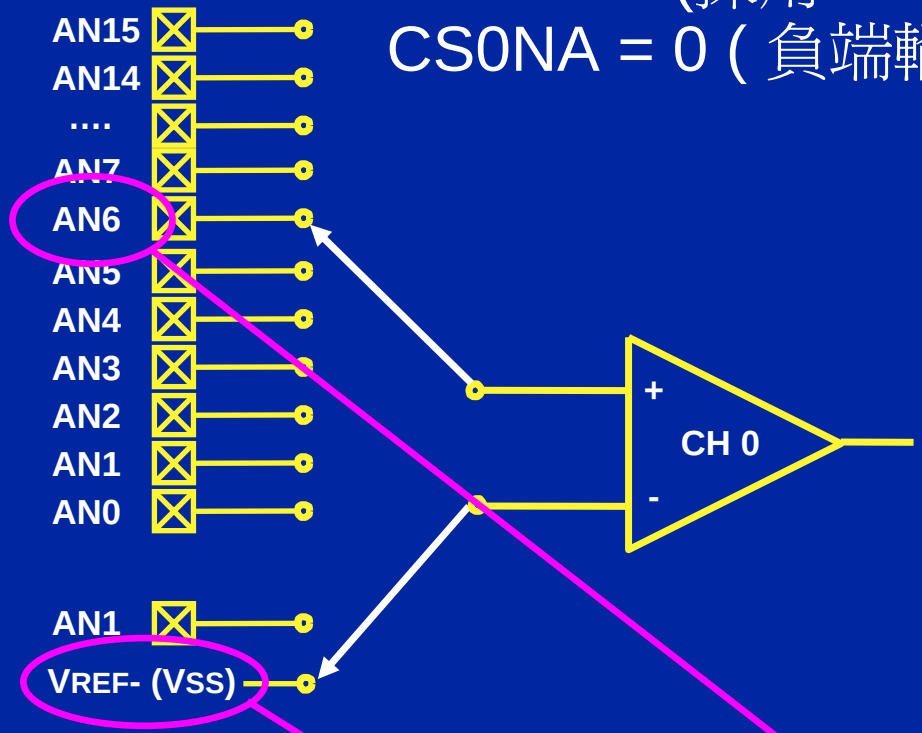
ALTS (ADCON2<0>)

= 0時，輸入只選 A 組多工器

= 1時，先選 A 組輸入再選 B 組輸入相互交替

A/D 轉換器
多工 (交錯式) 輸入說明 (一)

ALTS = 1 (採用 A、B 兩組交替模式)
CS0NA = 0 (負端輸入爲 Vref- 即接地)



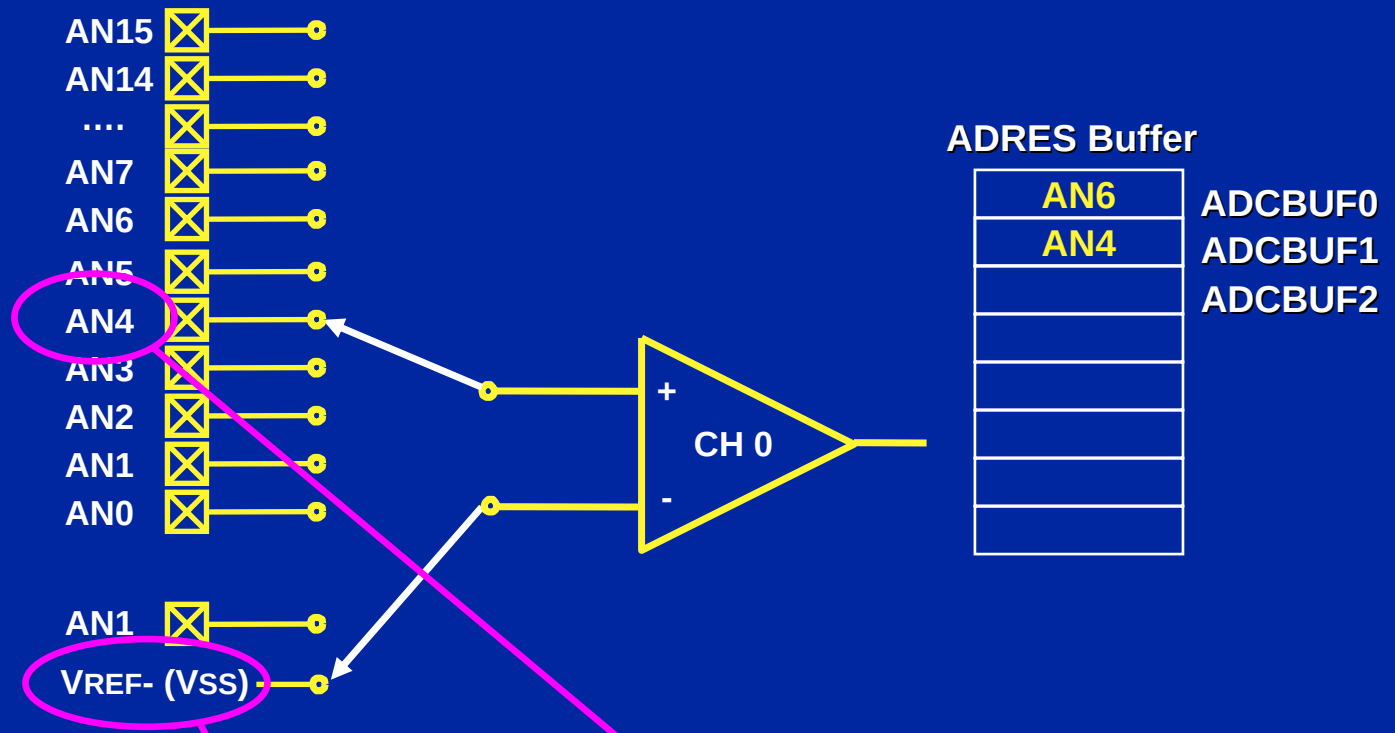
ADRES Buffer

AN6	ADCBUF0
	ADCBUF1
	ADCBUF2

ADCHS

		CH0NB=0	CH0SB = 0100	
		CH0NA=0	CH0SA = 0110	Bit 0

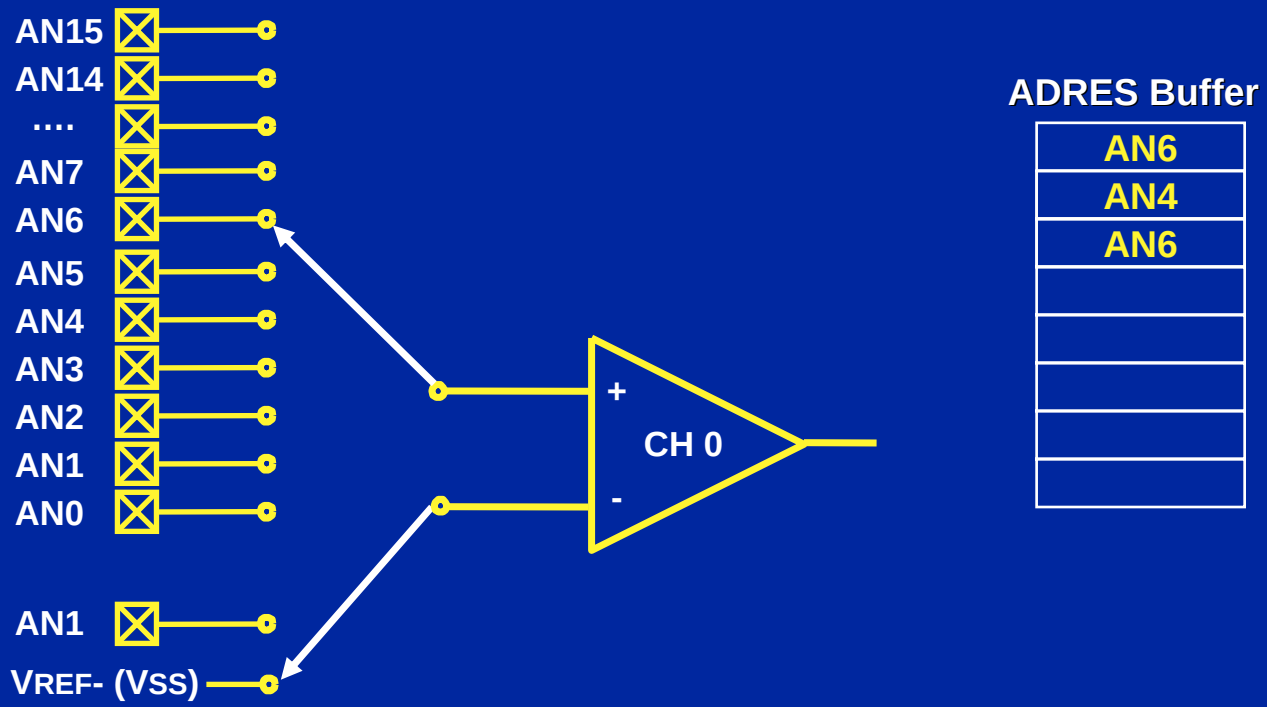
A/D 轉換器
多工 (交錯式) 輸入說明 (二)



ADCHS

		CH0NB=0	CH0SB = 0100	
		CH0NA=0	CH0SA = 0110	Bit 0

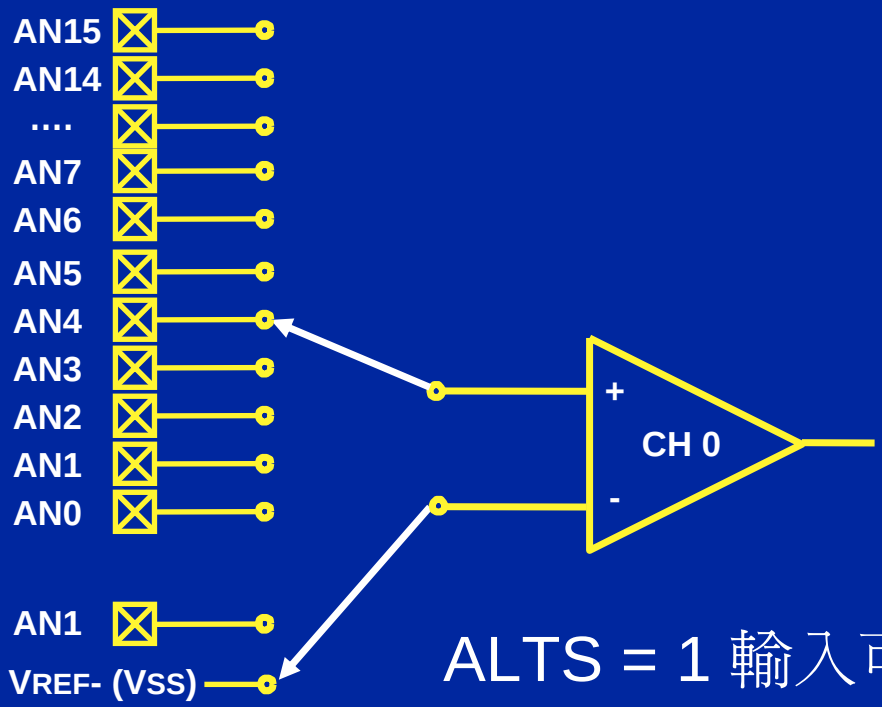
A/D 轉換器
多工 (交錯式) 輸入說明 (三)



ADCHS

						CH0NB=0	CH0SB = 0100								
															Bit 0
								CH0NA=0	CH0SA = 0110						

A/D 轉換器
多工 (交錯式) 輸入說明 (四)



ADRES Buffer

AN6
AN4
AN6
AN4

ALTS = 1 輸入可自動交互更換

ADCHS

		CH0NB=0	CH0SB = 0100

Bit 0

		CH0NA=0	CH0SA = 0110

ADCSSL 暫存器

ADCSSL

CSSL<15:0> - 輸入掃描選擇

- 1 – 啓動掃描該腳位
- 0 – 不掃描

啓動掃描功能需將 CSCNA (ADCON2<10>) 位元設爲 1

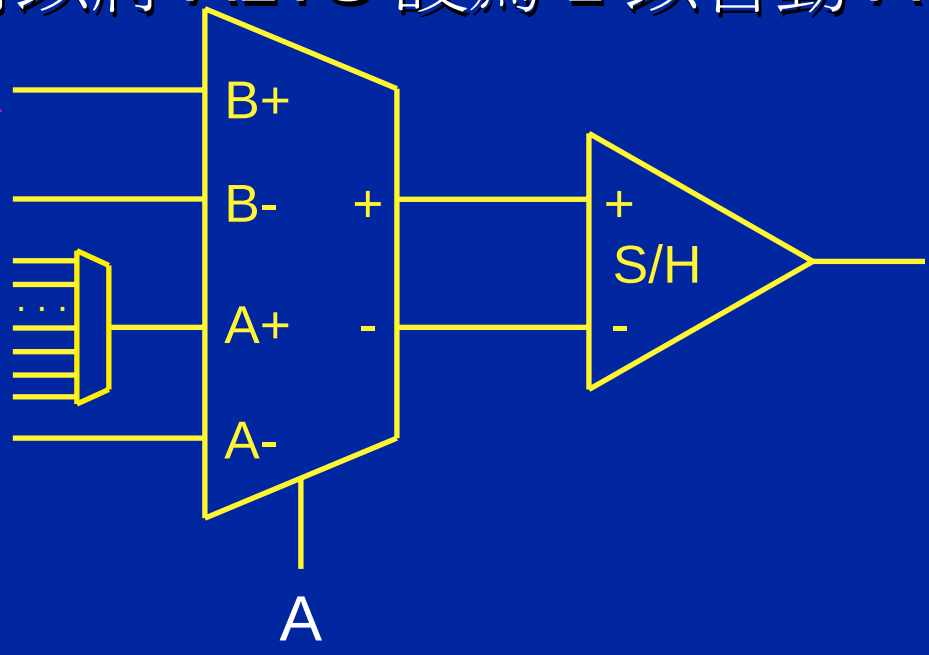
A/D 轉換器 - 掃描輸入

- 注意！只有 A 多工器的輸入端才有掃描之功能
- 這時 CH0SA<3:0> 的輸入被 CSSL<15:0> 的輸入取代
- 啓動掃描功能時，也可以將 ALTS 設爲 1 以啓動 A / B 交互切換功能

