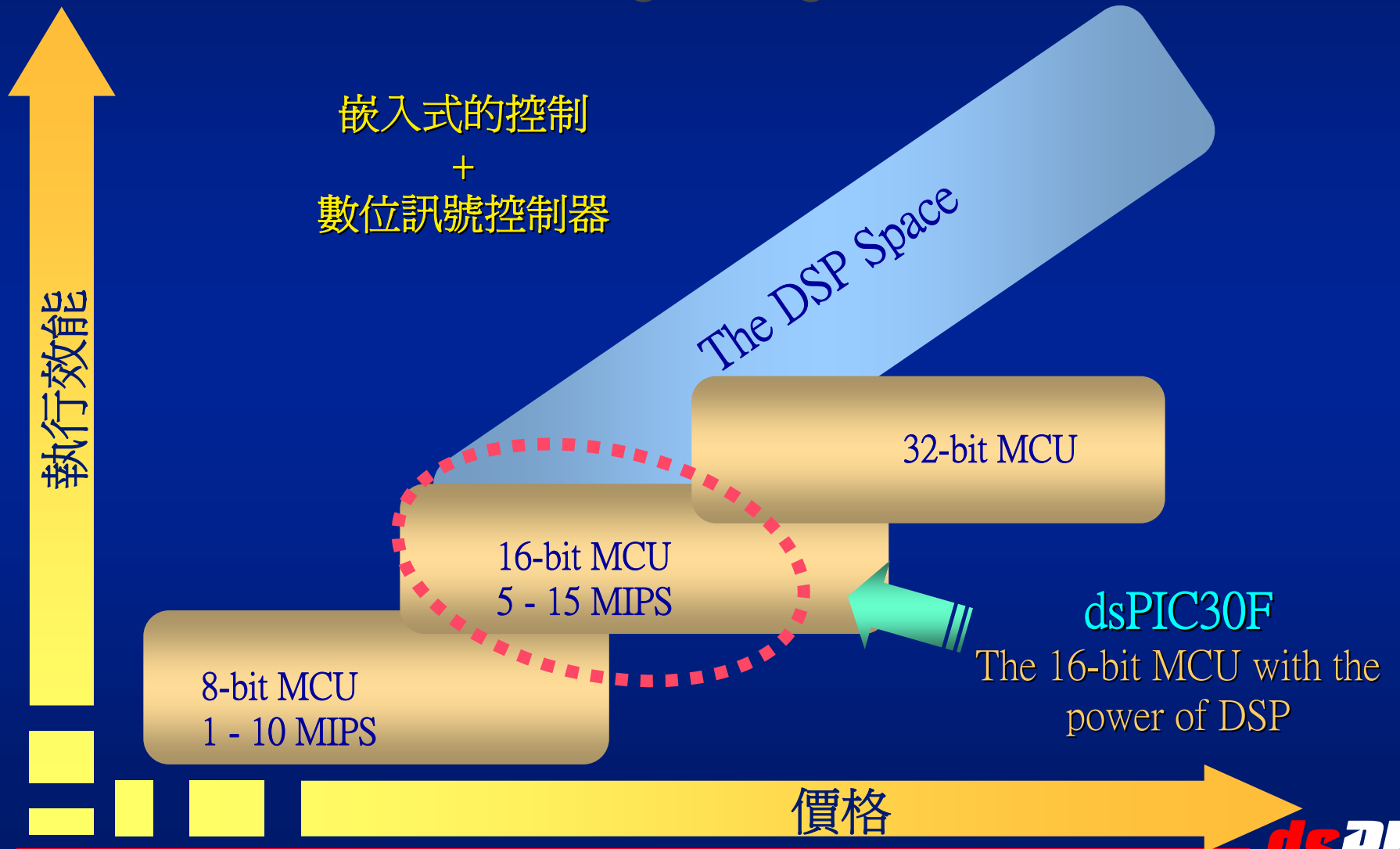
什麼是數位訊號控制器

Digital Signal Control



dsPIC30F

- 看起來像是一個 MCU
  - 使用簡單；內建功能齊全的周邊
- 執行效能又像一個 DSP
  - 內建功能強大的 DSP 運算核心
- MCU 般的價格

是一個內建 DSP 運算功能的 16 位元微處理器

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

dsPIC 課程摘要

- 本課程所涵蓋的內容
  - 程式記憶體的架構
  - 資料記憶體的架構
  - dsPIC 架構與運算核心
  - dsPIC 指令集
  - dsPIC 定址模式
  - 中斷功能
  - 使用 ASM30 & LINK30
  - 系統的可靠性設計

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

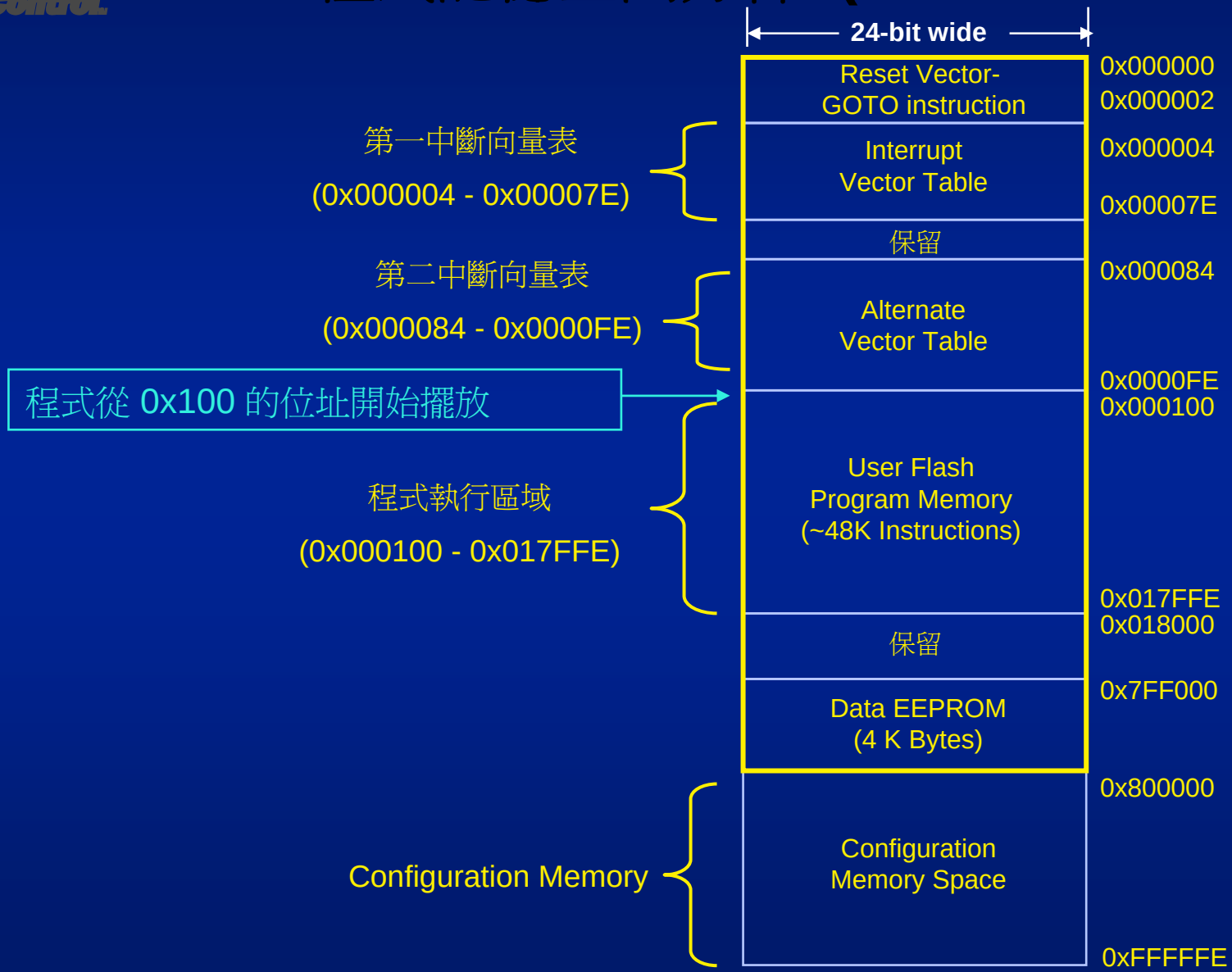
程式記憶體

Program Memory (PS)

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

程式記憶空間分佈 (dsPICC30F6014)

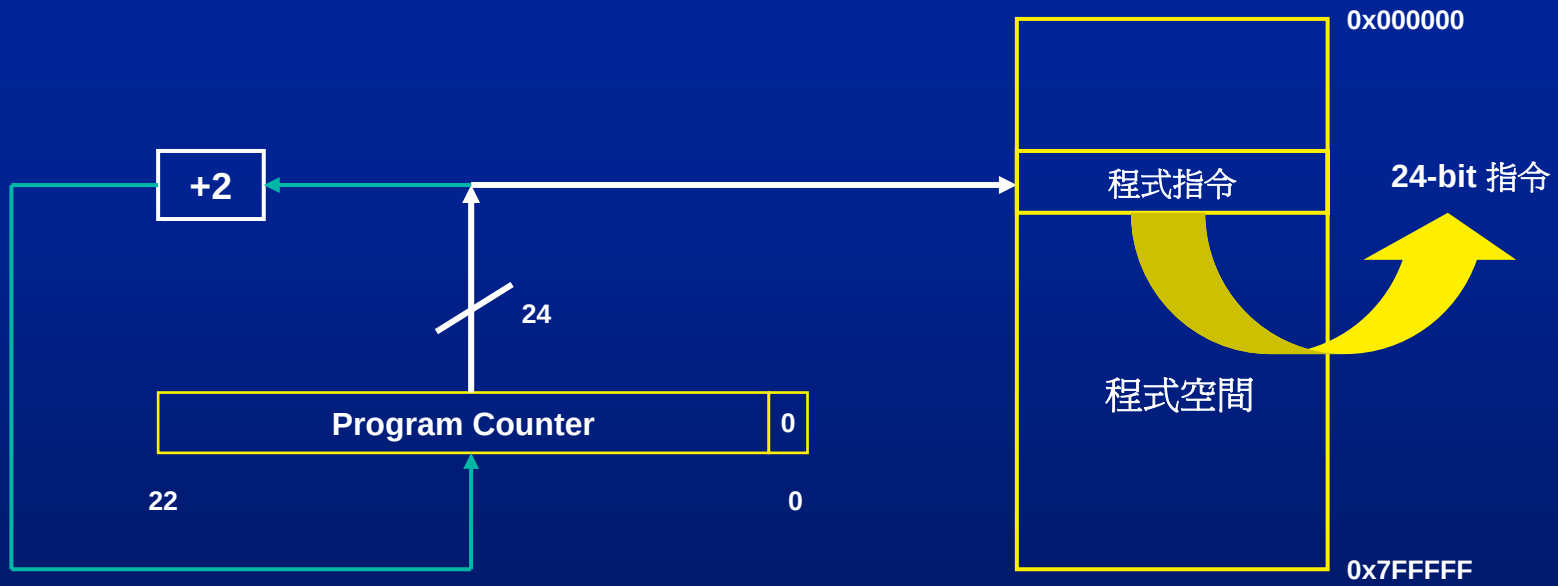


程式記憶體讀取的方式

- 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

使用程式計數器 (PC)

- 23-bit Program Counter (PC<22:0>)
  - 在指令提取模式下，PC<0> 將強制設爲 ‘0’ (偶數位址)
  - PC 每次會自動加 2 以提取下一個指令
    - PC 加二，所提取的是 32-bit 的資料但只有 24-bit 有效。