範例先討論 **ALTS=0** 的例子
即暫不使用 **B** 組多工器輸入

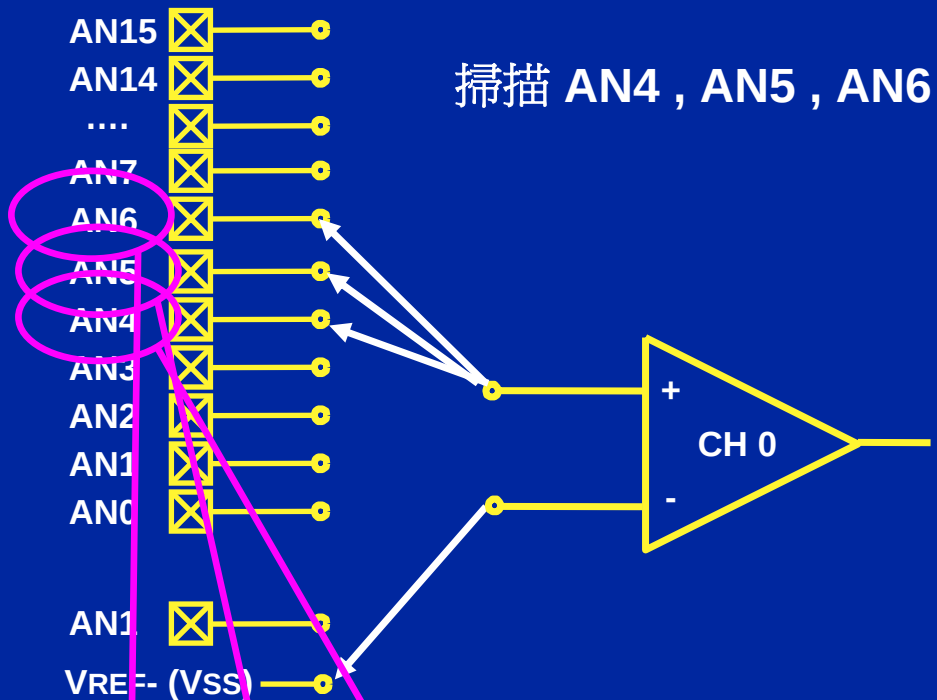
CSSL<15:0> AN0 to AN15

CH0NA AN1 or VREF-



A/D 轉換器
掃描輸入說明

ALTS = 0
CSCNA = 1



掃描 **AN4 , AN5 , AN6**

ADRES Buffer

AN4
AN5
AN6

ADCSSL

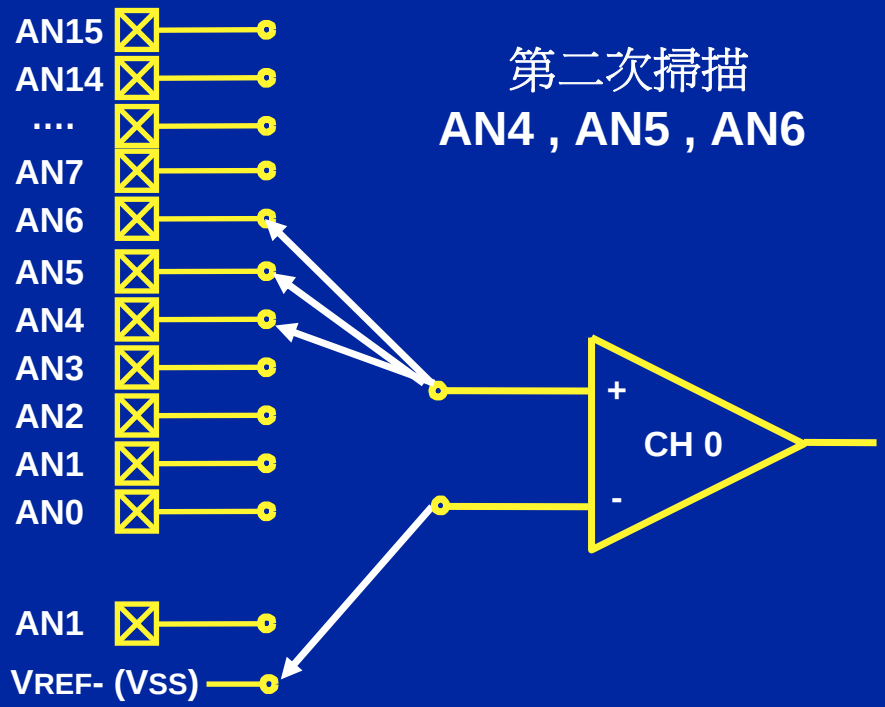
CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8

Bit 0

CSSL7	CSSL6=1	CSSL5=1	CSSL4=1	CSSL3	CSSL2	CSSL1	CSSL0

A/D 轉換器
掃描輸入說明

ALTS = 0
CSCNA = 1



第二次掃描
AN4 , AN5 , AN6

ADRES Buffer

AN4
AN5
AN6
AN4
AN5
AN6

ADCSSL

CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8

Bit 0

CSSL7	CSSL6=1	CSSL5=1	CSSL4=1	CSSL3	CSSL2	CSSL1	CSSL0

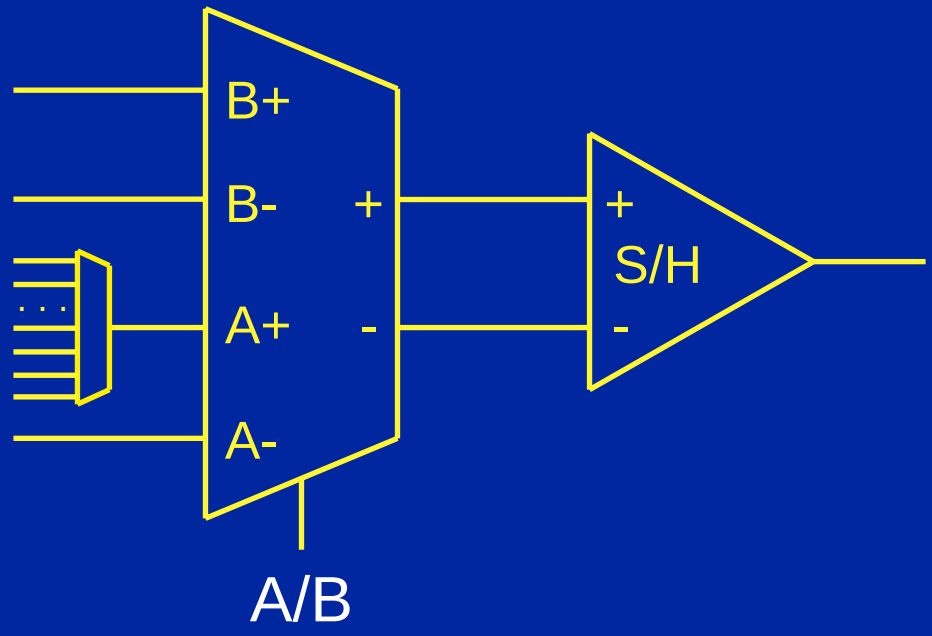
A/D轉換器
使用多工器與掃描輸入說明

CH0SB<3:0> AN0 to AN15

CH0NB AN1 or VREF-

CSSL<15:0> AN0 to AN15

CH0NA AN1 or VREF-



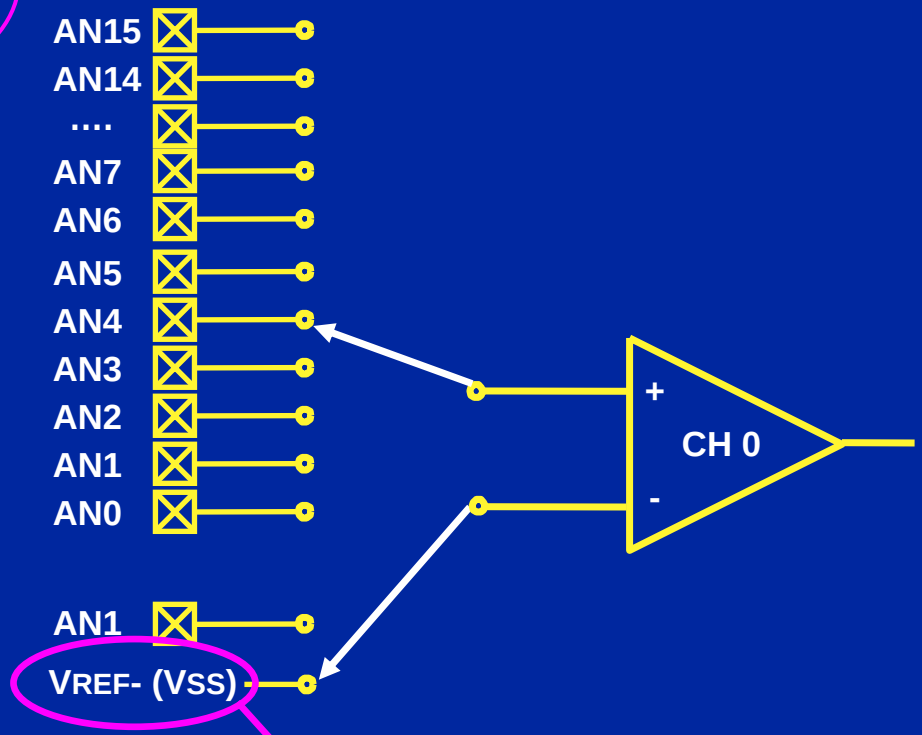
ALTS – 從 A 組輸入先掃描一個輸入再選 B 組輸入，
A / B 兩組之間相互交替做 AD 轉換

CSCNA – 啓動掃描多工器 A 的輸入功能

A/D轉換器
使用多工器與掃描輸入說明

ALTS = 1
CSCNA = 1

多工器 A 的輸入



ADCHS

		CH0NB=0	CH0SB = 0110

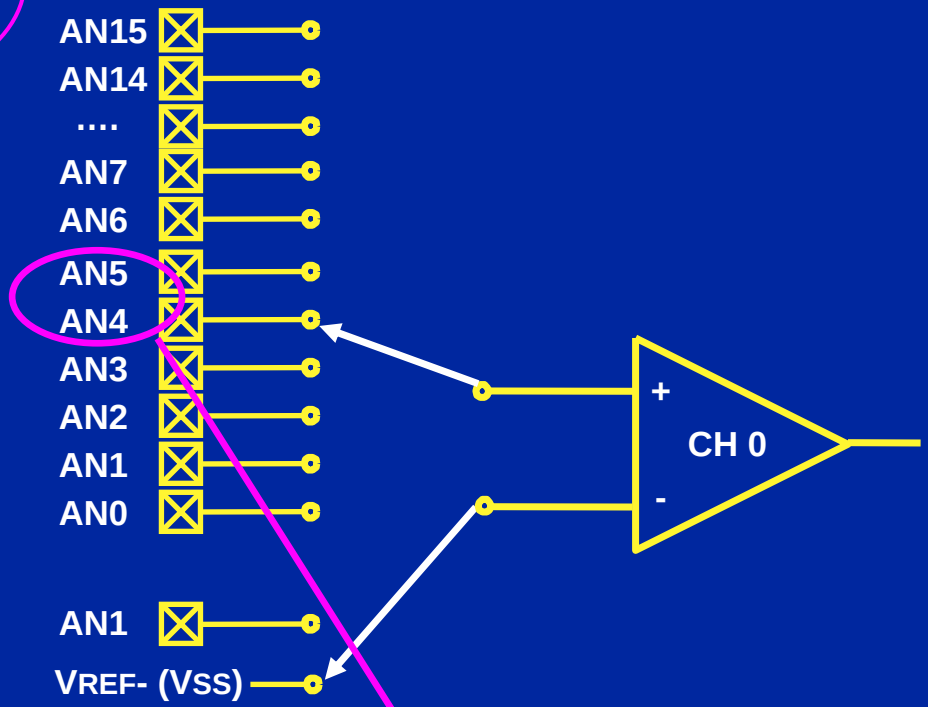
Bit 0

		CH0NA=0	CH0SA

A/D轉換器
使用多工器與掃描輸入說明

ALTS = 1
CSCNA = 1

多工器 A 的輸入



ADRES Buffer

AN4

ADCSSL

CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8

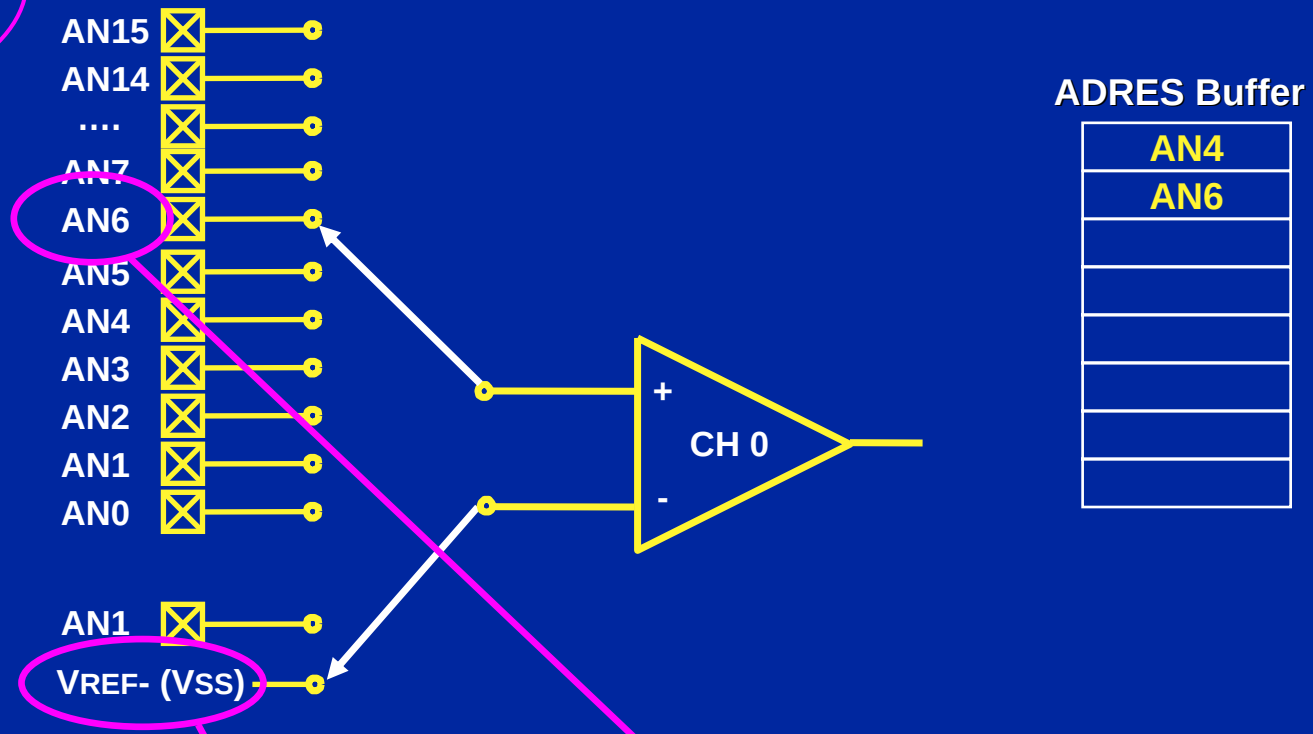
Bit 0

CSSL7	CSSL6	CSSL5=1	CSSL4=1	CSSL3	CSSL2	CSSL1	CSSL0

A/D轉換器
使用多工器與掃描輸入說明

ALTS = 1
CSCNA = 1

多工器 B 的輸入



ADCHS

		CH0NB=0	CH0SB = 0110

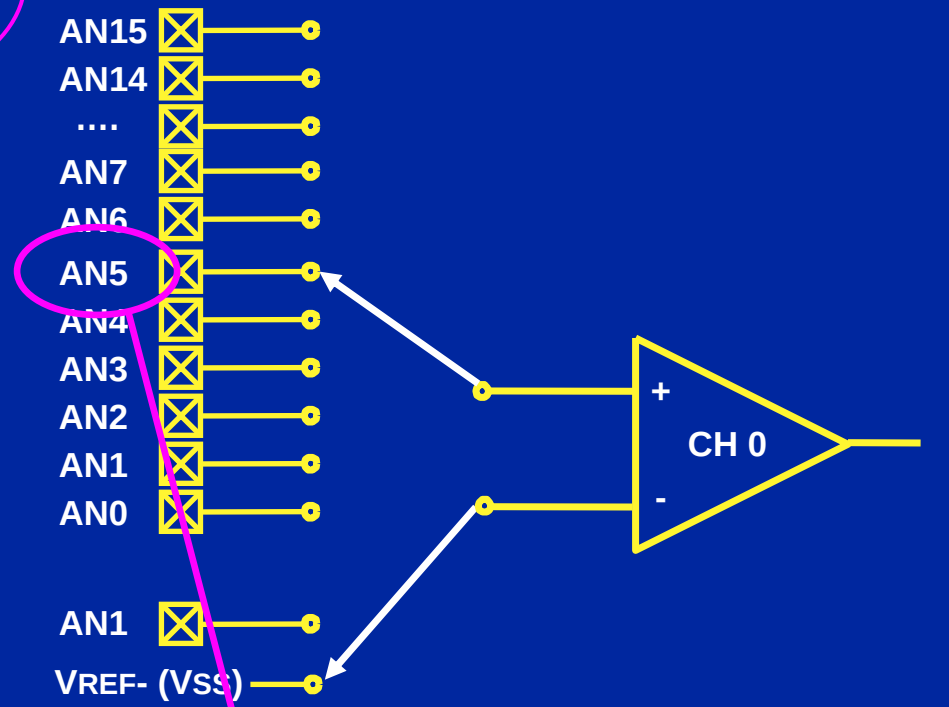
Bit 0

		CH0NA=0	CH0SA

A/D轉換器
使用多工器與掃描輸入說明

ALTS = 1
CSCNA = 1

多工器 A 的輸入



ADRES Buffer

AN4
AN6
AN5

ADCSSL

CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8

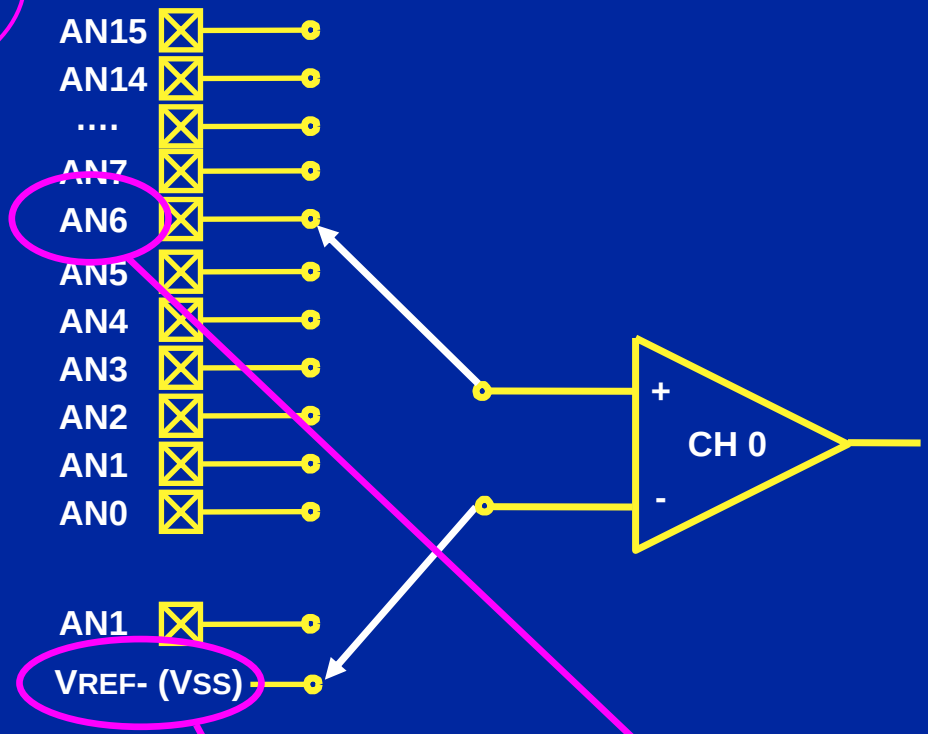
Bit 0

CSSL7	CSSL6	CSSL5=1	CSSL4=1	CSSL3	CSSL2	CSSL1	CSSL0

A/D轉換器
使用多工器與掃描輸入說明

ALTS = 1
CSCNA = 1

多工器 B 的輸入



ADRES Buffer

AN4
AN6
AN5
AN6

ADCHS

						CH0NB=0		CH0SB = 0110							

Bit 0

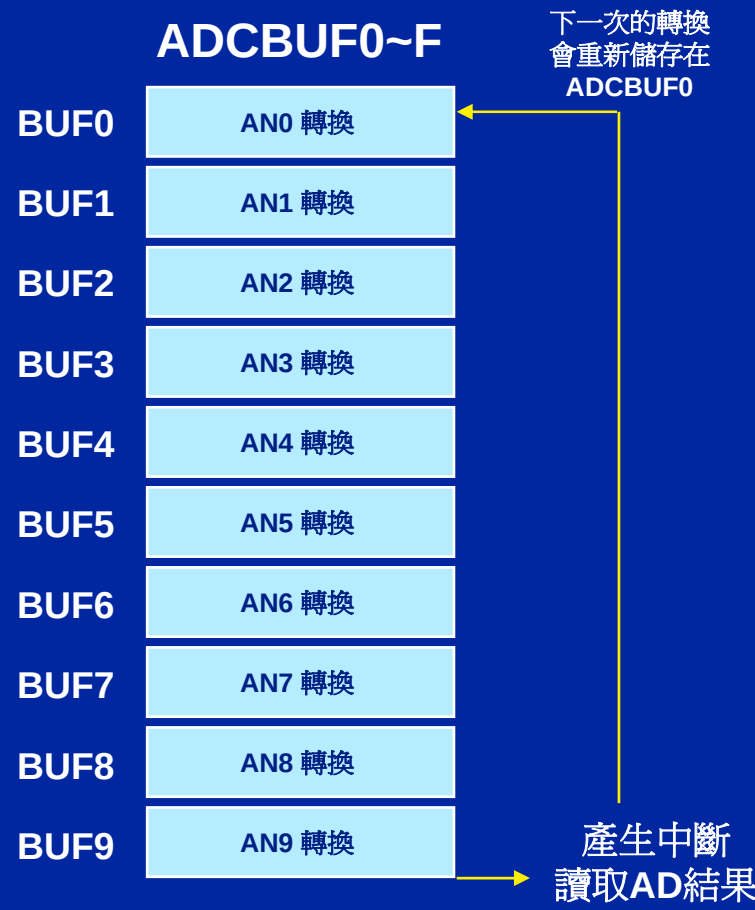
						CH0NA=0		CH0SA							

AD 掃描輸入與中斷設定

設定需求：

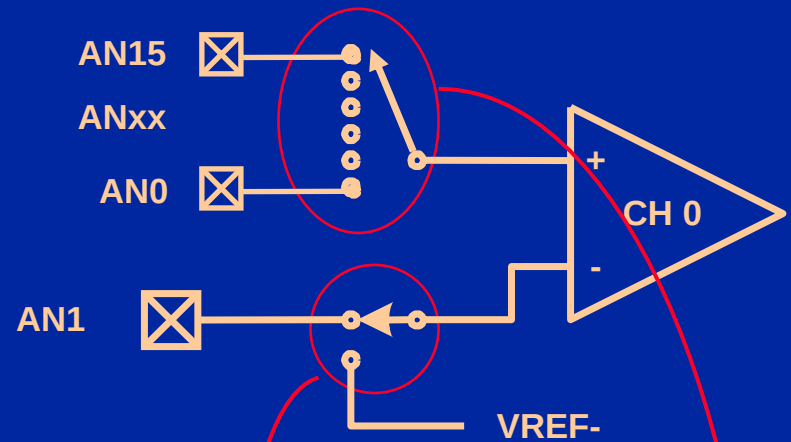
AN0-AN9 爲輸入腳，轉換 10 次後產生一次中斷

- SMPI<2:0>=1001，轉換10次後中斷一次
- BUFM=0，採用單一 buffer (16-word)
- ALTS=0，只使用 MUX A 爲輸入
- CH0SA<3:0>=N/A(掃描模式下該輸入無效)
- CHONA=0， Vref- for CH0- 輸入
- CSCNA=1，啓動輸入掃描功能
- CSSL<15:0>= 0000 0011 1111 1111，掃描輸入腳爲 AN0 ~ AN9
- CH0SB<3:0>= N/A
- CH0NB=N/A



差動輸入與交互掃描

- 輸入的設計是採用差動式輸入，正端的輸入可以減去負端的 AN1 或 VREF- 的電壓。



ADC Buffer

AN15-AN1
AN0-Vss
AN15-AN1
AN0-Vss

ADCHS

Vref- = Vss　　AN0　　Bit 8

		CH0NB=0	CH0SB<3:0> = 0000

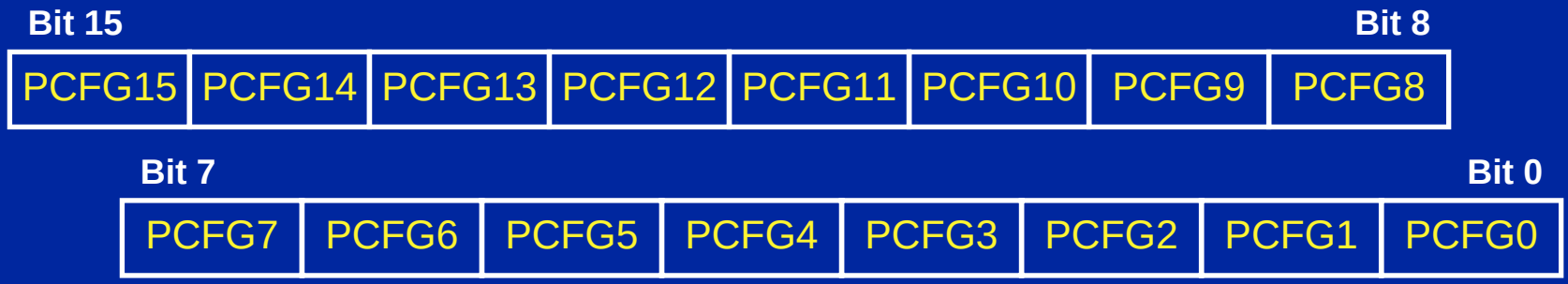
AN1　　AN15　　Bit 0

		CH0NA=1	CH0SA<3:0>=1111

ADPCFG 暫存器

ADPCFG

每一類比輸入腳位都可以單獨設定



PCFG<15:0> - 選擇類比輸入腳位

- 1 – 數位輸出入功能
- 0 – 類比輸入功能

- 零件編號不同，類比輸入的接腳數也會不一樣
- 重置後**(Reset)**，預設腳位的功能爲類比輸入
- 每一類比輸入腳位都可以單獨設定
- 設成類比輸入腳以後，**I/O** 讀取的結果爲 **0**

設定 AD 轉換模式

利用 SSRC<2:0> 來選擇 AD 轉換的觸發信號來源

000 = 手動轉換， SAMP=1 時取樣，清除 SAMP 位元後轉換

111 = 使用內部時序設定取樣時間及轉換時間(自動轉換)

~~011 = 馬達控制 PWM 間隔結束時，結束取樣ADC開始轉換~~

dsPIC30F Family Reference Manual 的 12-bit A/D Converter 書上的說明有誤

010 = Timer 3 計時比較完成後，結束取樣ADC開始轉換

001 = INT0 腳位電位轉態時，結束取樣ADC開始轉換

Hands-on Lab #2: Part #1

Code Requirements - Lab 2, Part 1

- Configure ADCON1
  - Unsigned Integer format
  - Automatic Sampling and Auto-trigger Conversion
- Configure ADCON2
  - Enable channel scanning
  - 4 Samples per Interrupt
- Configure ADCON3
  - A/D Sampling Rate = 8 kHz

Code Requirements - Lab 2, Part 1

- Configure ADCSSL to scan the following:
  - RP1 ⇒ AN6, RP2 ⇒ AN4
  - RP3 ⇒ AN5, U9 ⇒ AN8
- Configure ADPCFG
  - Set 4 channels for analog, remaining digital
- Enable ADC (do this last)
- In the A/D ISR
  - Unload the A/D buffer
  - Clear the A/D interrupt flag bit
- In all there are 8 lines of code to write