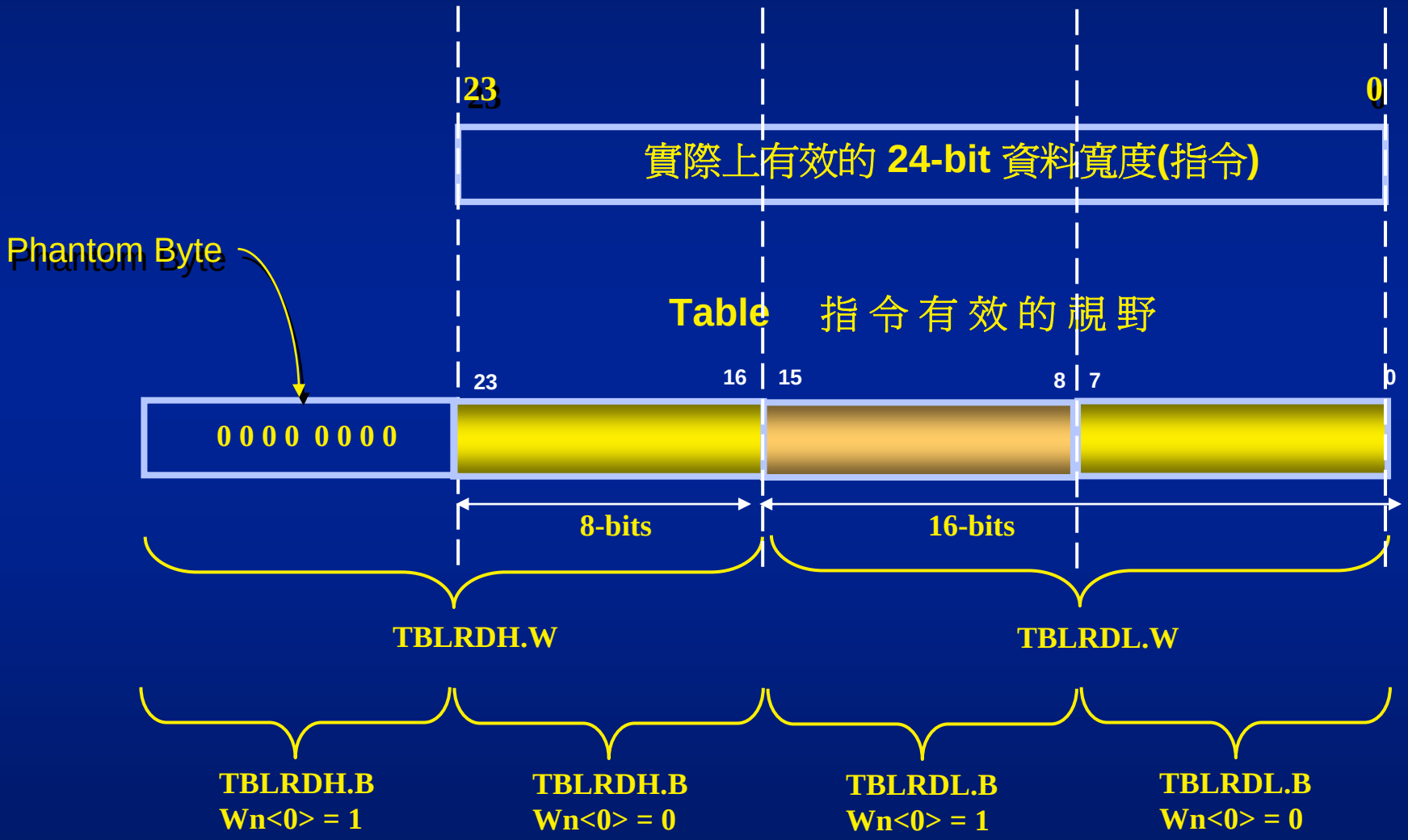
使用 Table 讀寫指令 (一)

- 24-bit 有效位址 (EA) 的組成
  - 有效位址是由 “TBLPAG 暫存器” + “W 暫存器” 所組成的一有效位址
  - Table Page 的大小爲 64K bytes 或 32K words
  - Configuration 的存取需將 TBLPAG<7> 位元設定爲 1
    - 存取 Config. 的設定只能使用這種方式

使用 Table 讀寫指令 (二)

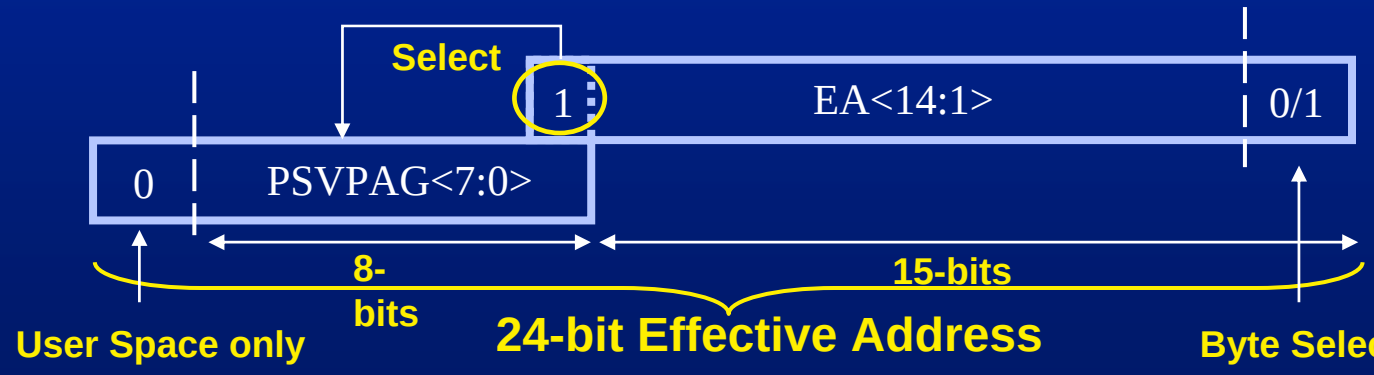
- Table 讀寫指令有四個 :
  - TBLRDL : Table Read Low
  - TBLWTL : Table Write Low
  - TBLRDH : Table Read High
  - TBLWTH : Table Write High
- TBLRDH 及 TBLWTH 所存取的最高 Byte 是不存在的 (總共只有 24-bit 的有效資料)
  - 讀取這個區域的資料 (Phantom byte) 其值爲 0x00

利用 Table 指令讀取程式記憶空間的資料

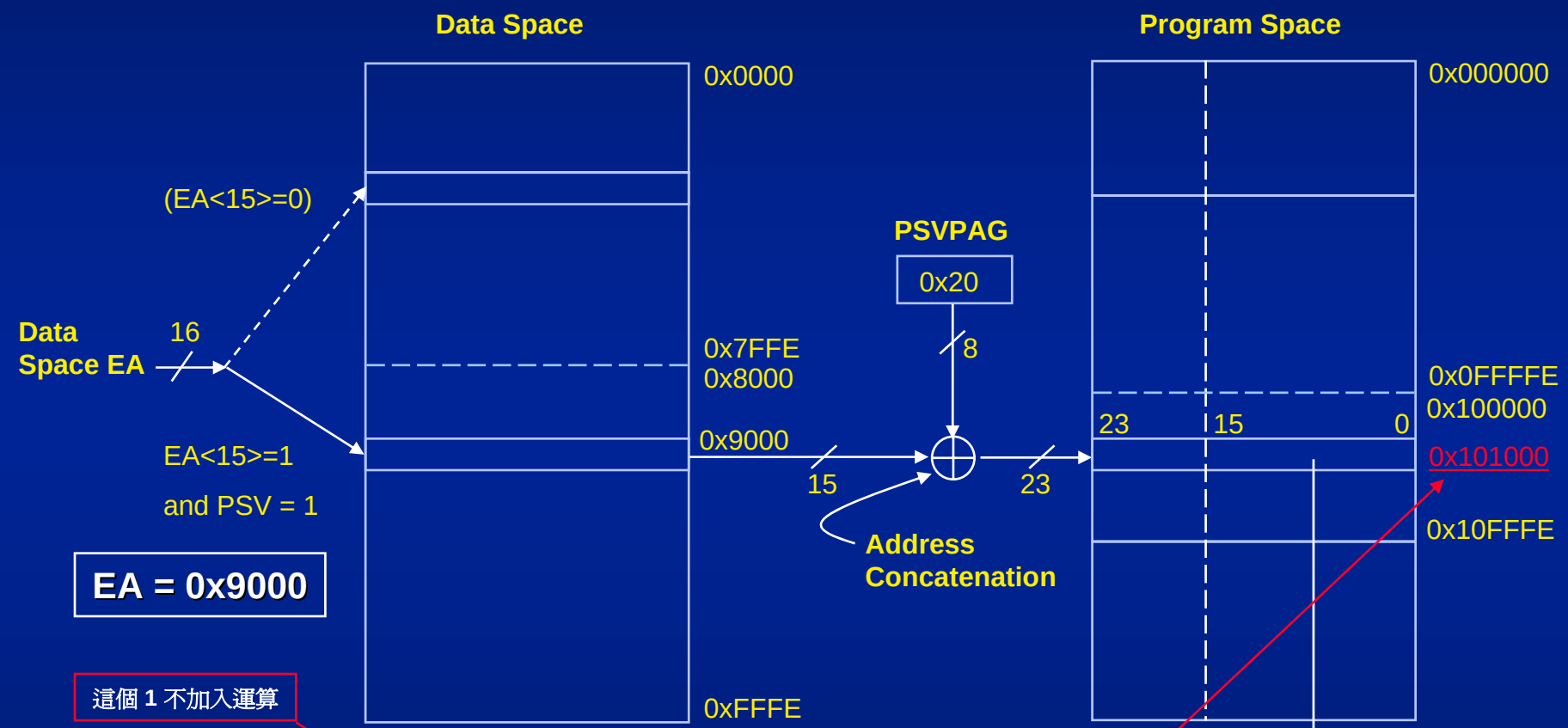


利用 PSV 映對到 RAM 的方式讀取程式記憶體資料

- 每一個程式記憶區塊是以 16 K Word 的空間映對到32K Byte 的 RAM
  - 使用此功能需將 PSV bit (CORCON<2>) 設為 1 ，且 EA<15> =1 之後，PSV 功能將被打開
  - PSVPAG (Program Space Visibility Page register) 暫存器是設定最高 Byte 的位址 (共8個位元；b23:b16)
    - EA<14:1> 是由指定的 [Wn] 所提供



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

利用 PSV 方式讀取程式記憶體資料

- 在這 24-bits 裡，只有較低的 16-bits 可以映對到 RAM 較高位址的 32KB 空間
  - 較高的 8 bits 應該填入無效的指令或 NOP

使用 PSV 的益處

- PSV 讓大型的查表資料更加簡單、有效率
- PSV 提供一般型態的 data/program 溝通管道
  - (Von Neumann - like)
- 範例: PSV 使用在 FIR 濾波器
  - One MAC per filter tap, executed within a REPEAT loop
  - PSV allows filter coefficients to be stored in PS saving valuable SRAM
  - Minimal performance impact. 1 cycle per REPEAT iteration plus:
    - 1 extra cycle at REPEAT entry due to PSV data pipe fill
    - 1 extra cycle at REPEAT exit due to PSV data pipe flush

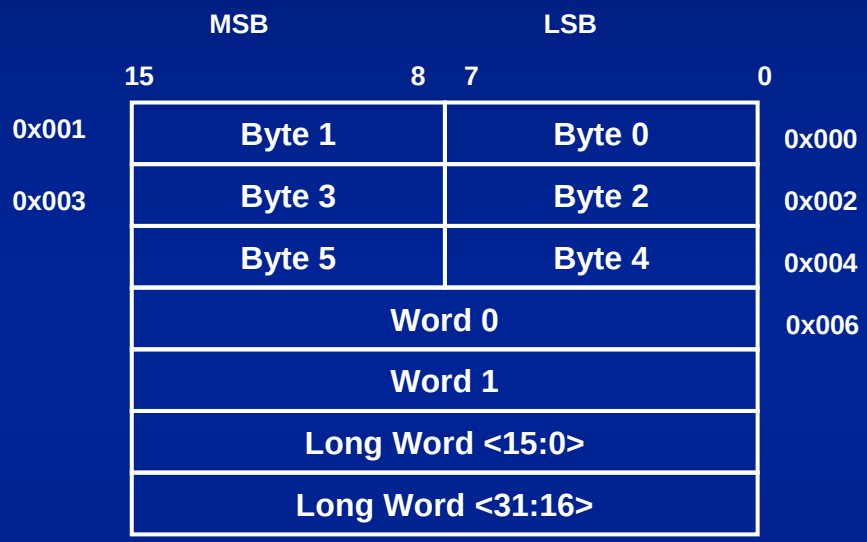
資料記憶

Data Memory

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)

Little-Eendian 儲存格式



資料在 **RAM** 的存放方式

* 所有對 **word** 的存取，必須是一個偶數位址 **(EA<0>=0)**

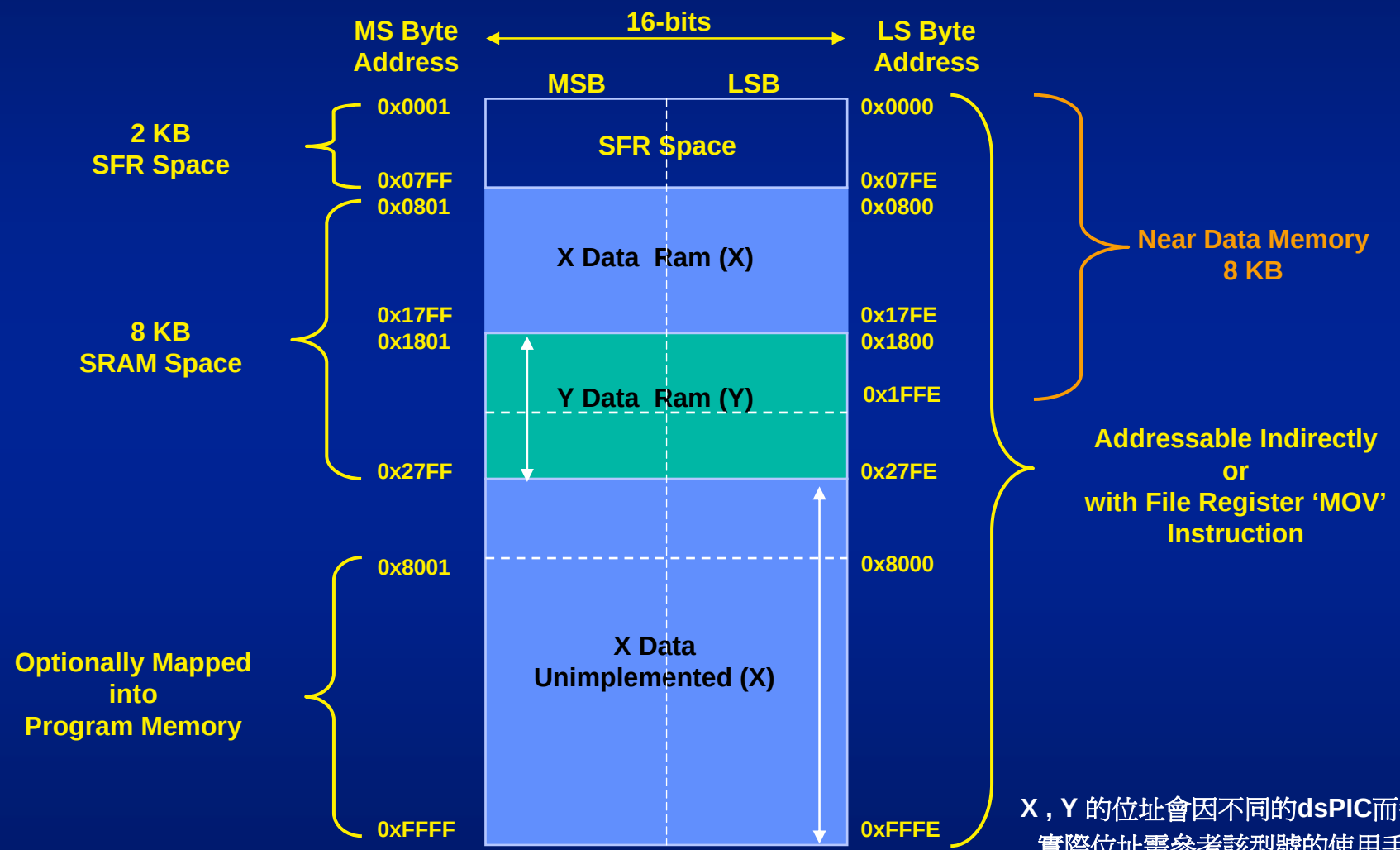
Data Memory 架構 (二)

- 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 ….

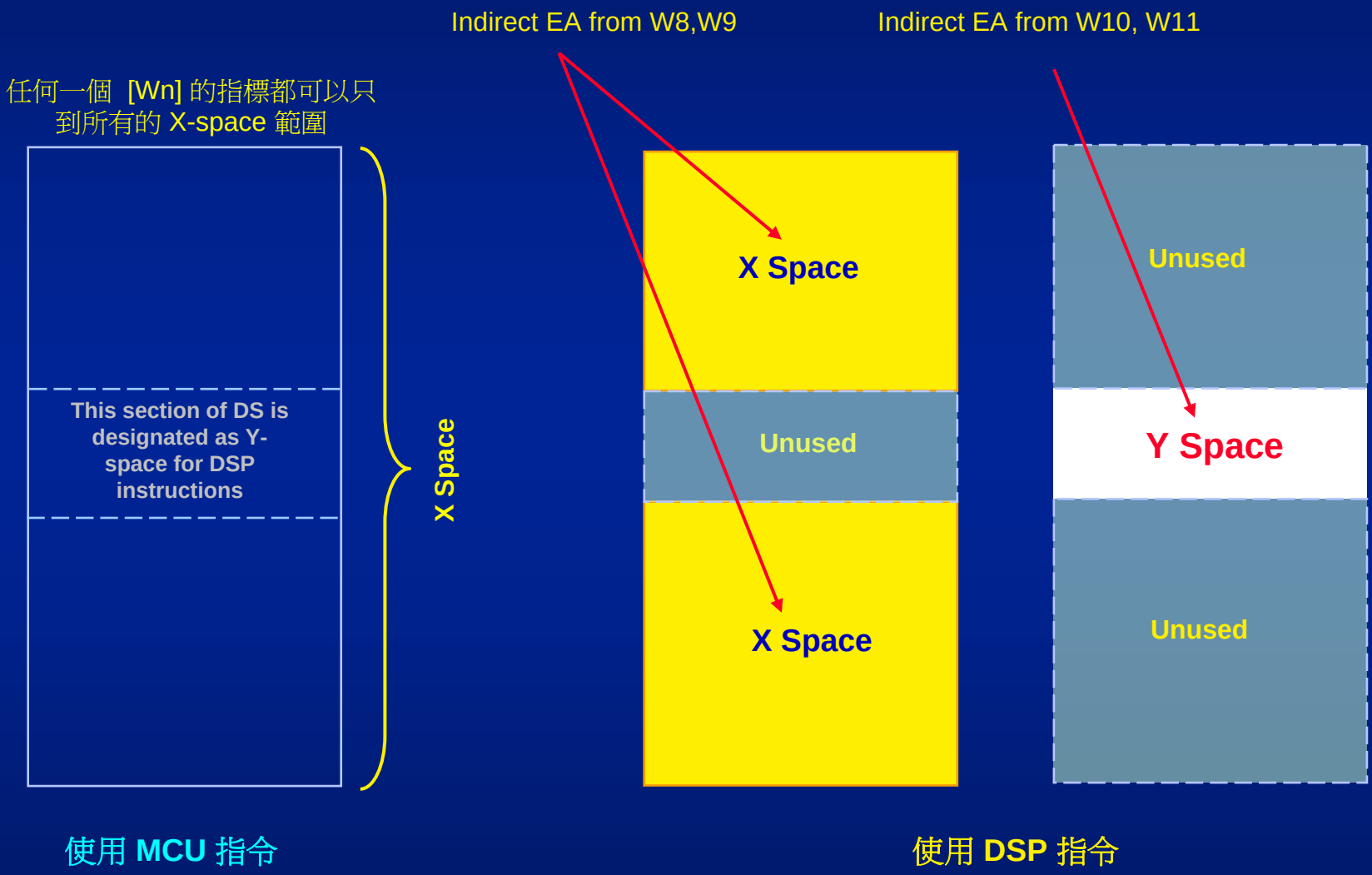
  - 兩個分別獨立空間 (X & Y) 給使用 DSP 的 MAC 指令
    - 允許在同一週期裡同時對 X , Y 兩個資料區同時存取，以提高效率

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

X , Y 位址產生單元
AGUs

- 兩個位址產生單元 X AGU 和 Y AGU
- X， Y AGUs 都可以支援 MODULO 定址模式
- 只有 X AGU 可以支援 Bit-Reversed 定址模式
- X AGU 所指到的資料區可以被讀寫
- Y AGU 所指到的資料區只能讀取不支援寫入

Data Memory 架構 (四)

- RAM 區的前面 8 KB 稱之爲 “Near” RAM
  - 位址從 0x0000 - 0x1FFF
  - 可適用於直接定址模式的指令
    - 13-bit 獨立的位址直接填入到指令裡，所使用的使暫存器直接定址模式
  - 包含特殊功能暫存器 (SFR) ，位址從 0x0000 - 0x07FE
  - 要使用直接定址存取 64KB 的範圍，唯有使用 Wn 暫存器方式存取

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)

dsPIC 架構
與
運算核心

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

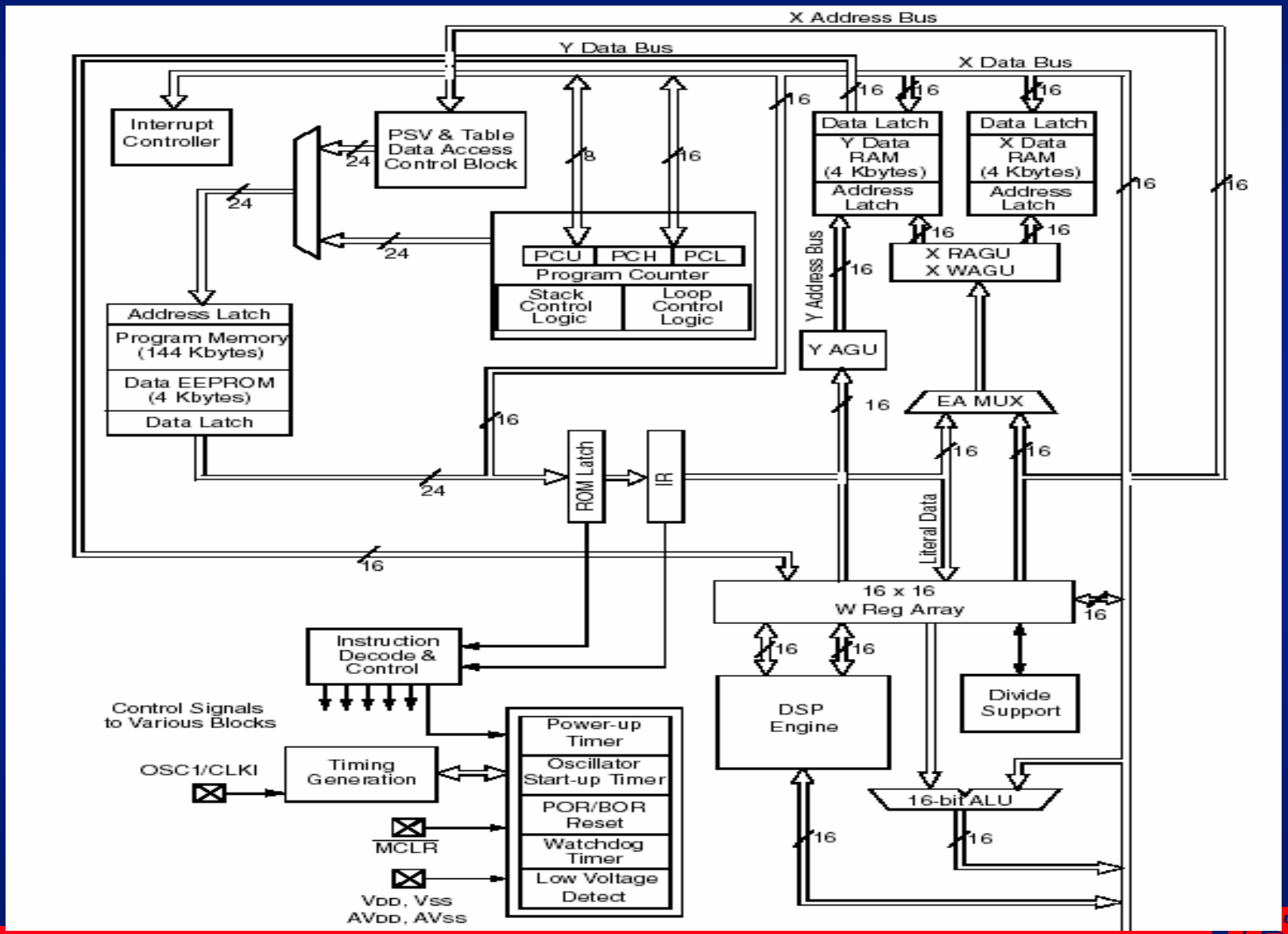
* 這些單元與 MCU 共用

dsPIC30F CPU 基本組成(二)

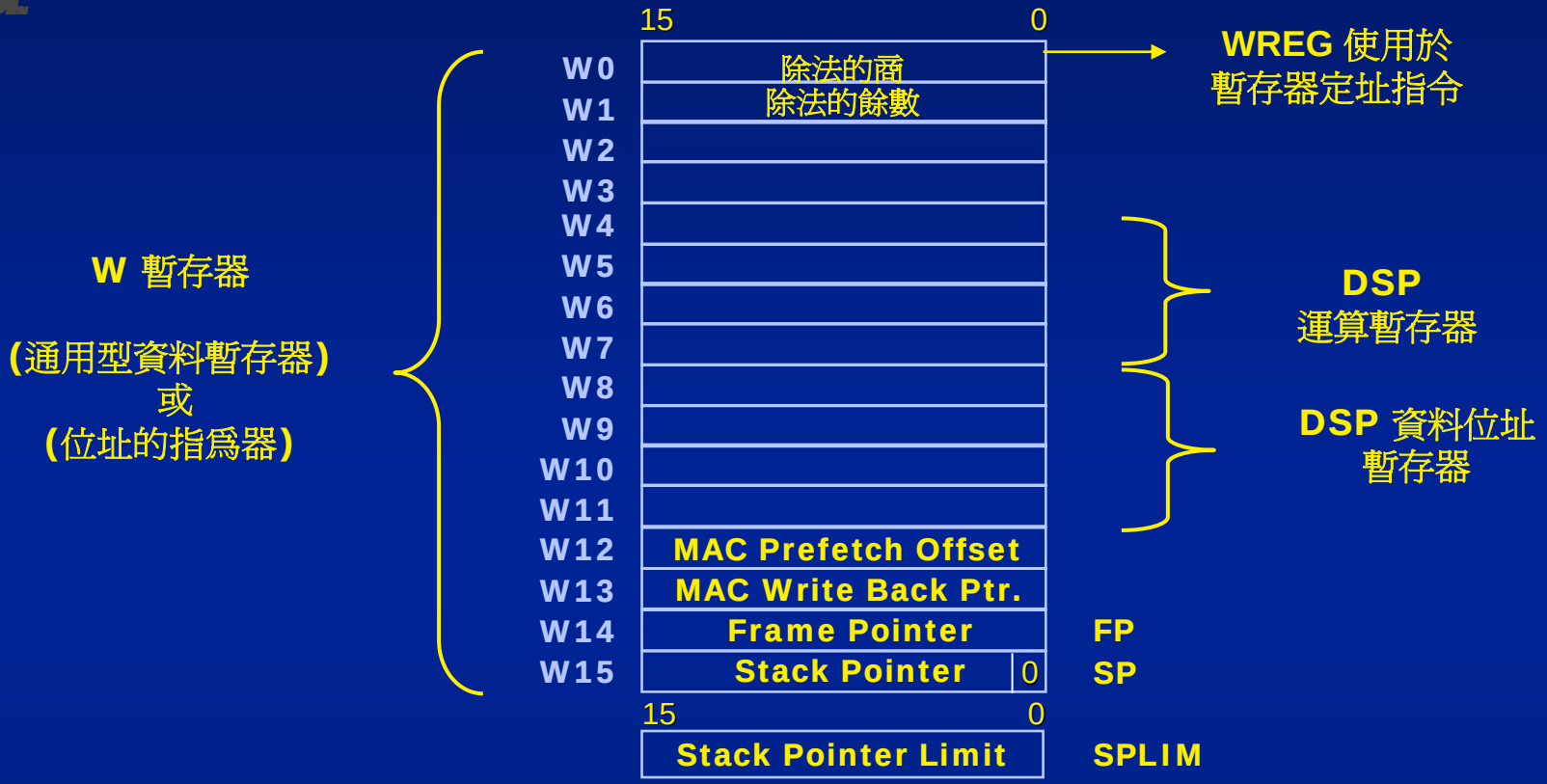
- 構成 MCU 的主要單元
  - 16-bit ALU
  - 除法運算邏輯
  - W 暫存器陣列
  - 位址產生器