Objectives - Lab 2, Part 1

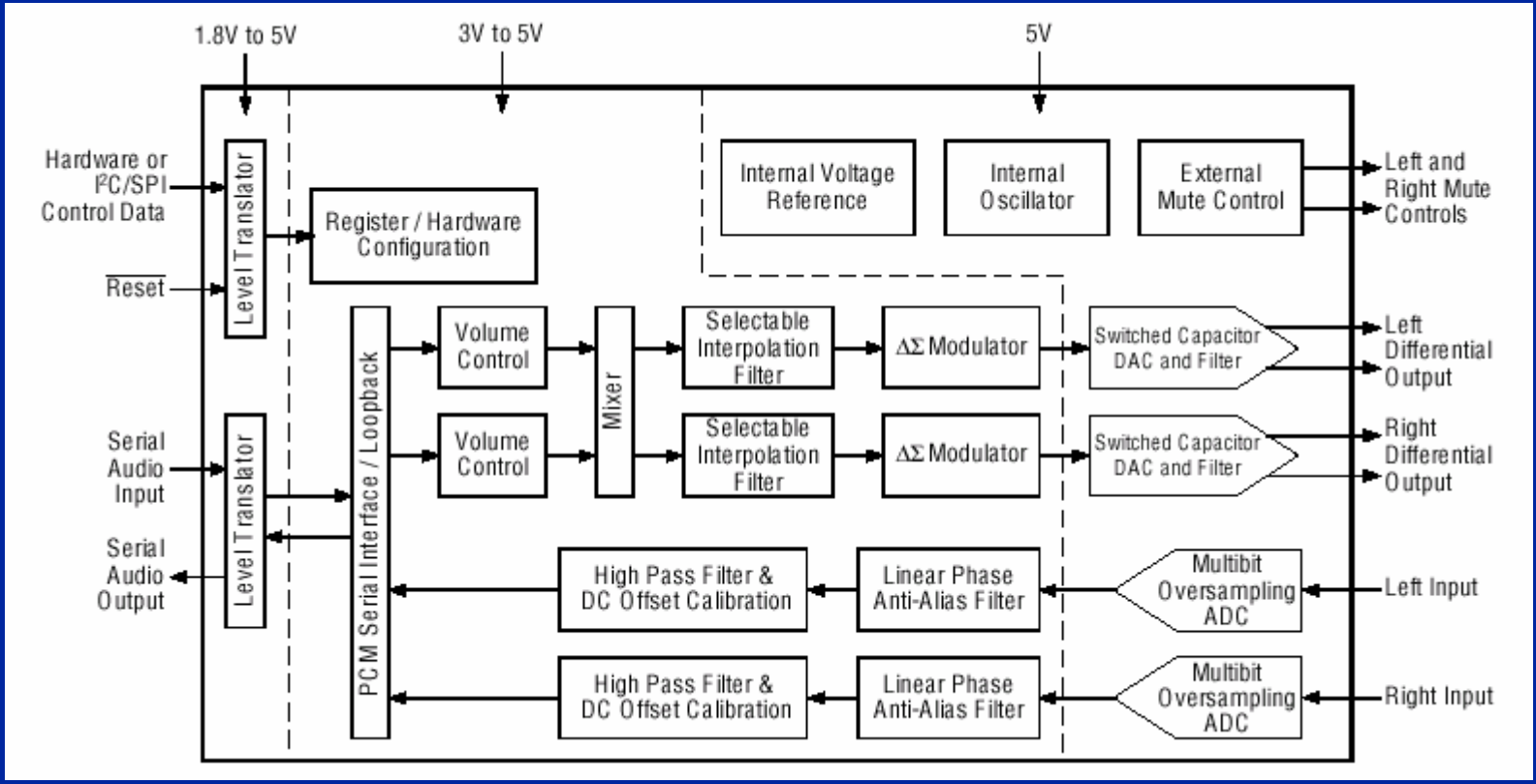
- Develop basic understanding on configuring 12-bit ADC module
  - What is the effective sampling rate per scanned input channel?
- Implement channel scanning on Mux A
- Continue using instruction set

Data Converter Interface

Why use a DCI?

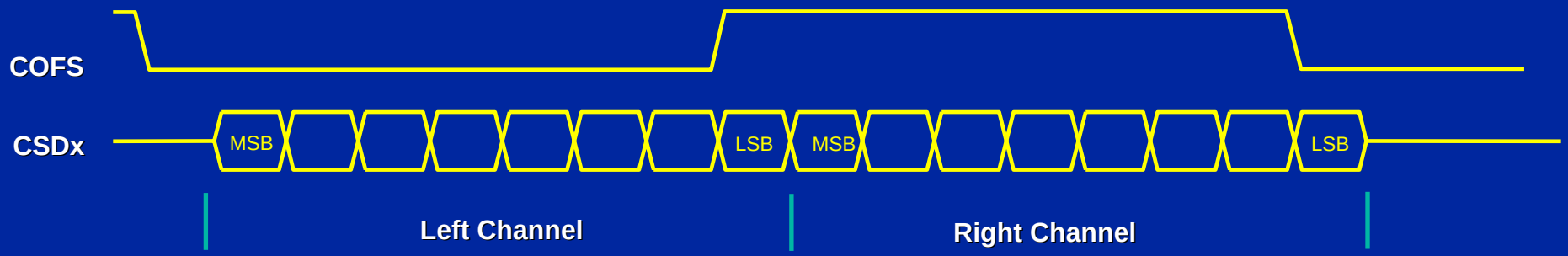
Typical I²S 24-bit Audio CODEC

- Serial Interfaces required:
  - DCI for Audio data
  - I²C™/SPI™ for Control data



I²S Protocol

- Philips specification (Inter-IC Sound)
  http://www.semiconductors.philips.com/acrobat/various/I2SBUS.pdf
- Common interface for stereo CODECs
- Frame Sync (FS) has 50% duty cycle
- FS edge(↑ or ↓) initiates transfer of appropriate channel data



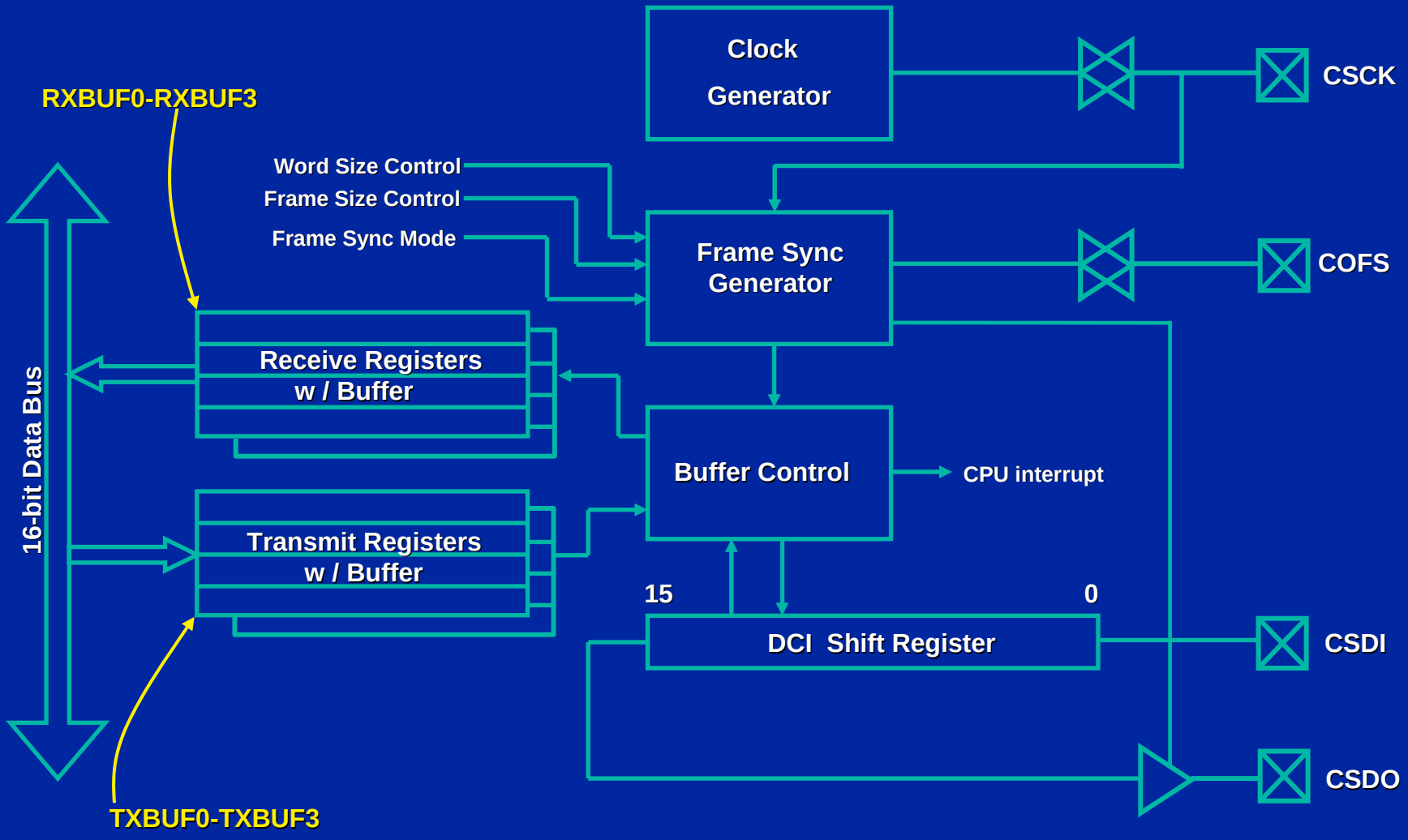
Why use a DCI?

- DCI designed for continuous, streaming data
  - Interface to CODECs, A/D and D/A converters
- Works like SPI™, but SCK is continuous
  - CODECs often uses SCK as operating clock
- Streaming data is organized into a frame
- Frame Sync (FS) signal initiates a transfer
  - FS usually occurs at the audio sample rate
  - FS is generated by the master device
- Frames typically contain several “timeslots”
  - Each “timeslots” contains a data word

DCI Features

- Support for PCM audio codecs
  - DCI supports speech, modem, and general audio applications
- Automatic synchronous serial data transfer
- Support for I^2S and AC'97 protocols
- TDM Mode features support up to 16 data timeslots
- Module has up to 4 word buffer (16-bit words)
- Master or slave operation
- Separate baud generator for SCK signal

DCI Block Diagram



Mode Selection Options

- COFSM<1:0> sets FS mode
  - 20 or 16-bit AC'97
  - I²S Frame Sync
  - Multi-channel Frame Sync (Slot)
- DJST - data alignment to FS pulse
- CSCKD, COFSD, set master/slave SCK/FS
- CSCKE sets clock sample edge

DCICON1 Register

DCIEN	-	DCISIDL	-	DLOOP	CSCKD	CSCKE	COFSD
bit15	14	13	12	11	10	9	bit8

UNFM	CSDOM	DJST	-	-	-	COFSM<1:0>	
bit7	6	5	4	3	2	1	bit0

Mode Selection Options

- WS<3:0> - data word size (4 to 16 bits)
- COFSG<3:0> - data frame size (1 to 16 words)
- (WS * FSG) determine frame size in bits
  - Maximum of 256 bits/frame; fcsck=256*FS
  - CODEC's will often use CSCK as operating clock
- BLEN<1:0> sets samples between interrupts
  - Also specifies size of buffer in words

DCICON2 Register

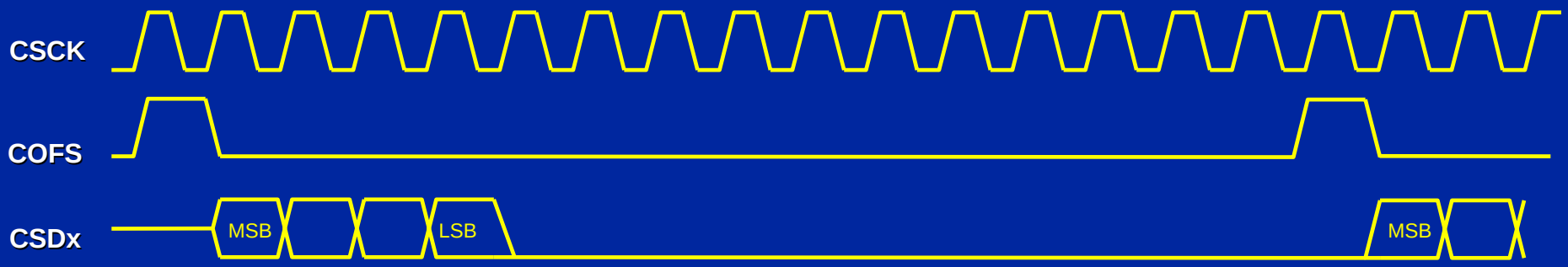
-	-	-	-	BLEN<1:0>		-	COFSG3
bit15	14	13	12	11	10	9	bit8

COFSG<2:0>			-	WS<3:0>			
bit7	6	5	4	3	2	1	bit0

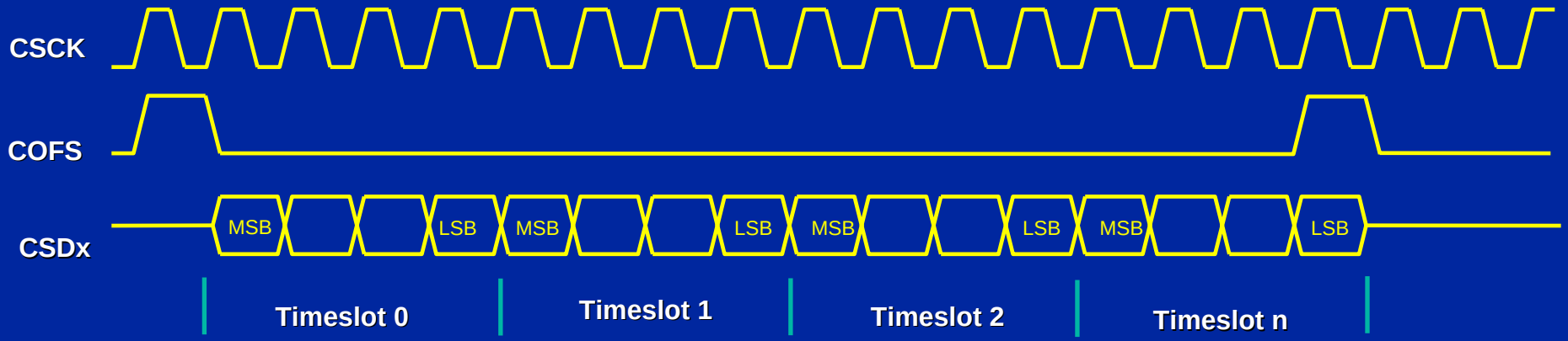
Data Converter Interface

- DCICON3 - Baud Rate Generator
  - Write 12-bit value to DCICON3 to enable SCK
  - May be enabled separately from DCI module
- TSCON, RSCON - Select Time Slots
  - Sets timeslots to transmit/receive (up to 16)
  - Buffer pointer incremented after each active slot
- DCISTAT - Status Bits
  - TMPTY, TUNF, RFUL, ROV
    - ✓ Buffer status dependent on BLEN <1:0>
    - ✓ Buffer status updated during buffer transfer
  - SLOT<3:0> - Status of current slot