- 硬體迴圈控制以支援 DO 及 REPEAT 迴圈指令

dsPIC CPU 方塊圖



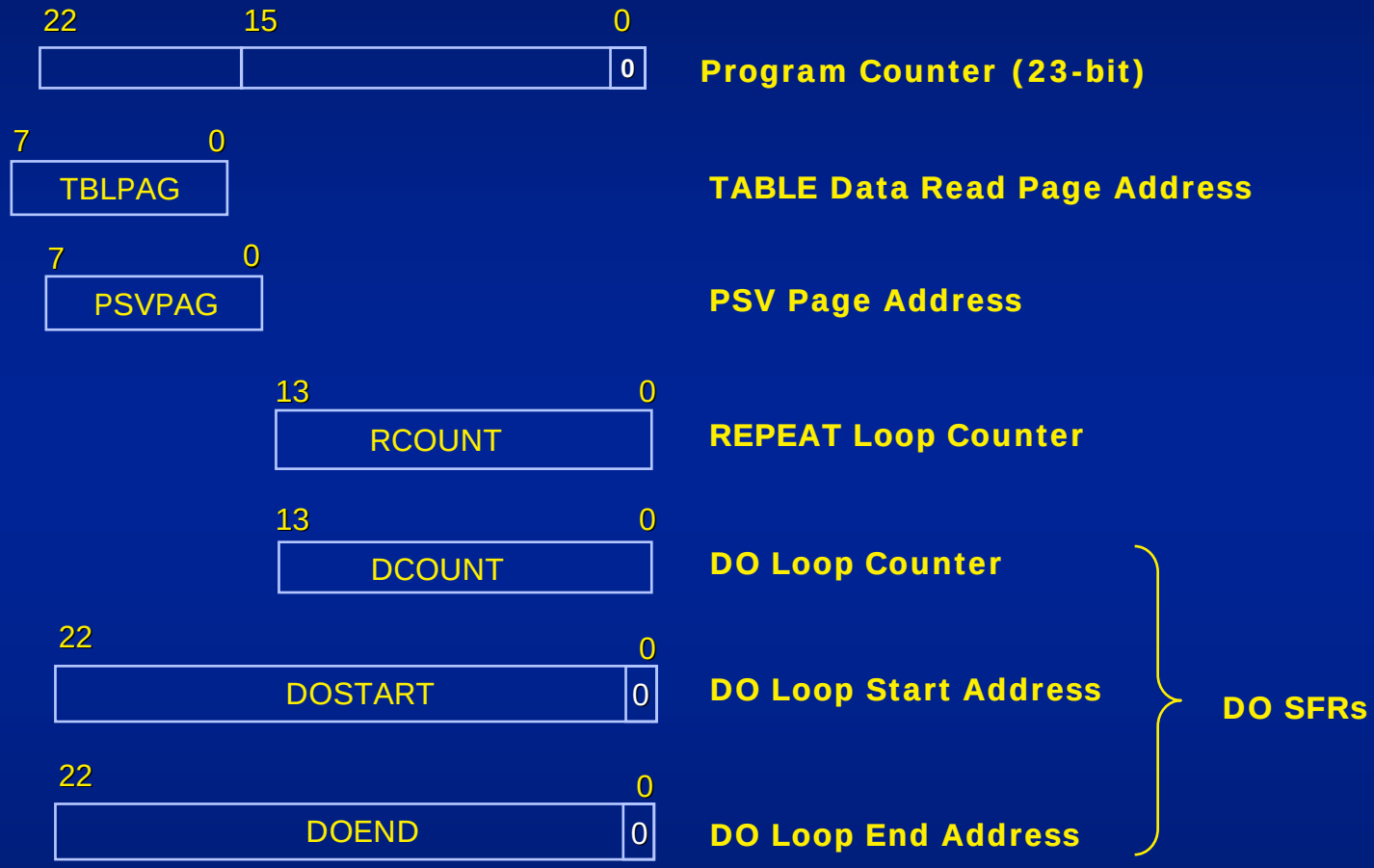
記憶體模型 (一)


15
0
W0
W1
W2
W3
W4
W5
W6
W7
W8
W9
W10
W11
W12
W13
W14
W15
除法的商
除法的餘數
MAC Prefetch Offset
MAC Write Back Ptr.
Frame Pointer
Stack Pointer
0
FP
SP
15
0
Stack Pointer Limit
SPLIM
WREG 使用於
暫存器定址指令
W 暫存器
(通用型資料暫存器)
或
(位址的指爲器)
DSP
運算暫存器
DSP 資料位址
暫存器
DSP 累積器
(40-bit)
ACCA
ACCB
ACCAU
ACCAH
ACCAL
ACCBU
ACCBH
ACCBL
39
32 31
16 15
0
SRH
SRL
OA
OB
SA
SB
OAB
SAB
DA
DC
IPL2
IPL1
IPL0
RA
N
OV
SZ
C
狀態暫存器
DSP Status
MCU Status

狀態暫存器

- OA , OB , OAB
  - Accumulator Overflow Status bit (overflowed into <32:39>)
- SA , SB , SAB
  - Accumulator Saturation STICKY Status (bit 31 or 39 overflow)
- DA
  - Do Loop Active bit (Read Only)
- DC
  - MCU ALU Half Carry/Borrow bit
- RA
  - REPEAT Loop Active bit (Read Only)
- N , OV , Z , C

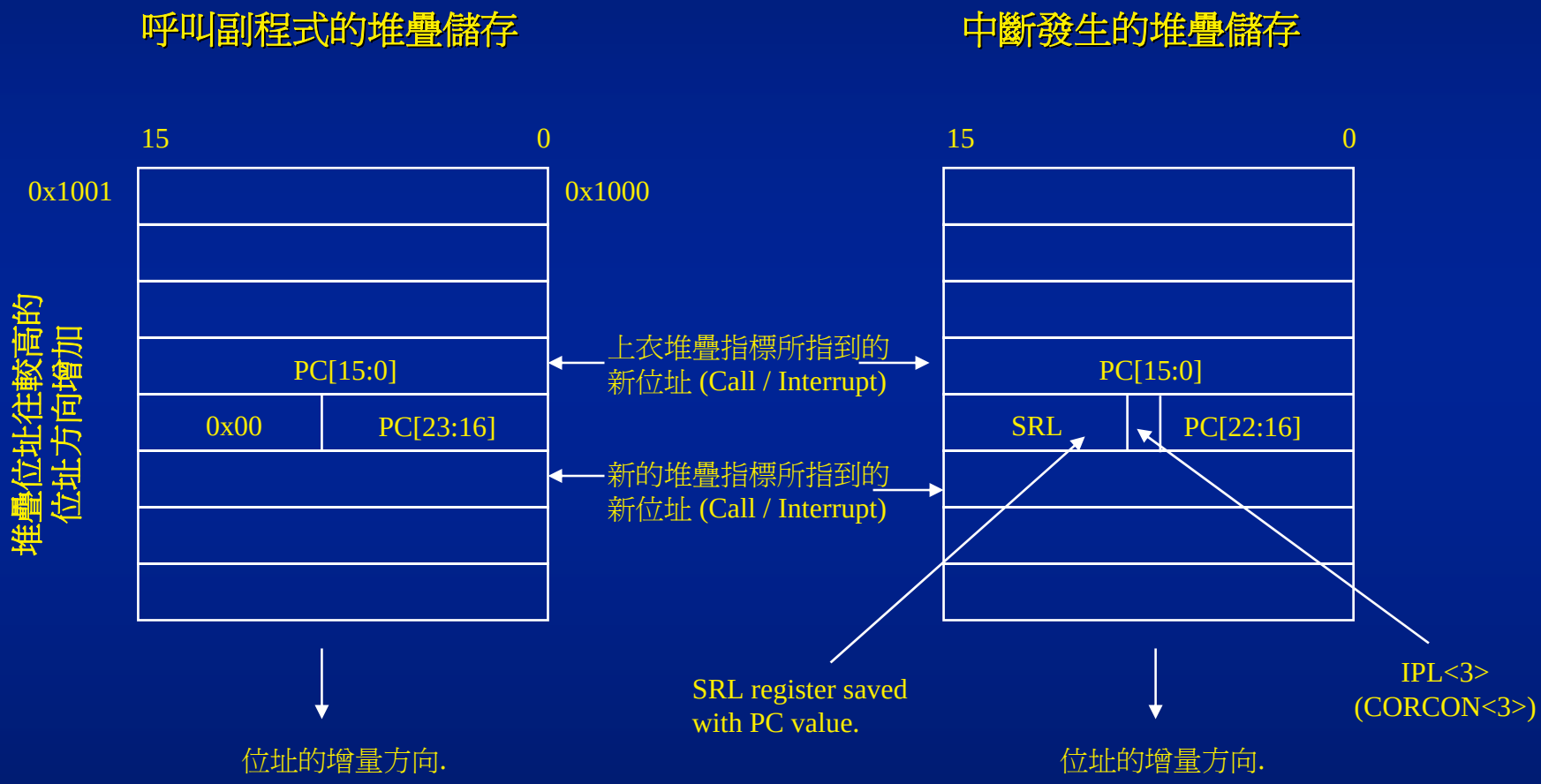
記憶體模型 (二)


22
15
0
0
Program Counter (23-bit)
7
0
TBLPAG
TABLE Data Read Page Address
7
0
PSVPAG
PSV Page Address
13
0
RCOUNT
REPEAT Loop Counter
13
0
DCOUNT
DO Loop Counter
22
0
DOSTART
0
DO Loop Start Address
22
0
DOEND
0
DO Loop End Address
DO SFRs
15
0
US
EDT
DL2
DL1
DL0
SATA
SATB
SATDW
ACCSAT
IPL3
PSV
RND
IF
Core Control Register (CORCON)

軟體堆疊 (一)

- W15 是軟體的堆疊指標
  - PUSH 指令
    - 將資料推入目前堆疊指標(W15)所指的RAM位址
    - W15 的指標加一
  - POP 指令
    - W15的堆疊指標先減一
    - 自堆疊中取回資料
  - 執行 CALL 指令時 PC 的值會自動存入堆疊裡
  - 中斷發生時，PC 及 SRL (MCU 狀態旗號) 也會自動存入堆疊裡

軟體堆疊 (二)

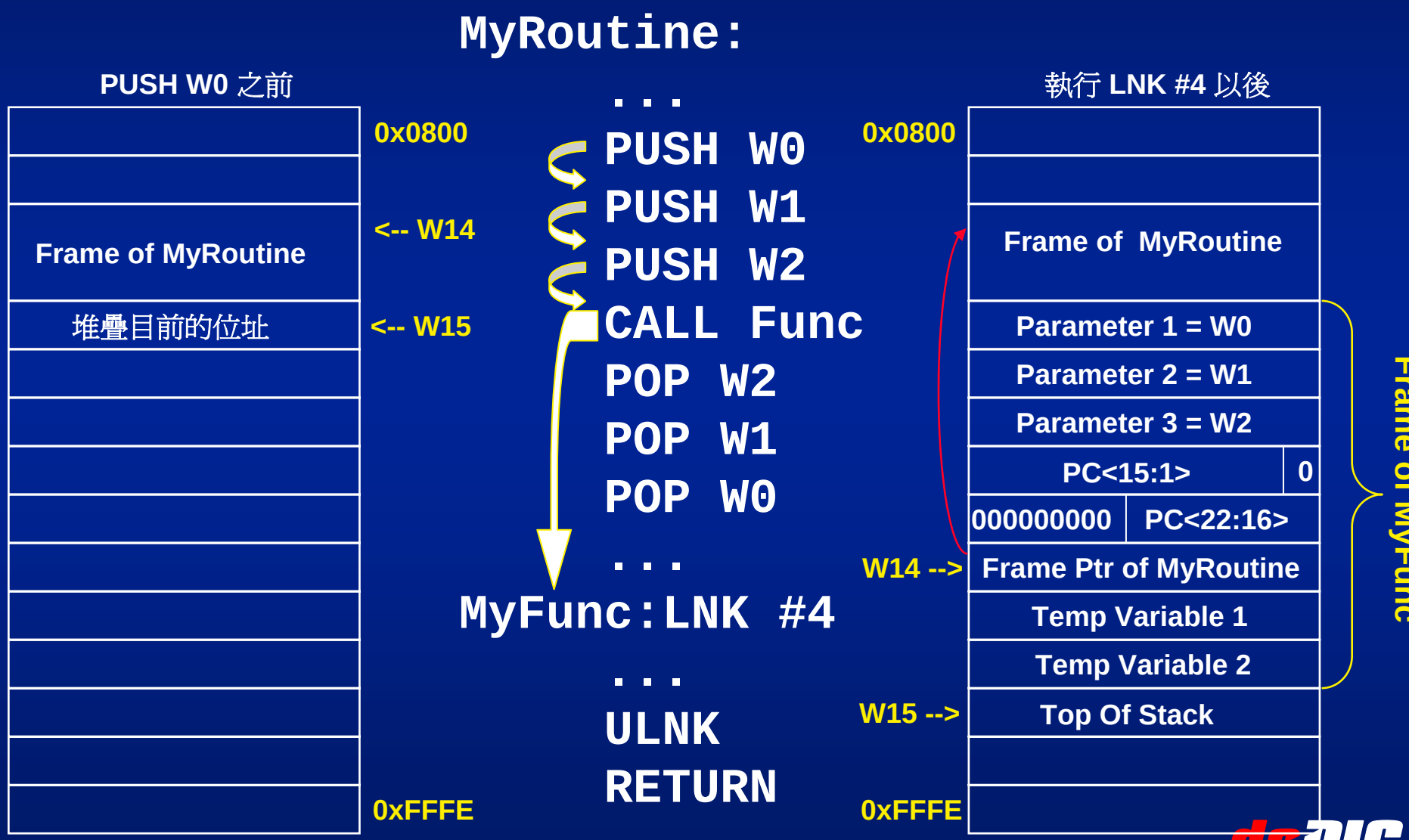


Note: 堆疊指標 **(W15)** 永遠指向下一個可用的堆疊位址

軟體堆疊：框架指標
Frame Pointer

- W14 可以被使用在堆疊的框架指標 (FP)
- LNK #N；分配 (N / 2) 個 word 給區域變數使用
  - W14 指向上一個函數的最後堆疊位址；W15 指向新的堆疊位址
  - 呼叫新的函數/副程式時，W14 的位址會被先推入現今的堆疊，W15 加一
  - W15 的堆疊位址再被存到 W14 (新的框架指標就指向新的區域變數的堆疊位址)
  - 區域變數的儲存單位是以 16-bit (word) 爲單位
  - 利用 [W14 + n] 就可以存取函數/副程式內的區域變數
  - W14 可以存取函數所傳遞的參數
- ULNK；還原暫存區域
  - 先將 W14 (區域變數的起始位址-1) 的指標位址存到 W15
  - 將 W15 目前所指到位址的內容(上一層的框架指標位址) 存回到W14
  - W14 指向上一層的區域變數的起始位址

框架指標的操作


MyRoutine:
...
PUSH W0
PUSH W1
PUSH W2
CALL Func
POP W2
POP W1
POP W0
...
MyFunc:LNK #4
...
ULNK
RETURN
PUSH W0 之前
0x0800
Frame of MyRoutine
<-- W14
堆疊目前的位址
<-- W15
0xFFFE
執行 LNK #4 以後
0x0800
Frame of MyRoutine
Parameter 1 = W0
Parameter 2 = W1
Parameter 3 = W2
PC<15:1>
0
000000000
PC<22:16>
W14 -->
Frame Ptr of MyRoutine
Temp Variable 1
Temp Variable 2
W15 -->
Top Of Stack
0xFFFE
Frame of MyFunc

REPEAT 指令 (一)

- 指令：REPEAT #lit14 或 REPEAT Wn
  - 重複執行下一個指令 “N+1” 次，“N” 可以到 14-bit 的立即常數值 或 使用 W 暫存器做計數
  - 迴圈計數值會存入 RCOUNT 暫存器
  - 硬體會設定狀態旗號 RA bit；如果 RCOUNT 的值 > 1 的話，指令的預提取動作會被禁能
  - 如果 RCOUNT 的值 = 1，硬體會清除狀態旗號 RA bit，指令的預提取動作會被致能

REPEAT 指令 (二)

- REPEAT 指令執行時是可以被中斷的
  - 中斷發生時，RA bit 會被推入堆疊 (SRL)
  - 如要在中斷裡再執行 REPEAT 指令時，需將 RCOUNT 暫存器的值存入堆疊
  - 強制清除 RCOUNT 暫存器可以直接離開迴圈
- REPEAT 迴圈的限制
  - 任何指令都可以緊跟著 REPEAT 指令，除了：
    - 改變程式路徑的指令， DO 或 REPEAT 指令
    - DISI, LNK, PWRSAV, ULNK, RESET 指令

DO 指令 (一)

- DO 指令會重複指定的指令群 “N+1” 次
  - “N” 可以爲 14-bit 立即常數值或 W 暫存器
  - Loop 迴圈的起始位址存入 DOSTART 暫存器
  - Loop 迴圈的結束位址存入 DOEND 暫存器
  - Loop 迴圈的計數值存入 DCOUNT 暫存器
- Loop 迴圈內的指令最多可達 32K 個指令
  - 指令的執行不須連續，可以改變路徑

DO 指令 (二)

- 在程式裡，DO 指令可有 7 層的深度
  - 可以允許存在 6 層的巢狀式 DO 迴圈
  - DO 迴圈所使用到的暫存器在使用第二層DO 迴圈時，會自動存入 shadower (只有一層的存取)
  - 第二層以上的 DO 迴圈需自行使用軟體堆疊儲存
  - DO 迴圈層次指示位元 DL<2:0>， 增減由硬體自動處理
  - 當 DL<2:0>≠‘0’ 及 DCOUNT > 1，DA 位元會設為 1

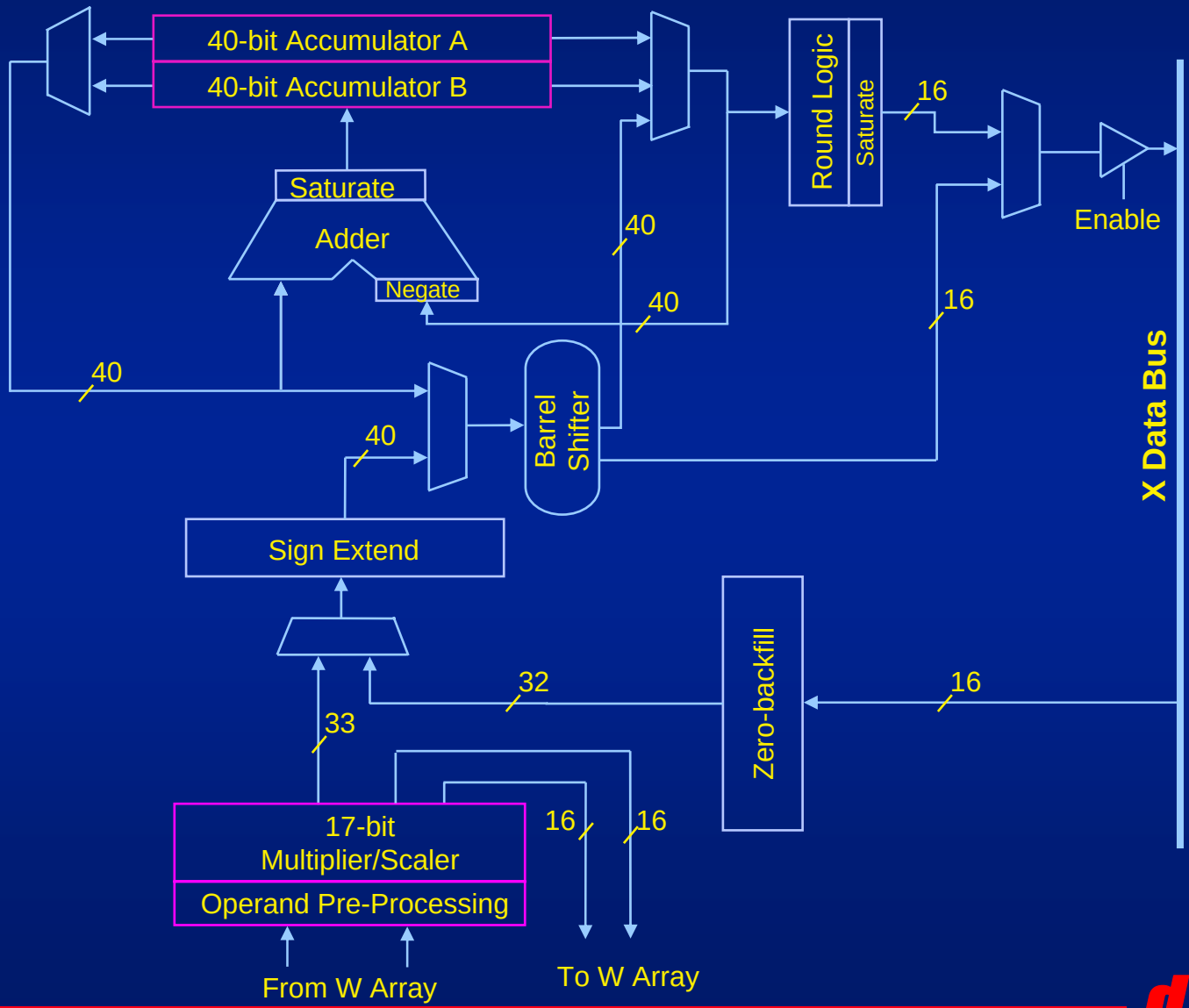
DO 指令 (三)

- Setting the EDT bit terminates DO loop execution (drops the DO level by one)
  - DL bits modified by HW to reflect new DO level
- DO loop is interruptible
  - ISRs can contain DO loops
    - On entry into ISR, the DA or DL<2:0> bits should be checked to determine if the DO SFRs need to be stacked prior to executing any DO loops within the ISR

DO 指令 (四)

- The following instructions cannot be used as one of the last two instructions of a DO loop:
  - Flow Control (i.e., GOTO, CALL, BRA)
  - RETFIE, RETURN or RETLW instructions
  - DO or REPEAT instructions
  - “Compare-and-Skip” type instructions (i.e., CPSGT, CPSEQ, CPSLT)

DSP 運算核新方塊圖


40-bit Accumulator A
40-bit Accumulator B
Saturate
Adder
Negate
Round Logic
Saturate
16
Enable
40
40
40
16
40
Barrel Shifter
X Data Bus
Sign Extend
32
16
33
Zero-backfill
17-bit Multiplier/Scaler
Operand Pre-Processing
16
16
From W Array
To W Array

17x17 硬體乘法器

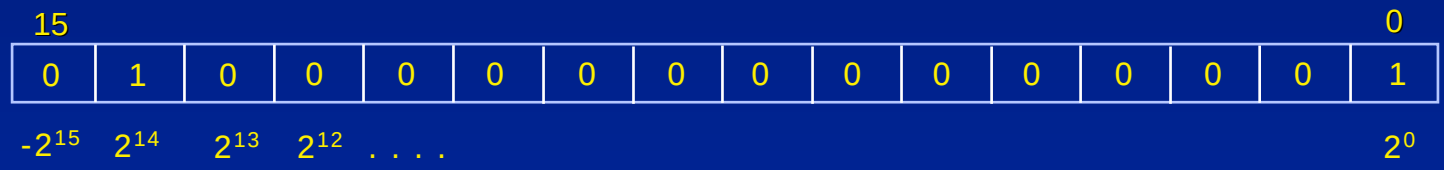
- 所有的乘法運算都只要一個指令週期
  - 有些乘法運算可以使用於 MCU 和 DSP 的指令
  - 乘積 32-bit 的結果有兩種數值格式
    - 1.31 小數格式 (fractional)
    - 32-bit 整數格式 (integer)
- 17-bit x 17-bit 有號數乘法器
  - (-1.0) * (-1.0) 運算精度的改良
  - 支援有號數、無號數的乘法

數值的格式: 整數與小數

- 整數格式
  - 以有號數的 2's 補數型態表示
  - 範圍 [-32768 ~ 32767] 或 [0x8000 ~ 0x7FFF]
- 小數格式 (Q15)
  - 以有號數的 2's 補數型態，在正負號後面緊接著的是小數點
    - 1 個正負號位元 + 15 個小數位元
    - 1.15 格式 : S.FFFFFFFFFFFFFFF
  - 最大數值範圍 [-1.0 ~ 0.9999]

數值 0x4001 不同的表示方式

整數 (Integer) :



$0x4001 = 2^{14} + 2^{0} = 16384 + 1 = \mathbf{16385}$

1.15 小數格式 (Fractional) :