Multi-channel (Slot) Timing



Single Word Transfer



Multiple Word Transfer

Slot Configuration Example

- Receive 1 channel audio data from a 16-bit Codec, that requires SCK = 64*fs and fs = 44.1 KHz
  - Buffer 1 sample per interrupt -

    BLEN = 00
  - Setup for Slot format, Master FS, Master SCK,

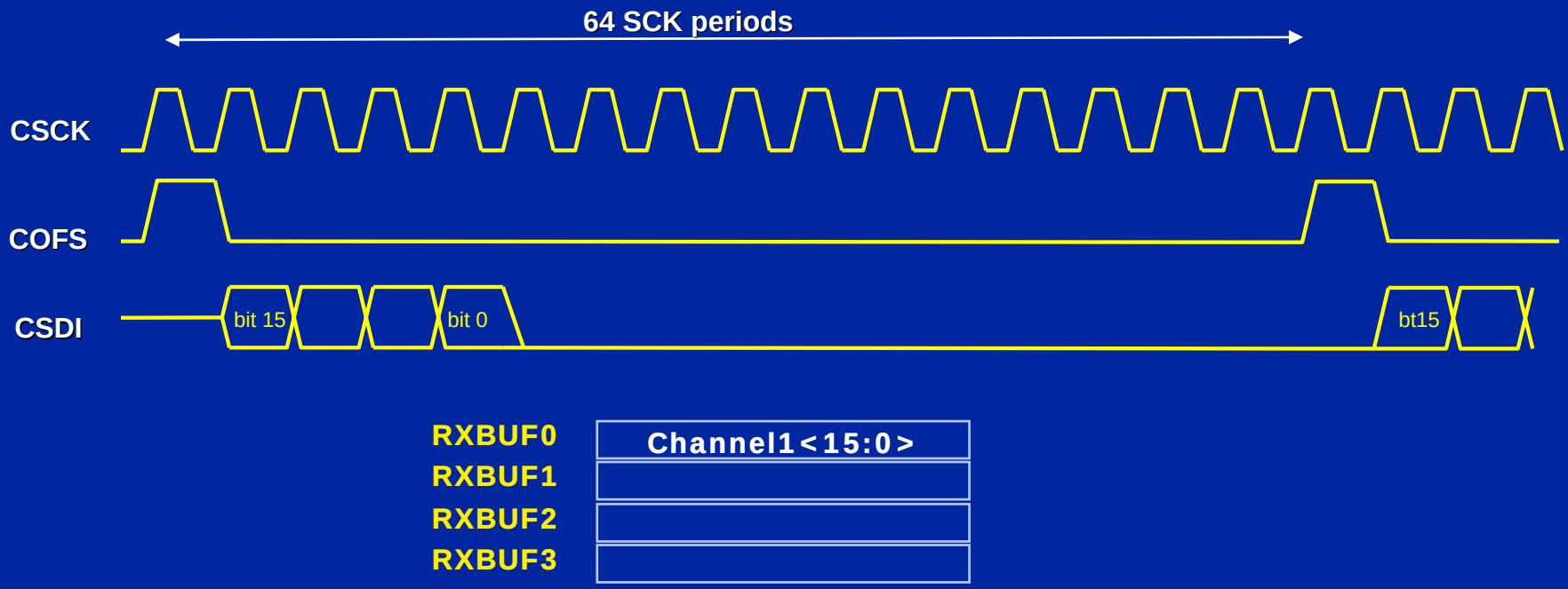
    COFSM = 00, COFSD = 0, CSCKD = 0
  - Setup 16-bit word, 4 words for 64 bits / frame

    WS = 1111, COFSG = 0011
  - Use first slot to send/receive

    RSCON = TSCON = 0x0001
  - Set baud rate so SCK = 64 * audio sampling rate

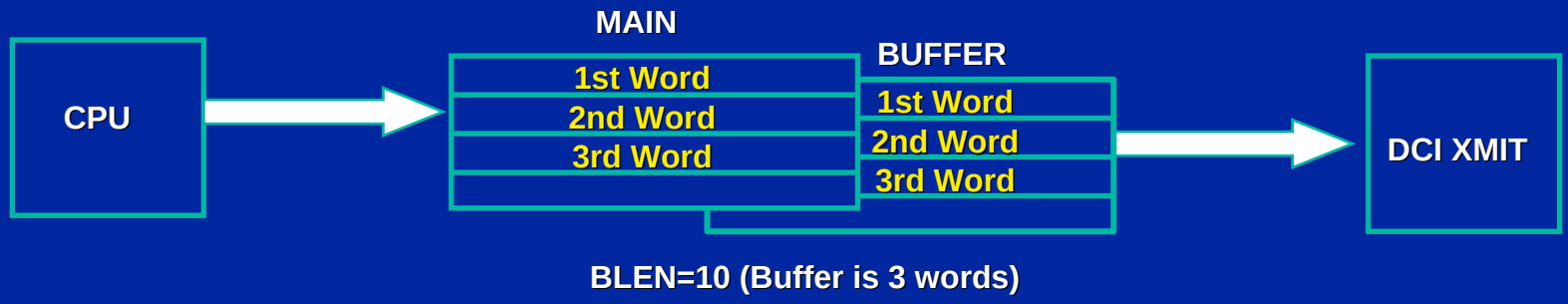
    DCICON3 = fcy / (2*64*44100) - 1

Slot Configuration Example Timing



DCI Buffer

- The BLEN bits specify the size of the buffer
  - Maximum is 4 words
- Each entry in the register is double buffered
- Software writes to main register, at interrupt, main register transferred to buffer, then used by DCI transmitter.



Hands-on Lab #2: Part #2

Code Requirements - Lab 2, Part 2

- Configure DCICON1
  - DCI to operate in Multi-channel / Slot mode
  - DCI in Master mode
- Configure DCICON2
  - All 4 buffers used
  - 16 time-slots/frame and 16-bit words/time-slot
- Configure DCICON3
  - FS = 7200 Hz
  - SCK = 256*FS

Code Requirements - Lab 2, Part 2

- Configure TSCON
  - Transmit on Time Slot #0
- Enable DCI (do this last)
- In the DCI ISR
  - Load the DCI buffers
- In the main.s file, call the routine to set up the DCI module
- In all there are 8 lines of code to write

Objectives - Lab 2, Part 2

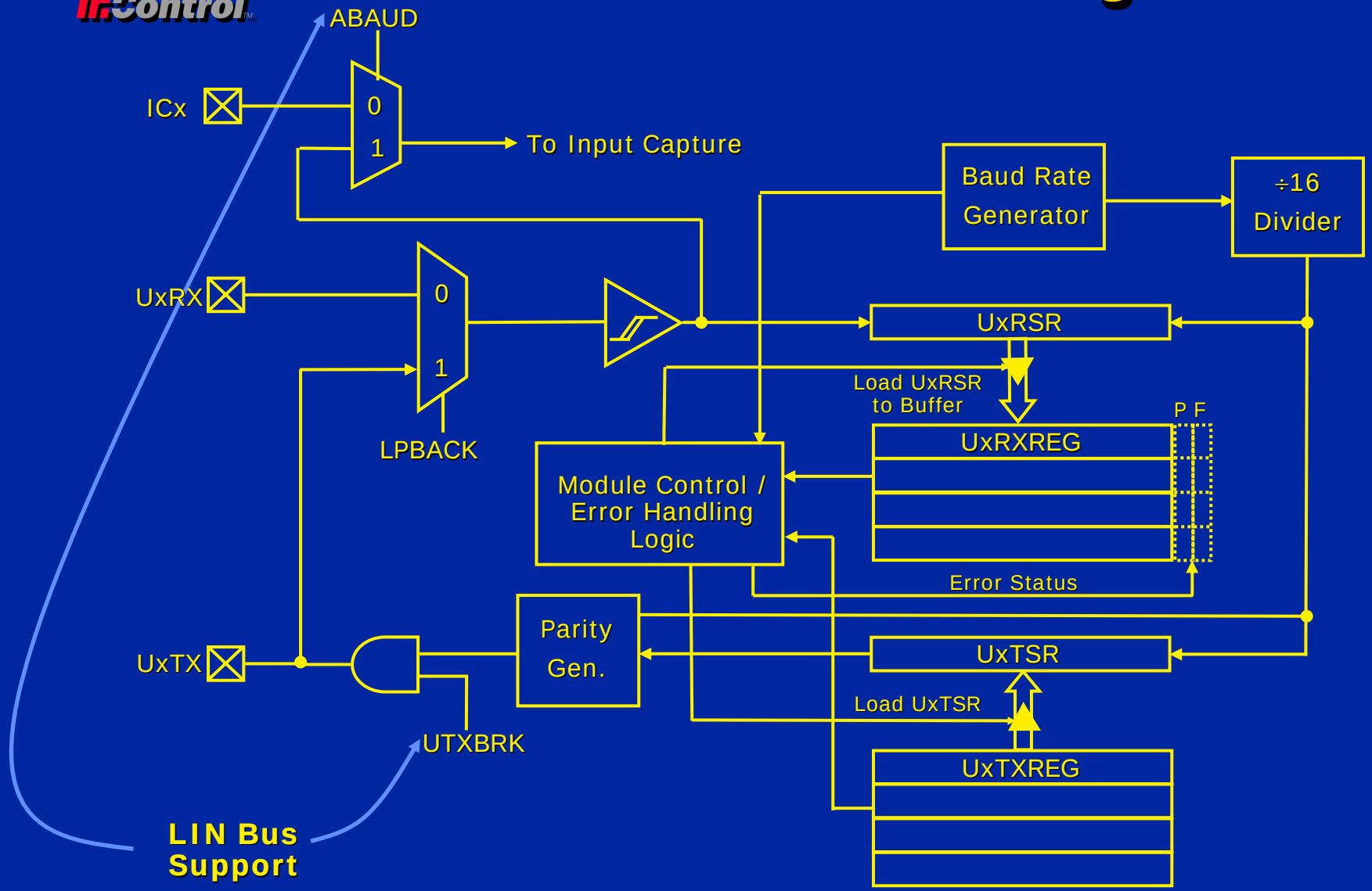
- Develop basic understanding on configuring DCI module
- Why did we choose 7200 Hz as the FS frequency?
- Why did we choose 256*FS for the serial clock frequency?

UART Module

UART - Overview

- Serial transmission and reception of 8/9-bit data
  - Full-duplex, asynchronous communication
  - Support for communication protocols such as RS-232, RS-422, RS-485 and LIN
  - 4-deep Transmit and Receive buffers
  - Transmit and Receive interrupts
  - Error detection
  - Support for receiver addressing
  - Loopback mode
  - Alternate TX/RX pins on some devices
  - Wake-up from SLEEP

UART - Block Diagram



UART - Baud Rate Generator

- Dedicated 16-bit Baud Rate Generator
  - Controlled by UxBRG register
  - **Baud Rate = Fcy / (16 * (UxBRG + 1))**

    where Fcy = Instruction Cycle Frequency
- Both transmitting and receiving devices must use same Baud Rate

UART - Transmission

- UART module is enabled by setting the UARTEN bit in the UxMODE register

R/W-0	U-0	R/W-0	U-0	U-0	R/W-0	U-0	U-0
UARTEN	-	USIDL	-	-	ALTIO	-	-
bit15	14	13	12	11	10	9	bit8

- Transmission starts only when:
  - The data to be transmitted is written to the buffer, AND
  - The UTXEN bit in the UxSTA register is set

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R-0	R-1
UTXISEL	-	-	-	UTXBRK	**UTXEN**	UTXBF	TRMT
bit15	14	13	12	11	10	9	bit8

UART - Transmission (contd.)

- The first bit transmitted is a START bit
  - Low level on UxTX pin for 1 bit time
- Next, the actual data bits are sent
  - LSB first , MSB later and parity bit last
  - Data format (8 or 9 bits) and parity type (even, odd or no parity) configured by PDSEL bits in the UxMODE register
  - No parity for 9-bit data

R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0
WAKE	LPBACK	ABAUD	-	-	**PDSEL1**	**PDSEL0**	STSEL
bit7	6	5	4	3	2	1	bit0

UART - Transmission (contd.)