$0x4001 = 2^{-1} + 2^{-15} = 0.5 + 0.0000305175 = \mathbf{0.5000305175}$

整數格式 / 小數格式的關係

- 小數的值 = 整數值除以 32768
- 整數值 = 32768 乘以小數值
- 轉換的例子…

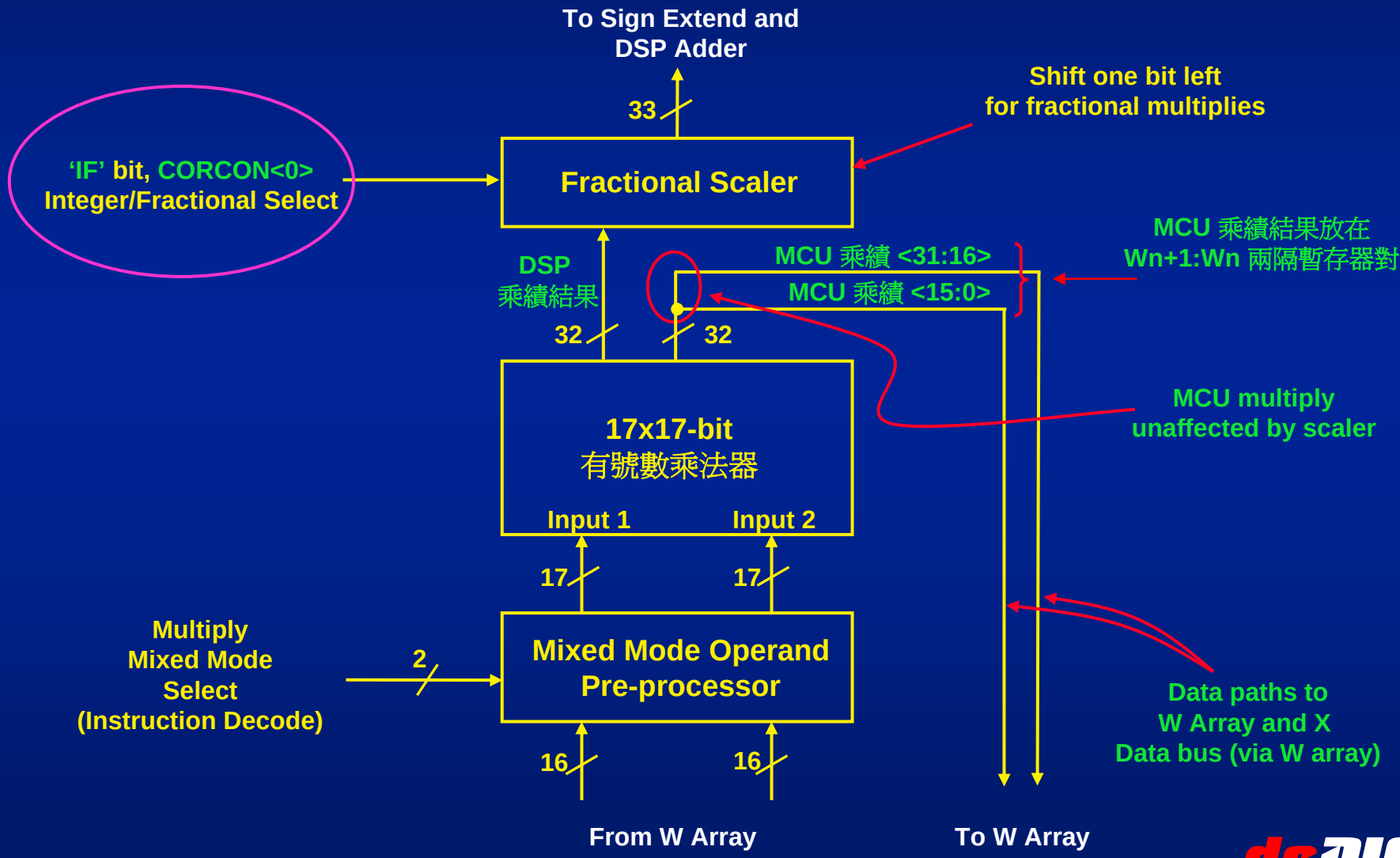
16 位元值	*整數值*	*分數值*
0x8000	-32768	-1.0
0xA000	-24576	-0.75
0xC000	-16384	-0.5
0xE000	-8192	-0.25
0x0000	0	0.0
0x2000	8192	+0.25
0x4000	16384	+0.5
0x6000	24576	+0.75
0x7FFF	32767	+0.999969

整數與小數的轉換：**16385 / 32768 = 0.5000305175 78125**

dsPIC30F 數值表示的範圍

位元數	整數範圍	小數範圍	小數的最小值
16-bit	-32768 32767	-1.0 to (1.0 - 2^{-15}) (Q15 Format)	3.052×10^{-5}
32-bit	-2,147,483,648 2,147,483,647	-1.0 to (1.0 - 2^{-31}) (Q31 Format)	4.675×10^{-10}
40-bit	-549,755,813,888 549,755,813,887	-256.0 to (256.0 - 2^{-31}) (Q31 Format with 8 Guard bits)	4.675×10^{-10}

乘法器方塊圖


To Sign Extend and
DSP Adder
33
Shift one bit left
for fractional multiplies
'IF' bit, CORCON<0>
Integer/Fractional Select
Fractional Scaler
MCU 乘積結果放在
Wn+1:Wn 兩隔暫存器對
MCU 乘積 <31:16>
MCU 乘積 <15:0>
DSP
乘積結果
32
32
17x17-bit
有號數乘法器
MCU multiply
unaffected by scaler
Input 1
Input 2
17
17
Multiply
Mixed Mode
Select
(Instruction Decode)
2
Mixed Mode Operand
Pre-processor
Data paths to
W Array and X
Data bus (via W array)
16
16
From W Array
To W Array

MCU 等級的乘法指令

(MUL.UU, MUL.SS, MUL.US, MUL.SU, MUL.B , MUL.W)

- 16-bit x 16-bit 整數乘法 (使用Wn方式)
  - 無號數，有號數 及 混和模式
  - 使用 W 暫存器 (直接 / 間接定址) 爲運算元
  - 乘積結果爲 32-bit 會被存入到相鄰的 W 暫存器對
    - W1:W0， W3:W2， ... ， W13:W12
  - 例 : MUL W4, W0, W2 → W4 * W0 存入 W3:W2
- 暫存器乘法 (使用 **MUL.B 或 MUL.W 指令**)
  - 使用 WREG 和任一個暫存器當運算元
  - 32-bit 的乘積是放在 W3:W2 暫存器對裡
  - 支援 8 x 8 乘法器，乘積結果至於 W2

DSP 等級的乘法指令

(ED, EDAC, MAC, MPY, MPY.N, MSC)

- 其結果是存入 ACCA 或 ACCB
- 支援有號數 / 無號數，小數 / 整數格式
  - IF 位元 **CORCON<0>**，選擇整數或小數型態運算
  - US 位元 **CORCON<12>**，選擇有號數或無號數型態運算
- Fractional: scaler automatically shifts the multiplier result one bit to the left
  - 小數相乘的積，最後一位(LSB)永遠爲零
  - This maintains proper alignment of the radix point (product has a 1.31 numerical format)

小數乘法範例 (IF位元，CORCON<0> = 1)

- **MPY W4*W5, A**
  - If W4 = 0xFFF7 (-0.000274658)
  - If W5 = 0x7FF7 (0.999725341)
  - 整數的乘積爲 0xFF FFFB 8051 (40-bits)
  - 小數的乘積爲 0xFF FFF7 00A2 (40-bits)

```
  - 0.000274658
x 0.999725341
-----------------
 -.000274582
```

```
      1.111 1111 1111 0111
x     0.111 1111 1111 0111
-----------------------------------
11.11 1111 1111 1011 1000 0000 0101 0001
```

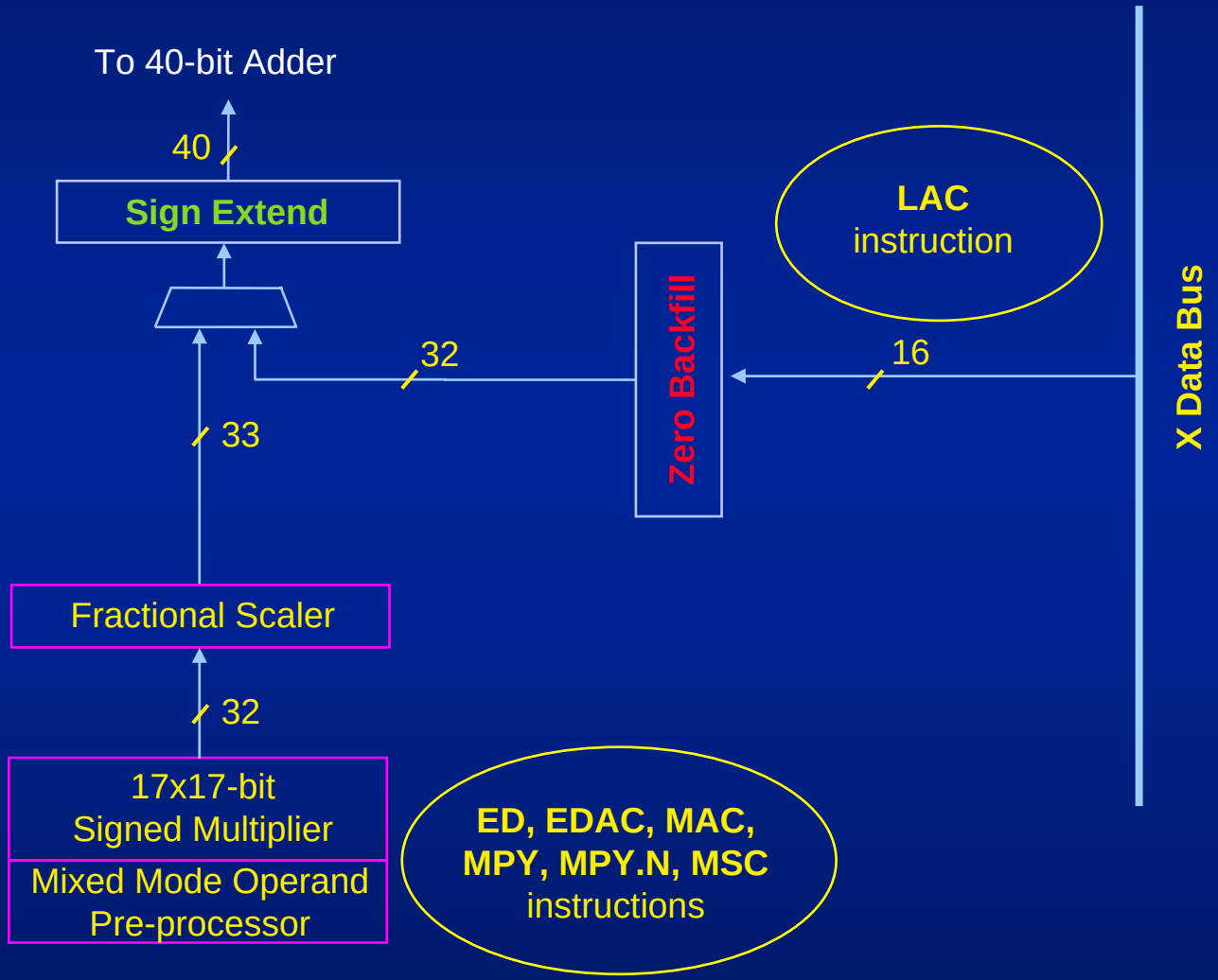
1.30 results

```
11.111 1111 1111 0111 0000 0000 1010 0010
```

1.31 results after scaler shift

Extra sign-extension bit (required for 0x8000*0x8000)

自動填零與有號數的擴展控制邏輯



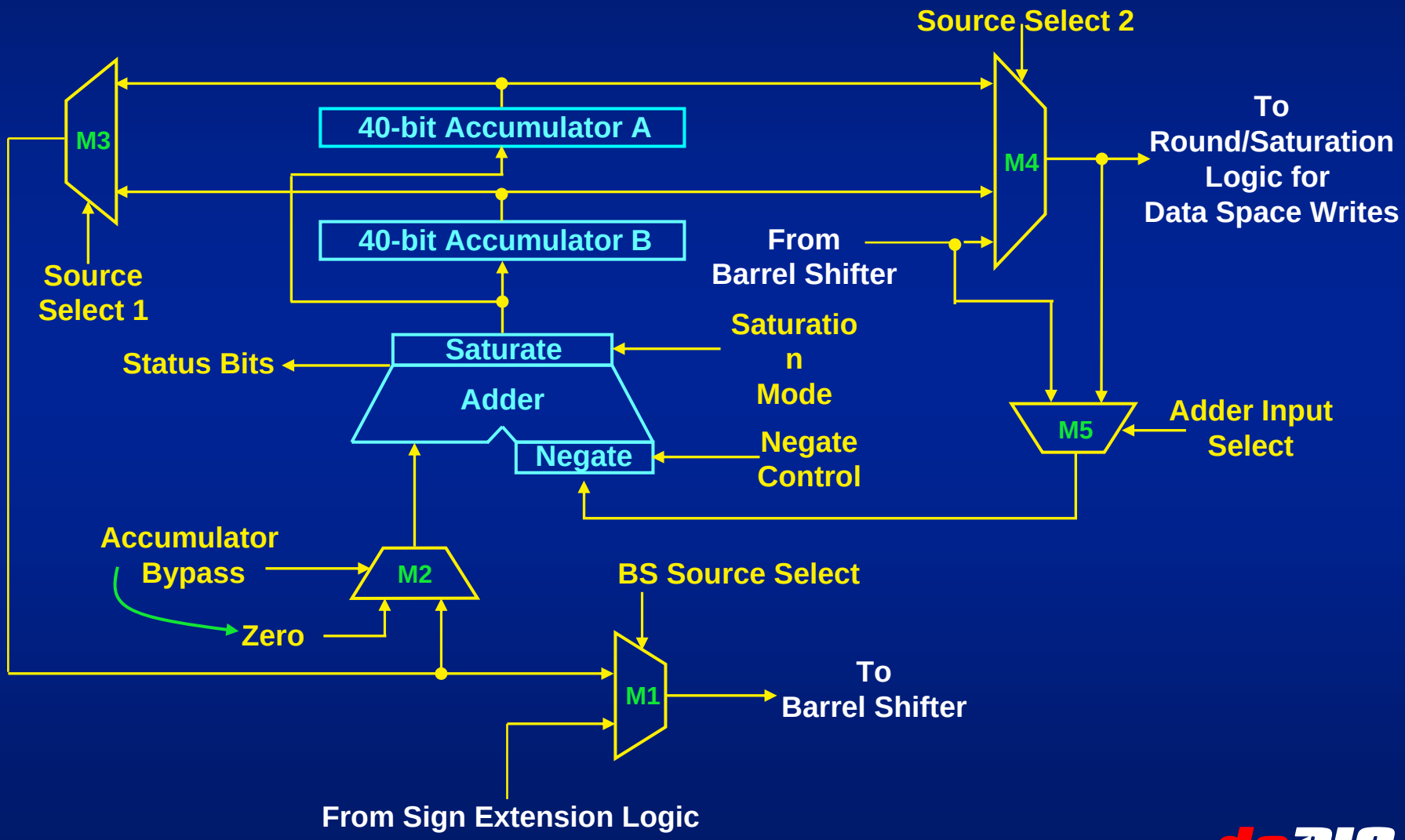
Zero Backfill & Sign Extend Control Block