- The last bit transmitted is a STOP bit
  - High level on UxTX pin for 1 or 2 bit times
  - Number of STOP bits configured by STSEL bit in the UxMODE register

R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0
WAKE	LPBACK	ABAUD	-	-	PDSEL1	PDSEL0	**STSEL**
bit7	6	5	4	3	2	1	bit0

- TRMT status bit in the UxSTA SFR
  - Bit is cleared if Transmit Shift Register (UxTSR) is busy or a transmission is pending

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R-0	R-1
UTXISEL	-	-	-	UTXBRK	UTXEN	UTXBF	**TRMT**
bit15	14	13	12	11	10	9	bit8

UART - Transmit Buffers

- 4-deep Transmit FIFO Buffer
  - Only the first character in the buffer is memory-mapped and thus user-accessible, UxTXREG
  - Characters in the buffer are shifted out of the buffer through UxTSR in FIFO
  - All 8 (or 9) data bits, are buffered
  - The UTXBF status bit in the UxSTA register indicates whether the buffer is full

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R-0	R-1
UTXISEL	-	-	-	UTXBRK	UTXEN	**UTXBF**	TRMT
bit15	14	13	12	11	10	9	bit8

UART - Transmit Interrupts

- Transmit Interrupt indicated by UxTXIF bit and enabled by UxTXIE bit
  - Set UTXISEL bit in the UxSTA register to 1
    - ✓ For transmitting a block of 4 characters per interrupt
  - Clear UTXISEL bit to 0
    - ✓ Used for transmitting a single character per interrupt
    - ✓ In this mode, an interrupt is generated as soon as the UTXEN bit is set

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R-0	R-1
UTXISEL	-	-	-	UTXBRK	UTXEN	UTXBF	TRMT
bit15	14	13	12	11	10	9	bit8

Hands-on Lab #2: Part #3

Code Requirements - Lab 2, Part 3

- We will use the UART2 module
- Configure U2BRG
  - Set the Baud Rate to 57600 baud
- Configure U2MOD
  - Enable the UART2 module for 8-bit data, No Parity and 1 Stop bit
- Configure U2STA
  - Set up UART2 so that all 4 buffers in the UART FIFO are transferred to the shift register before the module sets the TRMT bit

Code Requirements - Lab 2, Part 3

- Enable the UART2 Transmitter
- In the “display_string” subroutine:
  - Load the UART2 transmit buffer
- Call the UART routines from the main.s file
- In all there are 6 lines of code to write and 1 formula to compute using assembler directives

Objectives - Lab 2, Part 3

- Develop basic understanding on configuring UART module
- Implement byte move instructions using file-register addressing

System Reliability, Clocking and Power Management

Power Management Options

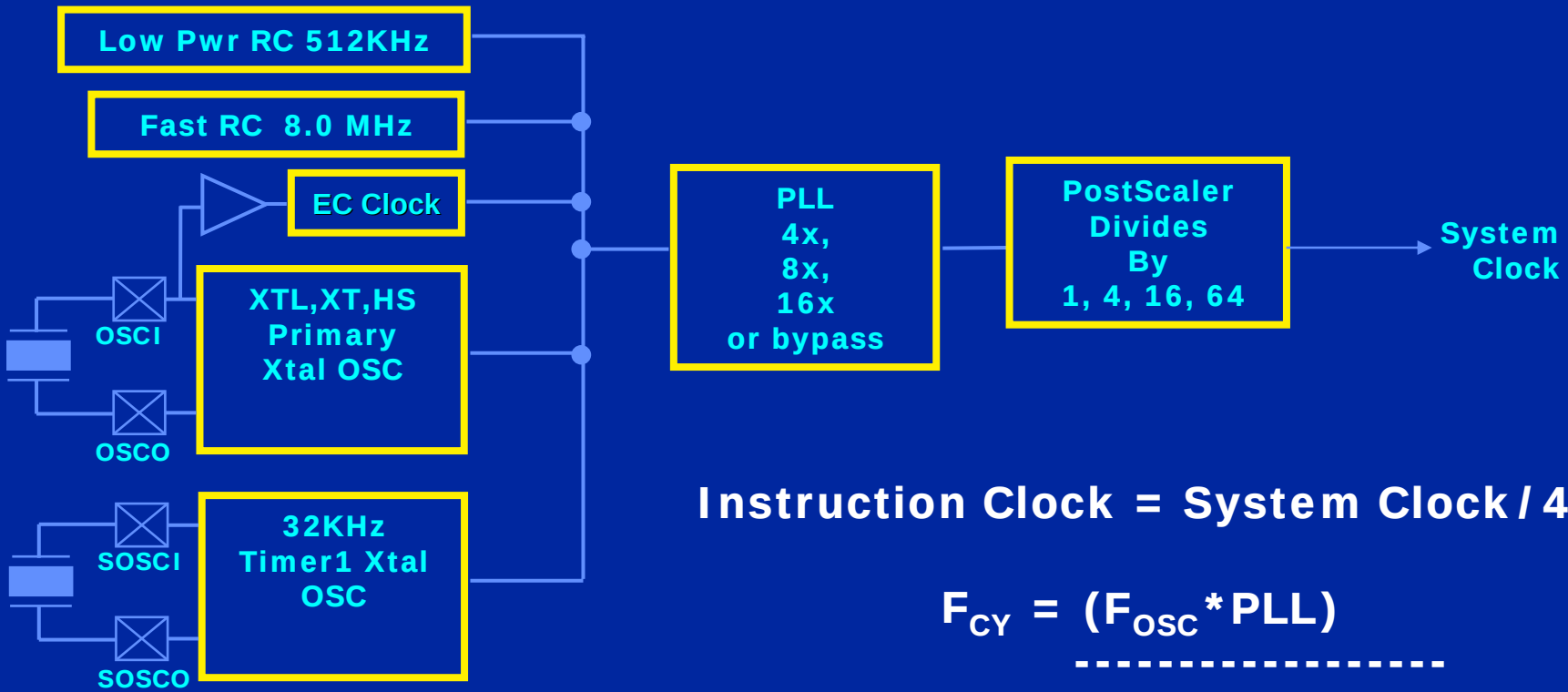
- Switch between clocks at run time
- Divide down system clock: 1, 4,16, 64
- SLEEP mode:
  - System clock is shut off
  - CPU is stopped
  - Peripherals are stopped, but may keep running
  - Wake up on interrupt or I/O pin change

Power Management Options

- IDLE mode:
  - System clock is running
  - CPU is shut off
  - Peripherals are running, but can be shut off
  - Reduce system clock speed for peripherals

Clock sources

- Includes 2 Internal RC Oscillators
- PLL multiplies oscillator source for high frequency operation



Instruction Clock = System Clock / 4

$$F_{CY} = \frac{(F_{OSC} * PLL)}{(4 * Post\ Scaler)}$$

Oscillator Control Register

- OSCCON
  - COSC<1:0> ⇒ Current Oscillator Status
  - NOSC<1:0> ⇒ New Oscillator Group Selection
  - POST<1:0> ⇒ Oscillator Postscaler Selection
  - LOCK ⇒ PLL Lock Status
  - CF ⇒ Clock Failure Status
  - LPOSCEN ⇒ 32 KHz LP Oscillator Enable
  - OSWEN ⇒ Oscillator Switch Enable

y= set value on POR /BOR

U-0	U-0	R-y	R-y	U-0	U-0	R/W-y	R/W-y
-	-	COSC<1:0>		-	-	NOSC<1:0>	
bit15	14	13	12	11	10	9	bit8

R/W-0	R/W-0	R-0	U-0	R-0	U-0	R/W-0	R/W-0
POST<1:0>		LOCK	-	CF	-	LPOSCEN	OSWEN
bit7	6	5	4	3	2	1	bit0

Oscillator Configuration Register

- FOSC
  - FCKSM<1:0> ⇒ Clock Switching Mode Selection
  - FOS<1:0> ⇒ Oscillator Source Selection on POR
  - FPR<3:0> ⇒ Primary Oscillator Selection bits

U	U	U	U	U	U	U	U
-	-	-	-	-	-	-	-
bit23	22	21	20	19	18	17	bit16

R/P	R/P	U	U	U	U	R/P	R/P
FCKSM<1:0>		-	-	-	-	FOS<1:0>	
bit15	14	13	12	11	10	9	bit8

U	U	U	U	R/P	R/P	R/P	R/P
-	-	-	-	FPR<3:0>			
bit7	6	5	4	3	2	1	bit0

Oscillator Configuration Register

- FOSC
  - FCKSM<1:0> ⇒ Clock Switching Mode Selection
  - FOS<1:0> ⇒ Oscillator Source Selection on POR
  - FPR<3:0> ⇒ Primary Oscillator Selection bits

U	U	U	U	U	U	U	U
-	-	-	-	-	-	-	-
bit23	22	21	20	19	18	17	bit16

R/P	R/P	U	U	U	U	R/P	R/P
FCKSM<1:0>		-	-	-	-	FOS<1:0>	
bit15	14	13	12	11	10	9	bit8

U	U	U	U	R/P	R/P	R/P	R/P
-	-	-	-	FPR<3:0>			
bit7	6	5	4	3	2	1	bit0

Oscillator Configuration Register

- FOSC
  - FCKSM<1:0> ⇒ Clock Switching Mode Selection
  - FOS<1:0> ⇒ Oscillator Source Selection on POR
  - FPR<3:0> ⇒ Primary Oscillator Selection bits

U	U	U	U	U	U	U	U
-	-	-	-	-	-	-	-
bit23	22	21	20	19	18	17	bit16

R/P	R/P	U	U	U	U	R/P	R/P
FCKSM<1:0>		-	-	-	-	FOS<1:0>	
bit15	14	13	12	11	10	9	bit8

U	U	U	U	R/P	R/P	R/P	R/P
-	-	-	-	FPR<3:0>			
bit7	6	5	4	3	2	1	bit0

Setting FOSC

- Use the config __FOSC macro defined in the p30f6014.inc file
  - For Example:

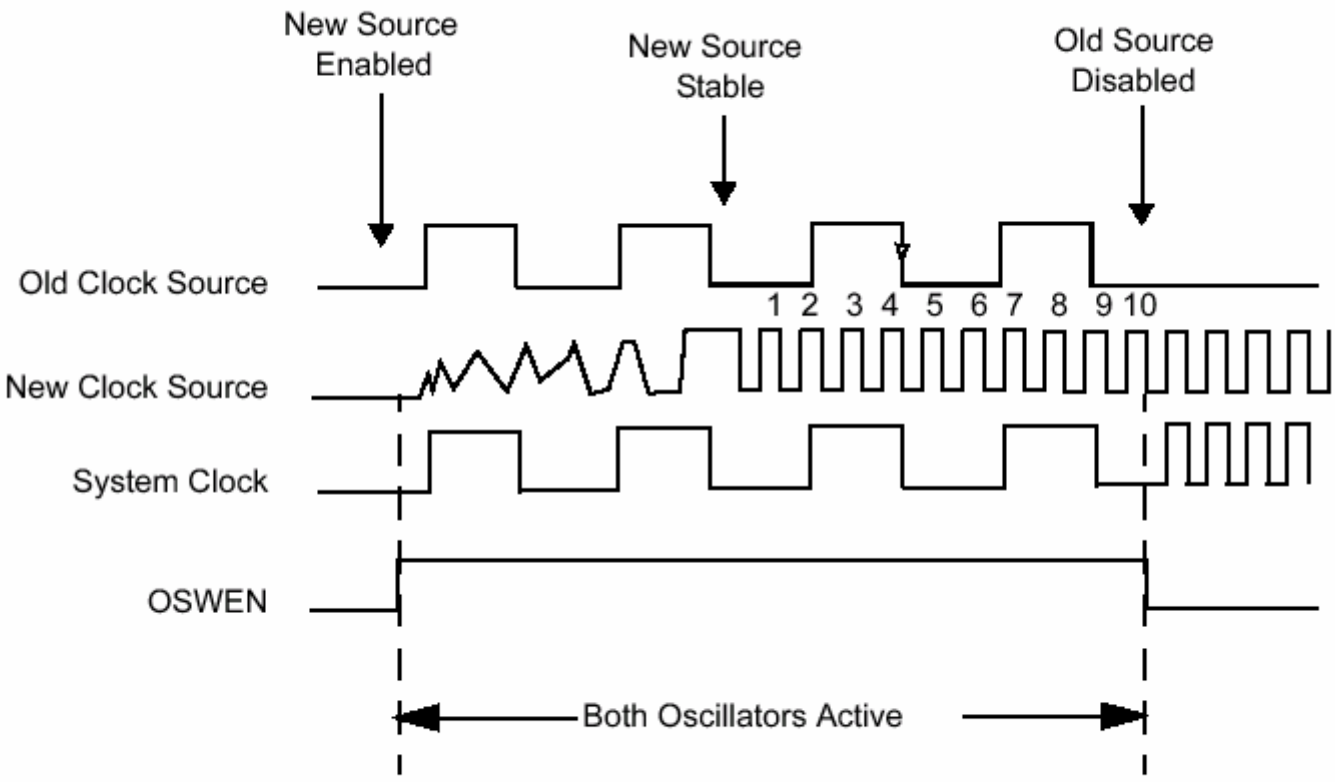
    config __FOSC, CSW_FSCM_ON & EC_PLL8

  OR
- Use the Configure > Configuration Bits dialog in MPLAB® IDE

Clock Switching

- Clock switching can assist you manage overall device performance, mWatt/MIPS
- Switch between device oscillator modes for:
  - Maximizing CPU throughput
  - Minimizing device current draw
- Can not switch directly between PLL modes
  - Switch first to the FRC Oscillator then to new PLL
- Code execution continues during clock switch sequence

Clock Transition Timing Diagram



Note: The system clock can be any selected source – Primary, Secondary, FRC or LPRC.

Performing a Clock Switch

- Ensure Clock Switching is enabled
  - FCKSM bits in the FOSC configuration register
- When clock switching is desired
  - “Unlock” the OSCCONH byte
  - Write to the NOSC bits in OSCCONH
  - “Unlock” the OSCCONL byte
  - Set OSWEN bit in OSCCONL
  - Wait for OSWEN bit to be cleared
    - ✓ If OSWEN is not cleared then abort the clock switch by clearing OSWEN

OSCCON Unlock Sequence

```
DISI   #14                    ; disable level 0-6 Interrupts
mov.b   #0x??,  w0
mov     #OSCCONH, w1
mov     #0x78,  w2
mov     #0x9A, w3
mov.b   w2,  [w1]
mov.b   w3,  [w1]
mov.b   wreg, OSCCONH    ; 1 cycle window for byte write
```

Required Sequence!

```
; low byte unlock sequence and initiate some action
mov     #OSCCONL, w1
mov.b  #0x??,  w0
mov     #0x46,  w2
mov     #0x57,  w3
mov.b   w2,   [w1]
mov.b   w3,  [w1]
mov.b   w0,  [w1]             ; 1 cycle window for byte write
```

Required Sequence!

Fail-Safe Clock Monitor (FSCM) for Reliable Operation

- What happens on Oscillator Failure?
  - Device switches automatically to the FRC oscillator if FSCM is enabled
    - FSCM controlled by the FCKSM bits in the FOSC
  - Oscillator Failure Trap is generated
  - Code execution vectors to the Oscillator Fail trap handler, if one exists. Here the application should:
    - Clear the Clock Fail bit, CF, in OSCCON
    - Clear the OSCFAIL trap flag in INTCON1
    - Execute a RETFIE

Hands-on Lab #3

Code Requirements - Lab 3 Part 1

- Develop code for INT1 - INT4 interrupts:
  - INT1 Interrupt - Clock switch to ECxPLL8
  - INT2 Interrupt - Clock switch to FRC
  - INT3 Interrupt - Clock switch to LPRC
  - INT4 Interrupt - Toggles Post-scaler between "divide by 1" and "divide by 4" modes
- Develop OSCCON SFR write sequence code
  - Required to command a clock switch
- In all there are 21 lines of code to write !