- Control logic used to interface 16-bit and 32-bit data with the 40-bit accumulator
  - Think of it as a "data massager" which allows data to be properly placed in the 40-bit accumulator
  - Data loads are placed into (ACCxH:ACCxL)
  - DSP products are placed into (ACCxH:ACCxL)
- Sign extend logic sign extends from bit 31 through all of ACCxU
  - ACCxU = 0xFF for negative values
  - ACCxU = 0x00 for positive values

LAC [W4++], #-3, A

	Before	After	
W4	2000	2002	
ACCA	00 5125 ABCD	FF 9108 0000	
0x2000	1221	1221	
SR	0000	4800	(OA,OAB=1)

40-bit DSP Adder and Accumulators



40-bit Accumulators

- Two Independent 40-bit Accumulators
  - 32 result bits + 8 guard bits (for large dynamic range)
  - Accumulators are memory-mapped
  - Overflow detection, flags and associated branch instructions for each Accumulator
  - Two optional saturation modes
    - 31-Bit Saturation - (Normal Saturation)
    - 39-Bit Saturation - (Super Saturation)
  - Accumulator store mechanism is 16-bits wide
    - Optional biased or unbiased rounding
    - Includes independent data write saturation logic

40-bit Accumulator Alignment

Accumulator Alignment and Usage

A. 40-bit Accumulator consists of ACCxU:ACCxH:ACCxL
B. Load and Store Operations
C. Operations in Normal Saturation mode
D. Operations in Super Saturation mode

DSP instructions LAC, SAC and SAC.R are used to load and store the accumulators, since they provide sign-extension, shifting and rounding capabilities.

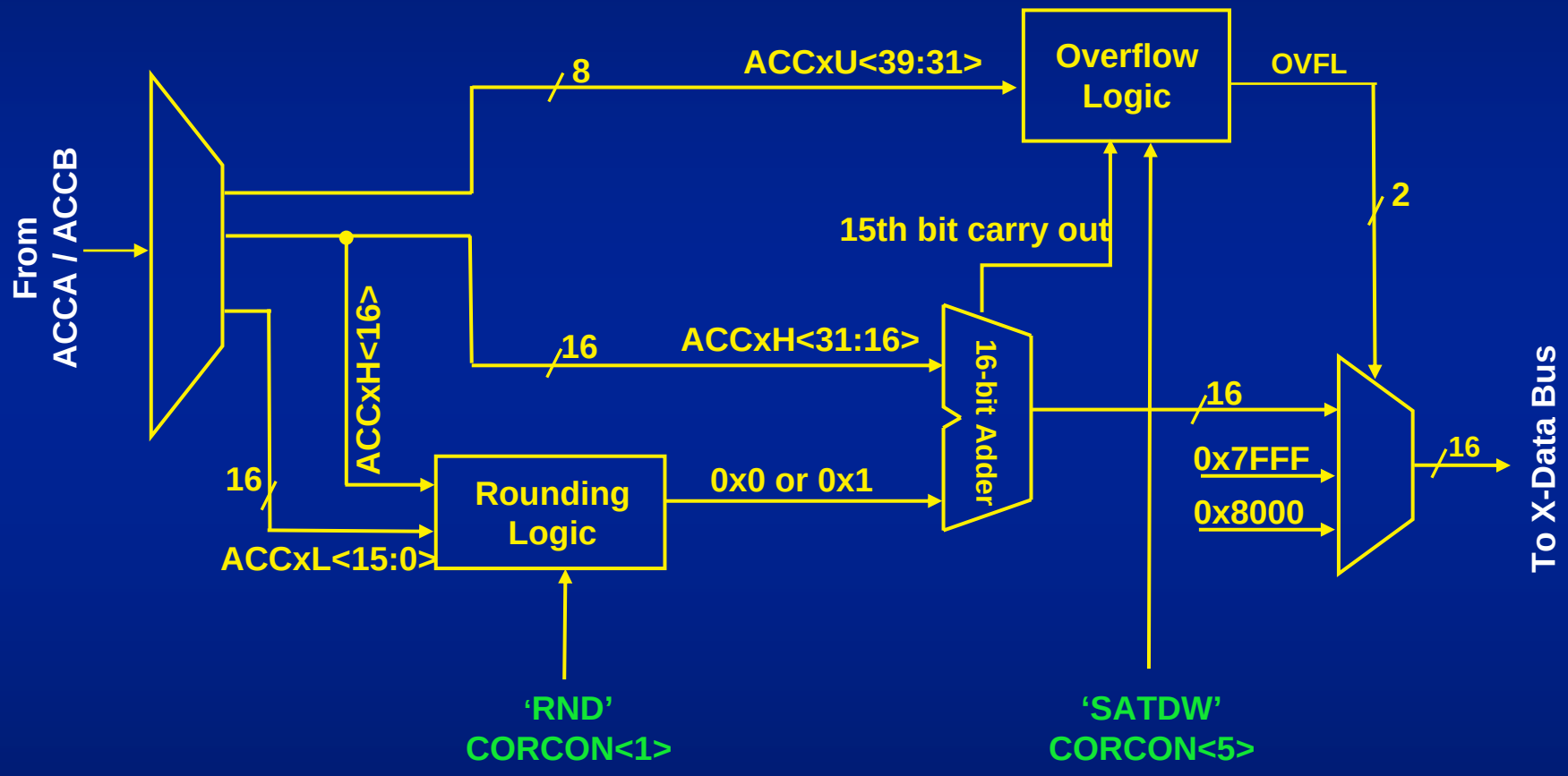
Four Methods of Accumulator Storage

- **SAC -** Stores truncated ACCxH
- **SAC.R -** Stores rounded ACCxH
- Accumulator Write-Back (via **CLR**, **MAC**, **MOVSAC**, **MSC**) - Stores rounded ACCxH
- **MOV** from memory mapped Accumulator register
  - Contents of specified register are stored
  - No rounding or DS Write Saturation performed
  - Not recommended for most applications (for storing integer products, this method is OK)

Data Space Write Saturation

- Provides a 16-bit interface to the Data Bus even when 1.31 Saturation is not used
  - For **SAC**, **SAC.R** and Accumulator Write-Backs, the 1.15 saturated contents of ACCx are stored
  - Contents of Accumulator are not changed
  - Mode is enabled by SATDW (CORCON<1>)
  - Example (SATDW = 1, ACCSAT = 0):
    - **SAC.R A, W0 ; Stores ACCA to W0**
    - If ACCA = 0x00 7FFF FFFF, the value stored to W0 = 0x7FFF

Round and Data Write Saturation



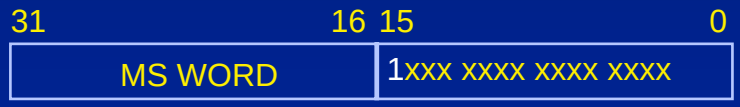
Accumulator Rounding

- Only utilized by **SAC.R** and Acc. Write-Back
- Mode is selected by RND bit (CORCON<1>)
  - Conventional (biased): Rounds up if ...
    - ACCxL<15> = 1
  - Convergent (unbiased/default): Rounds up if..
    - ACCxL is 0x8000 and ACCxL<16> = 1

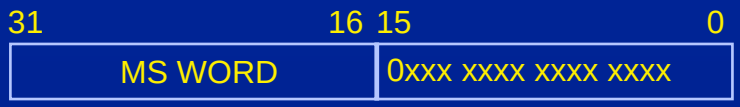
      OR
    - ACCxL is greater than 0x8000
  - Rounding does not affect Accumulator

Accumulator Rounding during Memory Store

CONVENTIONAL ROUNDING
RND = 1

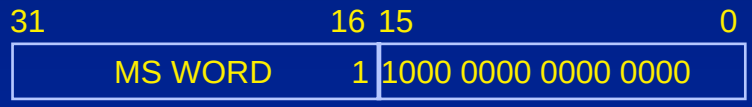


Round Up (add 1 to MS word) when:
- LS word >= 0x8000

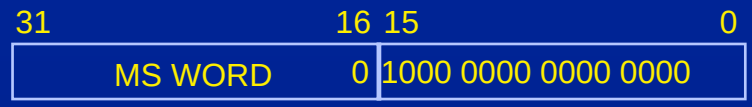


Round Down (add nothing) when:
- LS word < 0x8000

CONVERGENT ROUNDING
RND = 0



Round Up (add 1 to MS word) when:
- LS word = 0x8000 and bit 16 = 1
- LS word > 0x8000



Round Down (add nothing) when:
- LS word = 0x8000 and bit 16 = 0
- LS word < 0x8000

The 'RND' bit, CORCON<1>, selects the Accumulator rounding mode during memory store. [Default is convergent on POR]

Example 1 (Rounding): SAC.R A, Slit4, Wdo (Slit4 = [-8 ... +7])

Example 2 (Truncation): SAC A, Slit4, Wdo (Slit4 = [-8 ... +7])

40-bit Barrel Shifter

- 40-bit wide barrel shifter
  - Used for both MCU and DSP shift operations
  - Shift ACCA, ACCB, W registers or memory
  - Maximum shift range is 16 bits to the left or right - varies by instruction type
  - Overflow recognition and range limit detection
  - Integral part of DSP engine so permits shifting concurrent with other DSP operations

Shift Operation Types

- MCU shifts (arithmetic and logical)
  - Operate on 16-bit Working registers
  - Literal shift range of 15 bits to the left or right
  - Variable shift by any value
- DSP shifts (arithmetic only)
  - Operate on ACCA or ACCB
  - Literal & variable range of 16 bits to the left or right
    - Shifts beyond 16 bits cause an arithmetic error trap
  - If enabled, saturation is applied to the result
    - Saturation correct even for shifts that overflow ACCx

MCU Shifts

- **SL, ASR, LSR** Instructions
  - Literal shift range of 15 bits to the left or right
    - **SL Wb, #lit4, Wnd** (also **ASR**, **LSR**)
  - Variable shift by any value in Wns
    - **SL Wb, Wns, Wnd** (also **ASR**, **LSR**)
    - Wns > 15 will result in:
      - SL, LSR: Wnd = 0x0000
      - ASR when Wb<15>=0 : Wnd = 0x0000
      - ASR when Wb<15>=1 : Wnd = 0xFFFF

DSP Shifts

- **SFTAC** Instruction**:** Literal or variable shift value
  - **SFTAC B, #slit6 or SFTAC A, Ws**
  - Shift range of 16 bits to the left or right
  - If enabled, saturation is applied to the result
  - Saturation correct even for shifts that overflow ACCx
  - Variable shift in excess of maximum range will cause a trap (accumulator is preserved)
- **LAC**, **SAC** and **SAC.R** Instructions
  - Reduced range of up to 8 bits left or 7 bits right
  - Shifting is optional

dsPIC 指令集

指令集概述

- 總共 84 個指令
  - 大部分的指令爲單一 word (24 bit) 寬度，但少數指令爲兩個 word 的寬度
- 大部分的指令爲一個指令週期，除了以下的指令:
  - Program Flow Changes (2 cycles)
  - Branch Instructions (2 or 3 cycles)
  - Table instructions (2 cycles)
  - MOV.D Instructions (2 cycles)
  - DO instruction (2 cycles)
  - Divide Instructions* (18 cycles)

*Note: Division is iterative(used with REPEAT) and interruptible on any cycle

指令的分類

- The dsPIC® 30F ISA features :
  - MOVE instructions
  - MATH instructions
  - LOGIC instructions
  - SHIFT / ROTATE instructions
  - BIT instructions
  - STACK instructions
  - PROGRAM FLOW instructions
  - CONTROL instructions
  - DSP / Accumulator instructions