Objectives - Lab 3 Part 1

- Understand clock switch code sequence
- Implement a few more Move instruction types:
  - Byte addressing mode
  - Word addressing mode
  - Immediate addressing mode
- Understand clock selection on POR and MCLR
- Understand what happens when the desired new oscillator is non-existent or has failed to start

Code Requirements - Lab 3 Part 2

- Develop code for Oscillator Failure Trap handler:
  - Perform a fast context save of W0 - W3
  - Develop OSCCON SFR write sequence code to clear the CF bit in the OSCCONL byte
  - Clear the OSCFAIL trap flag bit in INTCON1
- In all there are 3 lines of code to write !

Objectives - Lab 3 Part 2

- Understand how often the PUSH.S and POP.S instructions may be used in an application
- Understand device operation in the event of a clock failure

Output Compare Module

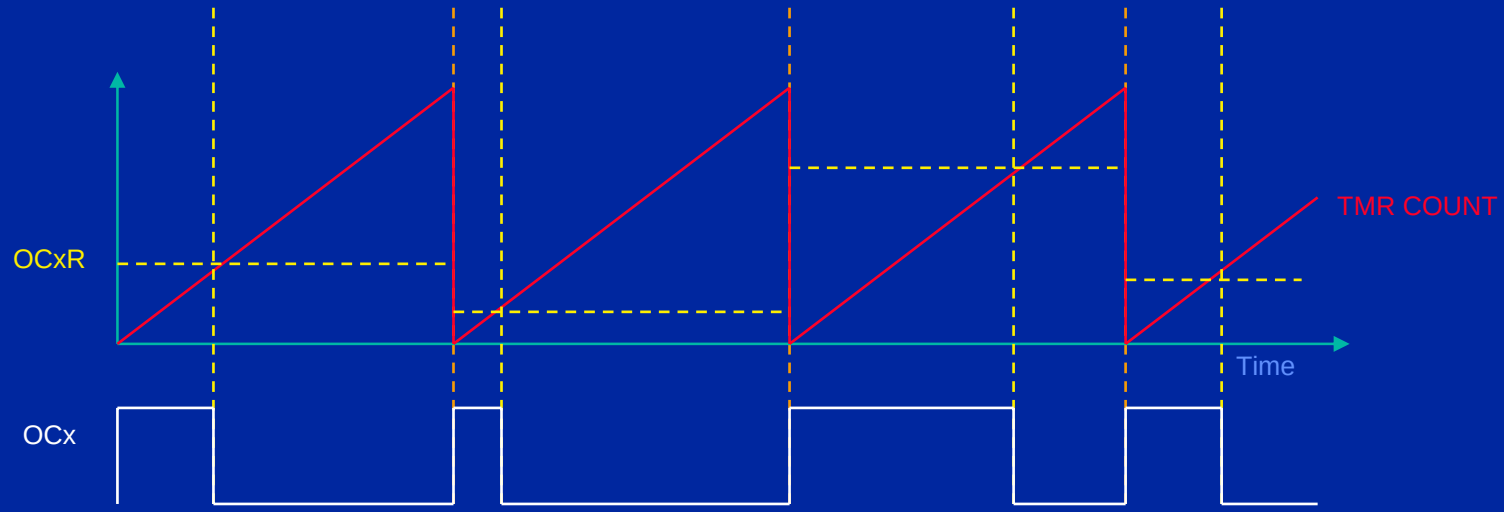
Output Compare Module

- Up to 8 Output Compare / PWM Channels
- Simple Compare Mode:
  - 16-bit Compare
  - Resolution = Instruction Cycle
  - Set, Reset of Toggle Pin
- Dual Compare Mode:
  - Single Pulse
  - Continuous pulse
- Timer 2 or Timer 3 as time-base

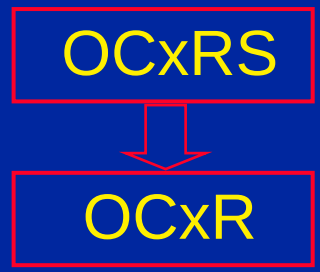
Output Compare Module

- PWM mode
  - 16-bit glitch-less (double buffered) PWM output
  - Full range of 0 to 100% duty cycle
  - Wide frequency range (very low to very high)
  - Selectable PWM shutdown on fault detection
    - This is an Asynchronous shutdown

OC PWM Mode



On timer period match
OCx set
OCxRS copied to OCxR
On OCxR match
OCx clear



Input Capture Module

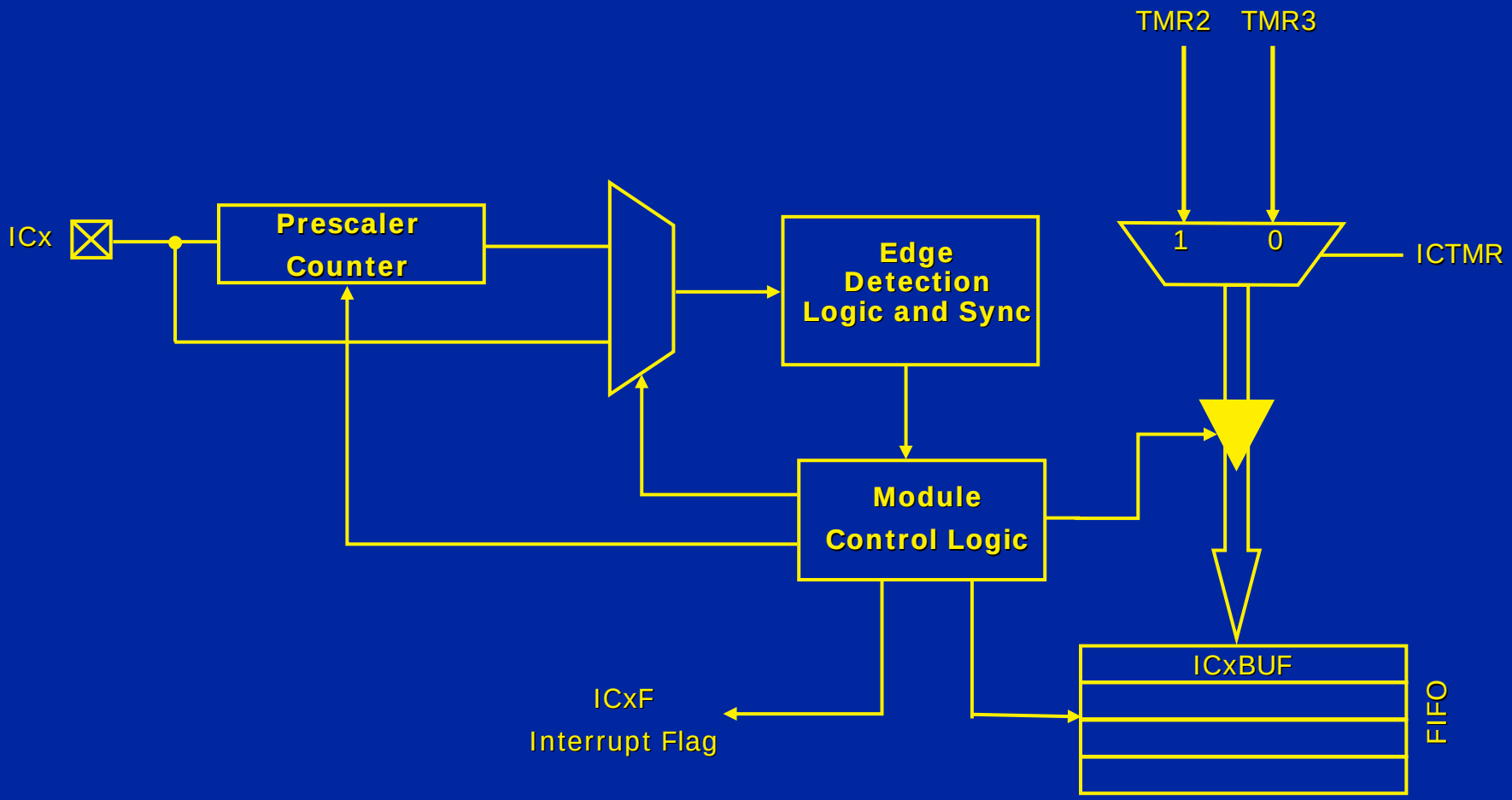
Input Capture

- Up to Eight Input Capture Channels
- Captures 16-bit timer value
  - At 30 MIPS resolution = 33 ns (Tcy)
  - At 30 MIPS with 16x pre-scale = 2.1 ns
- 4 deep buffer for each capture input
  - Interrupt on 1- 4 capture events
  - FIFO buffer overflow status
  - FIFO buffer empty status

Input Capture

- Timer 2 or Timer 3 as timebase
- Capture on:
  - ↑ edge
  - ↓ edge
  - Every 4th ↑ edge
  - Every 16th ↑ edge
  - ↑ edge and ↓ edge
    - Very useful for pulse and frequency measurement
    - Interface to hall sensors for rotor position feedback
    - Autobaud support for UART communications

Input Capture



Input Capture Control Register

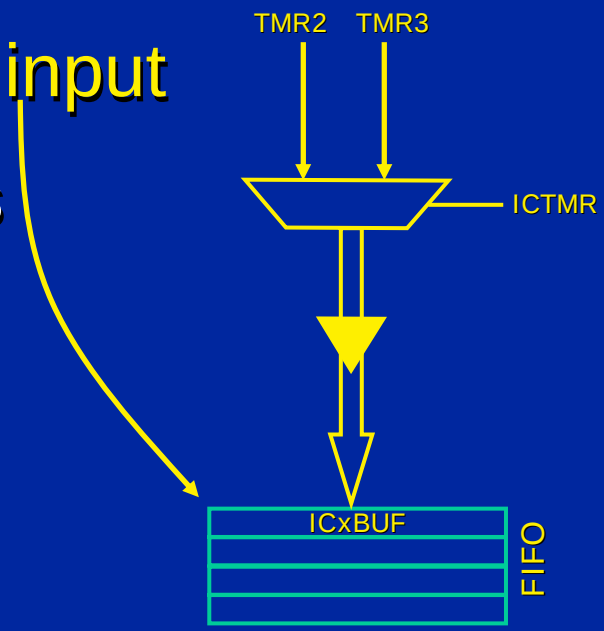
- ICxCON
  - ICSIDL ⇒ Stop in Idle mode
  - ICTMR ⇒ Time Base Select for Input Capture
  - ICI<1:0> ⇒ Capture events per Interrupt select
  - ICOV ⇒ FIFO buffer overflow status
  - ICBNE ⇒ FIFO buffer Not Empty status
  - ICM<2:0> ⇒ Input Capture mode select

U-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
-	-	ICSIDL	-	-	-	-	-
bit15	14	13	12	11	10	9	bit8

R/W-0	R/W-0	R/W-0	R-0, HC	R-0, HC	R/W-0	R/W-0	R/W-0
ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
bit7	6	5	4	3	2	1	bit0

Input Capture FIFO

- 4 deep buffer for each capture input
- Interrupt on 1-4 capture events
- FIFO buffer overflow status
- FIFO buffer empty status



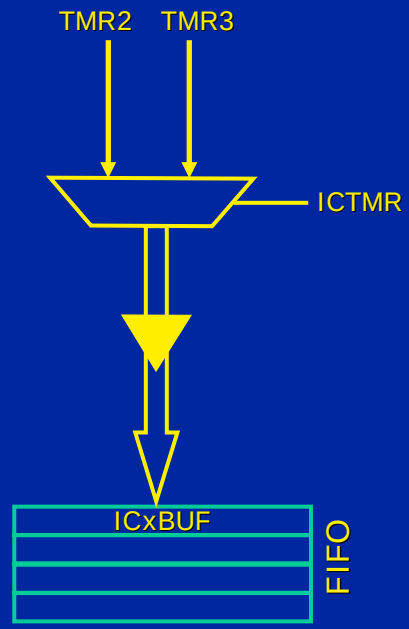
ICxCON SFR

-	-	ICSIDL	-	-	-	-	-
bit15	14	13	12	11	10	9	bit8

ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
bit7	6	5	4	3	2	1	bit0

Input Capture FIFO

- 4 deep buffer for each capture input
- Interrupt on 1-4 capture events
- FIFO buffer overflow status
- FIFO buffer empty status



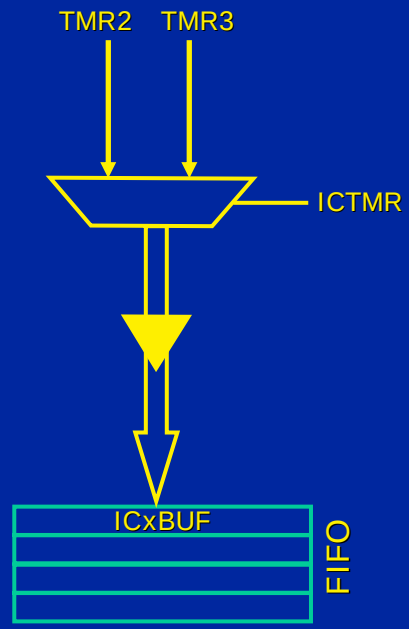
ICxCON SFR

-	-	ICSIDL	-	-	-	-	-
bit15	14	13	12	11	10	9	bit8

ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
bit7	6	5	4	3	2	1	bit0

Input Capture FIFO

- 4 deep buffer for each capture input
- Interrupt on 1-4 capture events
- FIFO buffer overflow status
- FIFO buffer empty status



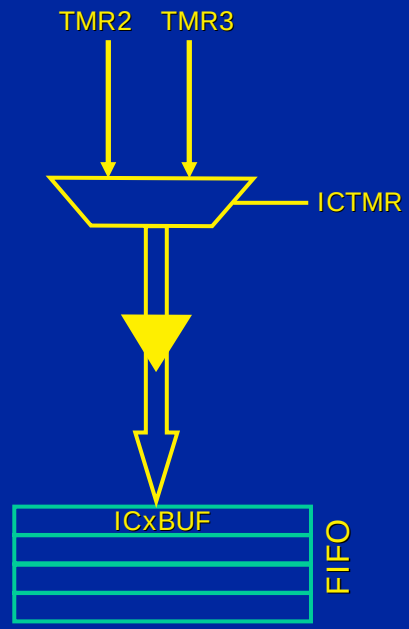
ICxCON SFR

-	-	ICSIDL	-	-	-	-	-
bit15	14	13	12	11	10	9	bit8

ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
bit7	6	5	4	3	2	1	bit0

Input Capture FIFO

- 4 deep buffer for each capture input
- Interrupt on 1-4 capture events
- FIFO buffer overflow status
- FIFO buffer not empty status



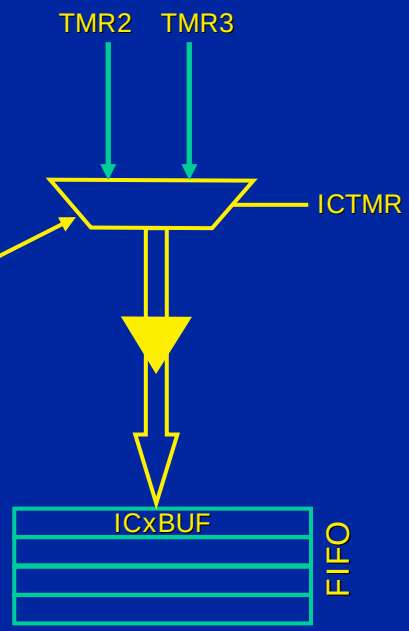
ICxCON SFR

-	-	ICSIDL	-	-	-	-	-
bit15	14	13	12	11	10	9	bit8

ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
bit7	6	5	4	3	2	1	bit0

Input Capture FIFO

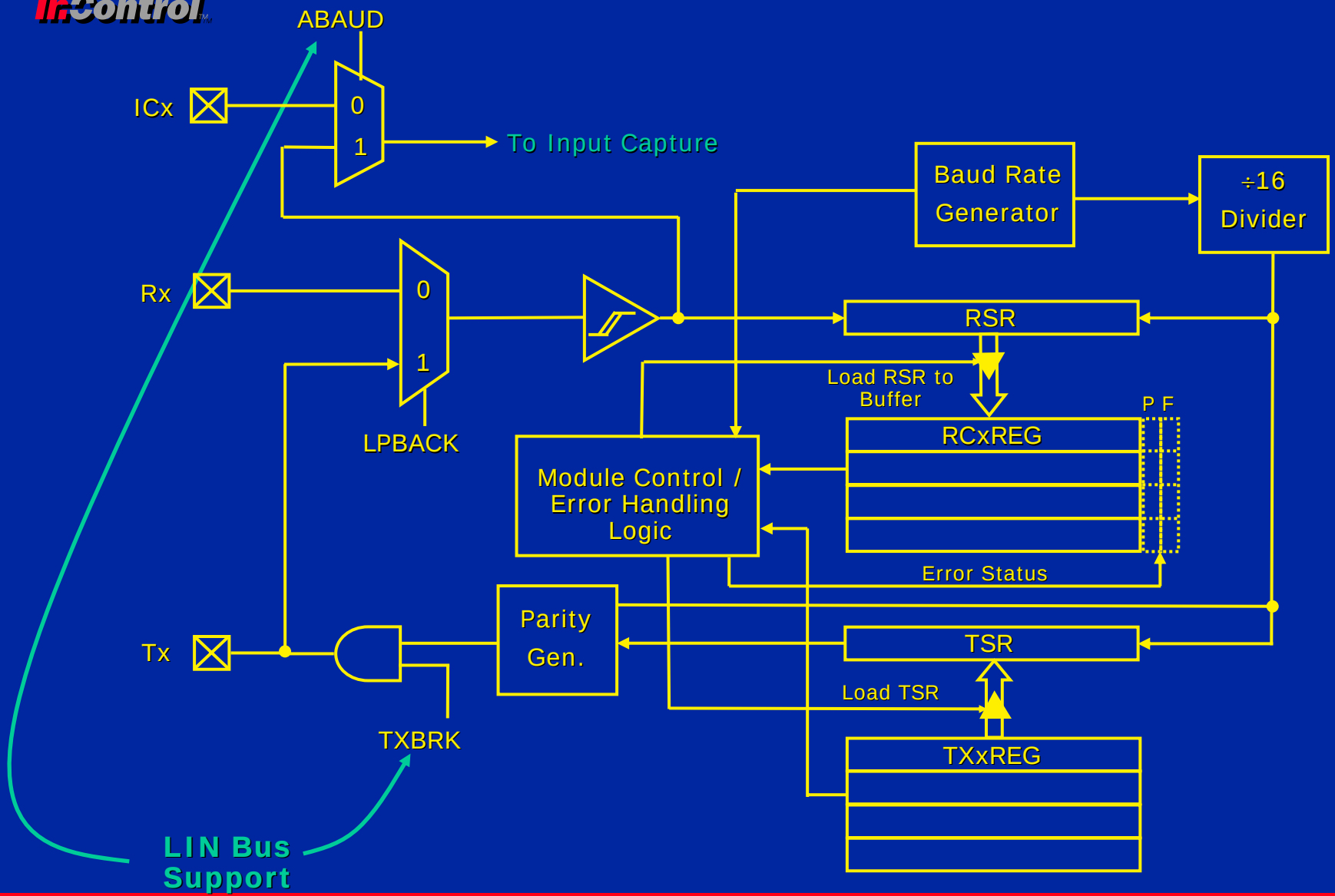
- 4 deep buffer for each capture input
- Interrupt on 1-4 capture events
- FIFO buffer overflow status
- FIFO buffer empty status



-	-	ICSIDL	-	-	-	-	-
bit15	14	13	12	11	10	9	bit8

ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
bit7	6	5	4	3	2	1	bit0

UART Block Diagram



Hands-on Lab #4

Code Requirements - Lab 4, Part 1

- Configure IC6CON
  - Interrupt on 4 captures
  - Capture on every rising edge
  - Select Timer 3 for timebase
- Within Interrupt Handler
  - Move all FIFO buffer contents to w0 - w3
  - Compute average frequency with DIV instruction
- In all there are 11 lines of code to write

Objectives - Lab 4, Part 1

- Develop basic understanding on Input Capture
  - Implementing FIFO buffer
  - Capture on every rising edge is useful for measuring frequencies
- Continue using Instruction set
  - MOV instruction
  - DIV.UD instruction
  - REPEAT instruction

Code Requirements - Lab 4, Part 2

- Configure IC6CON
  - Capture on every edge
  - Select Timer 3 for timebase
- Within Alternate Interrupt Handler
  - Move two FIFO buffer contents to w0 and w1
  - Compute difference of time with SUB instruction
- In all there are 8 lines of code to write

Objectives - Lab 4, Part 2

- Continue working with Input Capture module
  - Read FIFO buffer
  - Capture on every edge is useful for measuring pulse width
- Continue using Instruction set
  - MOV instruction
  - SUB instruction

Objectives - Lab 4, Part 2

- Define Alternate Input Capture ISR
  - Move first capture event to ram variable
  - Move second capture event to ram variable
  - Compute difference between capture events
- Adjust RP1 and view LCD

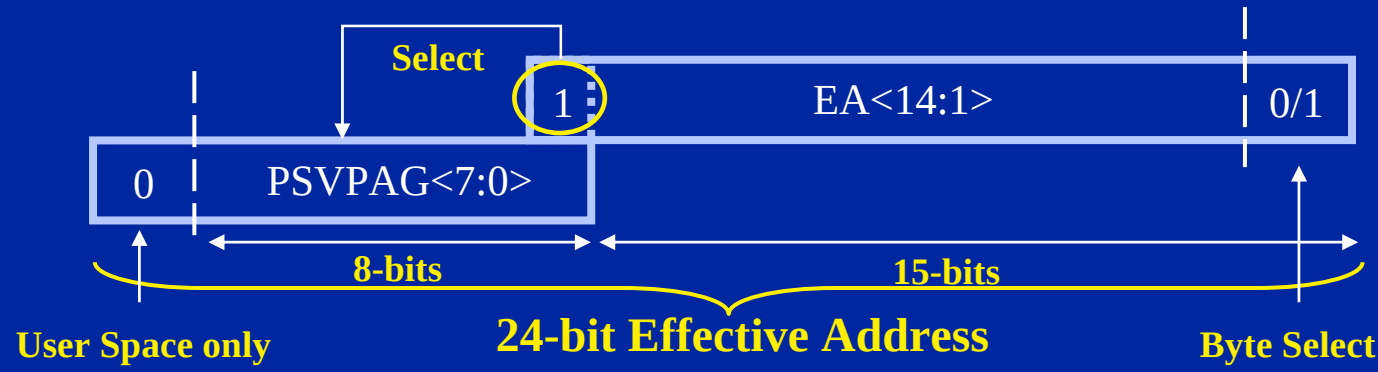
Program Space Visibility - (PSV)

Program Memory Organization

- Three methods for accessing PS words
  - via the 23-bit PC
  - via Table instructions (TBLRD and TBLWT)
  - Mapping a 32 KB segment of PS into the data memory address space
    (a.k.a. Program Space Visibility, PSV)

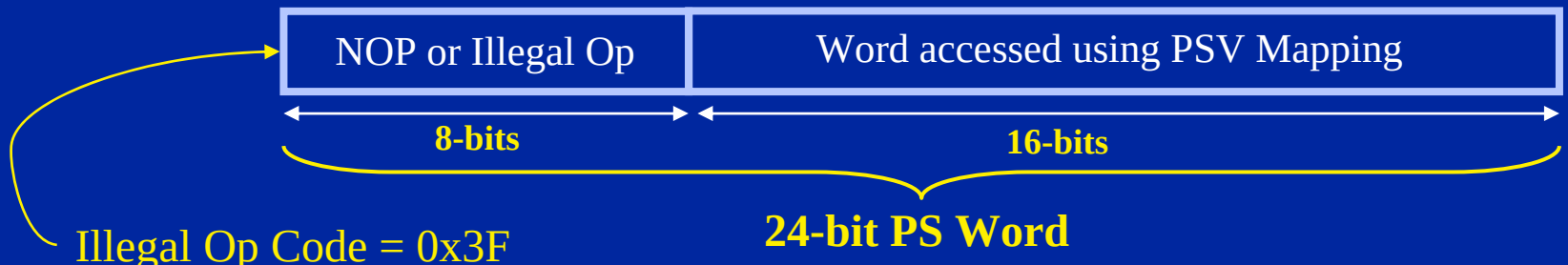
Data Access from PS using Program Space Visibility

- 32 KB segment of PS may be mapped into the data memory address space
  - If PSV bit (CORCON<2>) is set and if EA<15> =1 then data is accessed from PS
  - PSVPAG (Program Space Visibility Page register) supplies the upper byte for the 24-bit PS address



Data Access from PS using Program Space Visibility

- Only lower 16-bits of PM location can be accessed via mapping
  - Upper 8 bits should be programmed for an illegal instruction or contain a NOP



Defining Constants for PSV

```
                    ;Define constants in PM
                    .section .myconst,  "r”
MyConstString:  .asciz  "Northern Star”
```

What is the “r” attribute?

- Read only section dedicated for use in PSV window
- Linker will not allow read-only section to cross PSVPAG boundary
  - Boundary = 32 KB

PSV Addressing: Assembler operators

- Special Assembler operators
  - #psvpage and #psvoffset

```
;Initialize Page Boundary SFR
mov  #psvpage(MyConstString), w0
mov  w0, PSVPAG

;Initialize PSV page EA<15:0> pointer
mov  #psvoffset(MyConstString),  w0
```

PSV Addressing: Example

- Enable PSV
  - `bset CORCON, #PSV`

Example of reading two words from PSV window

```
mov [w0++], w3 ;Read PM word into w3
mov [w0++], w4 ;Read next PM word into w4
```

Example of reading two bytes from PSV window

```
mov.b [w0++], w3    ;Read PM byte into w3
mov.b [w0++], w4    ;Read next PM byte
                    ;into w4
```

Runtime Library

- Become familiar with run-time libraries
- Run-Time Library crt0.s
  - Data initialization routine included
- Run-Time Library crt1.s
  - No data initialization

Special Operators

- Accessing data in PM
- Obtain PM address of constant or symbol
- Obtain handle to PM address

Operators	Description
`tblpage(name)`	Get page for table read/write operations
`tbloffset(name)`	Get pointer for table read/write operations
`psvpage(name)`	Get page for PSV data window operations
`psvoffset(name)`	Get pointer for PSV data window operations
`paddr(label)`	Get 24-bit address of `label` in program memory
`handle(label)`	Get 16-bit reference to `label` in program memory
`.sizeof.(name)`	Get size of section `name` in address units
`.startof.(name)`	Get starting address of section `name`

Data Access Overhead using PSV

- PSV data fetch overhead:
  - Outside a REPEAT loop:
    - ✓ Data move ops, overhead = 1 cycle
    - ✓ ALU based ops, overhead = 2 cycles
  - Within a REPEAT loop:
    - ✓ Data pipelined, so overhead for all ops = 0 cycles
    - ✓ First & last iteration - data pipeline fill/flush
      - Data move ops, overhead = 1 cycle
      - ALU based ops, overhead = 2 cycles

Usefulness of PSV

- PSV allows very large tables of data to be stored and accessed quickly & efficiently
- PSV provides a bridge to a common data/program address space (Von Neumann) but only when needed
- Example: Using PSV for digital filters (FIR)
  - Filter coefficients stored in PS
  - Saves valuable SRAM
  - Minimal performance impact (2 cycles/filter iteration)

Hands-on Lab #5

Code Requirements - Lab 5, Part 1

- Implement special PSV Assembler operators
  - Initialize PSVPAG
  - Initialize Page Offset Pointer
- Configure CORCON to enable PSV
- Copy PSV window string into DM
  - Use data move byte instruction

- In all there are 7 lines of code to write

Objectives - Lab 5, Part 1

- Understand basics on using PSV window
  - PSV String Definition
  - Enable PSV
  - Assembler operators
  - PSV data access
  - PSV data access overhead

Code Requirements - Lab 5, Part 2

- Open lib5.inc file and set the following:
  - .equ LAB5_PART1, 0
  - .equ LAB5_PART2, 1
- Optimize PM to DM string copy routine
  - Reduce 5 instructions to 2 instructions
  - Use hardware loop type instruction

Thank You