MCU 類別的指令集

MCU 指令摘要

- 具有三個運算元 (3-operand) 的MCU 指令功能
  - 增強 C 編譯器的效率
  - A = B 與 C 的運算
    - A = W register
    - B = W register or indirect address
    - C = W register or indirect address
    - Indirect address can be pre/post modified

```
ADD     W1, W2, W3          ; W3 = W1 + W2
ADD     W1, [W2], [W3++]    ; *W3++ = W1 + *W2
MUL.UU  W1, [W2++], W4      ; W4 = W1 * [*W2++]
```

DSP 指令摘要

Assembly Syntax	Description
ADD Acc	Add Accumulators
ADD Wso,#Slit4, Acc	[Shift then] Add to Accumulator *
CLR Acc,Wx,Wxd,Wy,Wyd,AWB	Clear Accumulator
ED Wm*Wm,Acc,Wx,Wxd,Wy	Partial Euclidean Distance
EDAC Wm*Wm,Acc,Wx,Wxd,Wy	Euclidean Distance
LAC Wso,#Slit4,Acc	[Shift then] Load Accumulator *
MAC Wm*Wn,Acc, Wx,Wxd,Wy,Wyd,AWB	Multiply and Accumulate
MAC Wm*Wm,Acc,Wx,Wxd,Wy,Wyd	Square and Accumulate
MOVSAC Acc,Wx,Wxd,Wy,Wyd,AWB	Store Accumulator and Pre-fetch
MPY Wm*Wn,Acc,Wxd,Wx,Wyd,Wy	Multiply Wn by Wm to Accumulator
MPY Wm*Wm,Acc,Wx,Wxd,Wy,Wyd	Square to Accumulator
MPYN Wm*Wn,Acc,Wx,Wxd,Wy,Wyd	-(Multiply Wn by Wm) to Accumulator
MSC Wm*Wn,Acc,Wx,Wxd,Wy,Wyd,AWB	Multiply and Subtract from Accumulator
NEG Acc	Negate Accumulator
SAC Acc,#Slit4,Wdo	[Shift then] Store Accumulator *
SAC.R Acc,#Slit4,Wdo	[Shift then] Store Rounded Accumulator *
SFTAC Acc,#Slit5	Arithmetic Shift by Slit5 Accumulator
SFTAC Acc,Wn	Arithmetic Shift by (Wn) Accumulator
SUB Acc	Subtract Accumulators

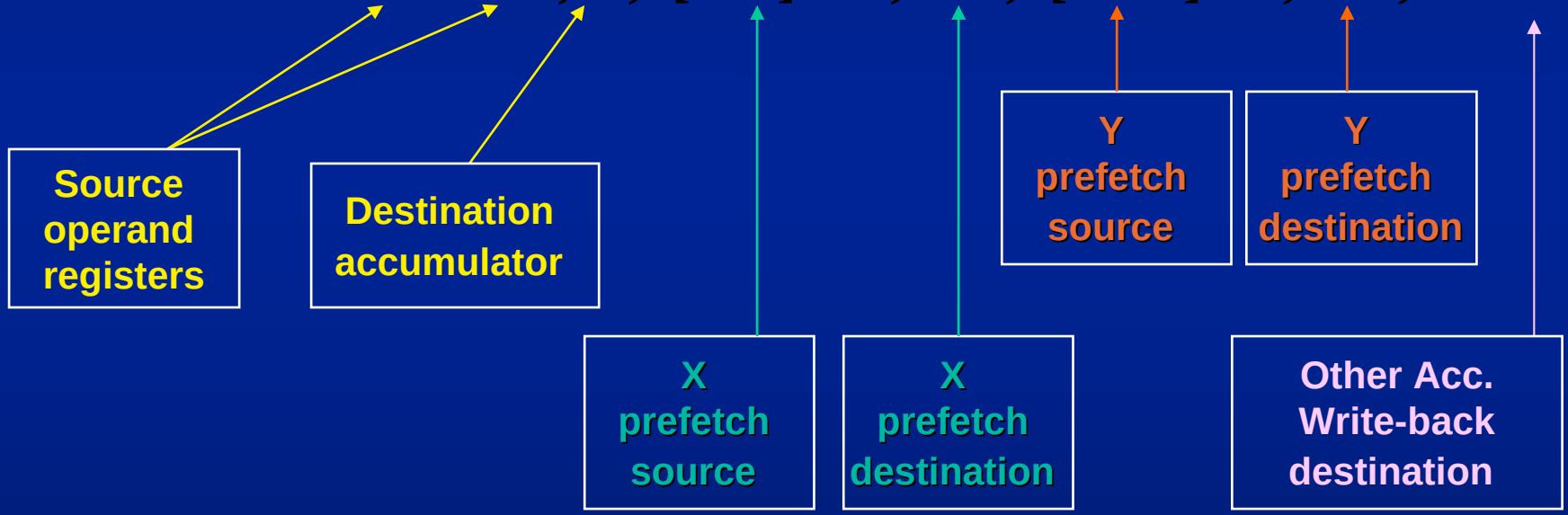
Note: Instructions in yellow, form the MAC class of DSP instructions. All instructions execute in 1 cycle and occupy 1 instruction word. Optional shift operations indicated by *

MAC 指令功能說明 (DSP 的指令)

- 底下的執行動作都在同一指令週期下完成：
  - 1 operation on ACCA (or ACCB) with data fetched into W register(s) from a previous cycle
  - 2 data fetches - 1 each from X- and Y-DS
  - 2 updates to the X and Y EAs used for the data fetch operations
  - 1 write back (WB) of the “other” accumulator, ACCB (or ACCA), to X-DS

MAC 指令解析(一)

- 指令範例：
  - **MAC W4*W5, A, [W8]+=2, W4, [W10]-=6, W5, W13**



MAC 指令解析(二)

- 範例說明 : MPY 及 MAC 指令
  - **MPY W4*W5, A, [W8], W4, [W10], W5, [W13]+=2**
  - **MAC W5*W6, B, [W9]-=6, W4, [W11+W12], W7**
- Data fetches are optional and may be made to only one bus, if so desired
- Accumulator WB is also optional
  - **MPY A, W4*W7 ; no pre-fetch/base syntax**

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

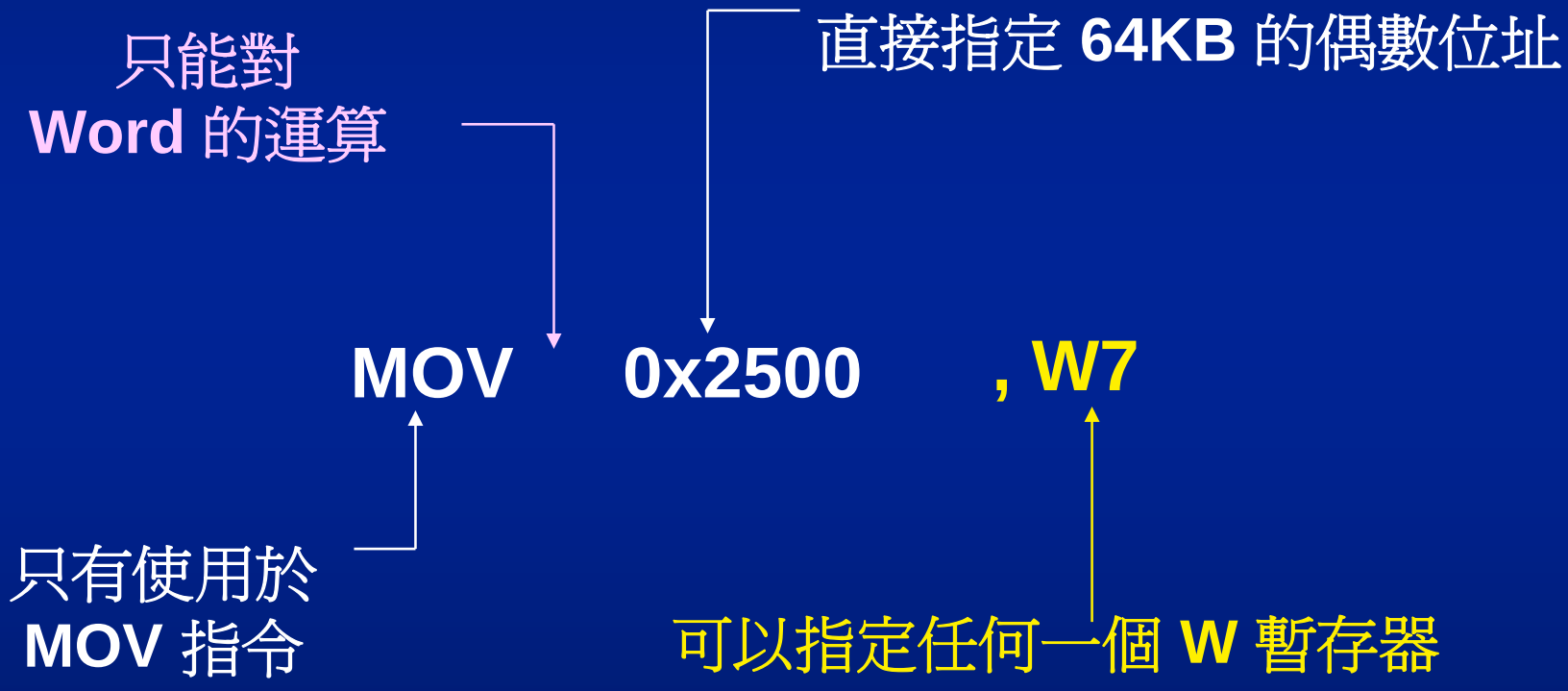
ADD{.b} 0x900 , {WREG}

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

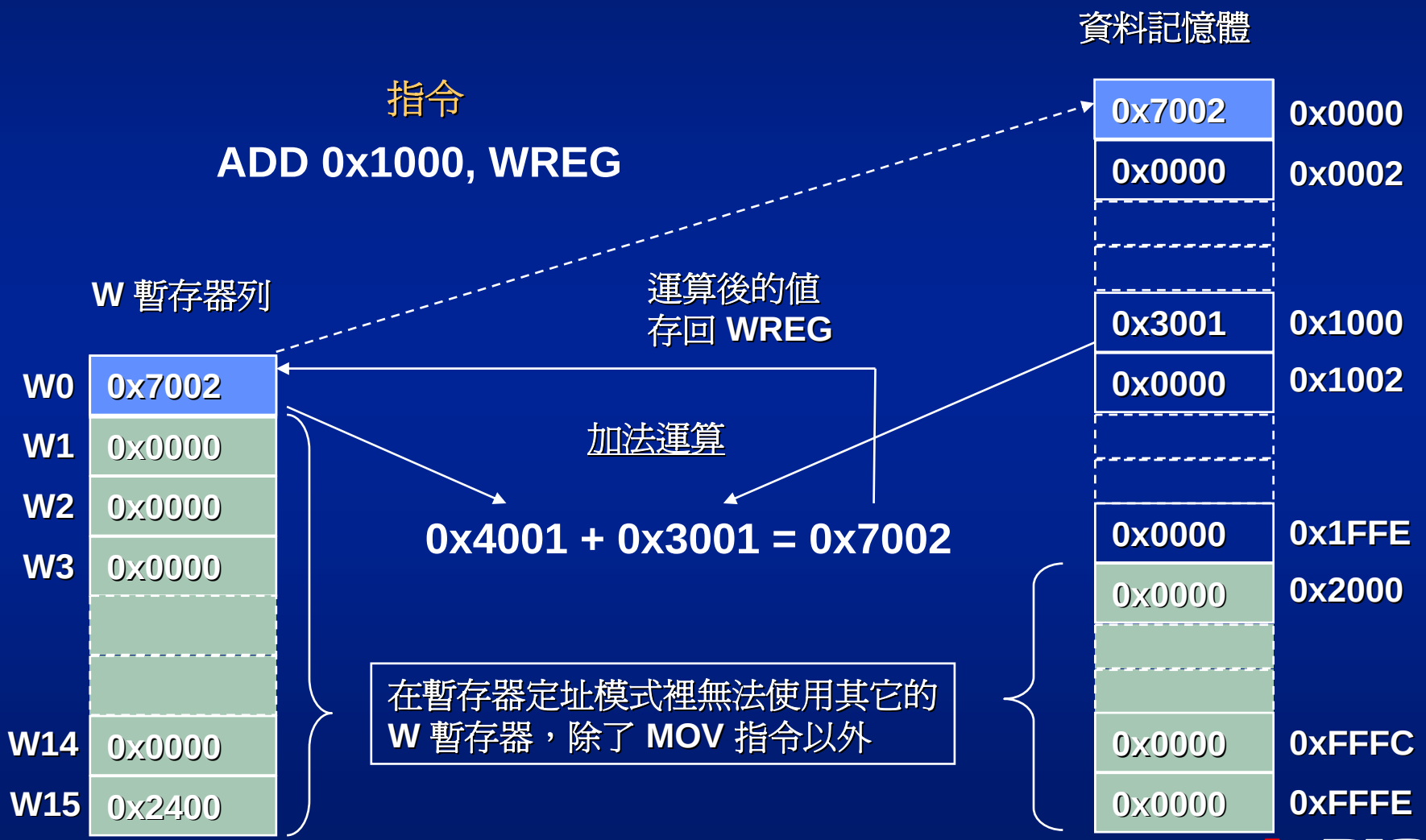
暫存器定址模式：使用 MOV 指令的定址方式



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

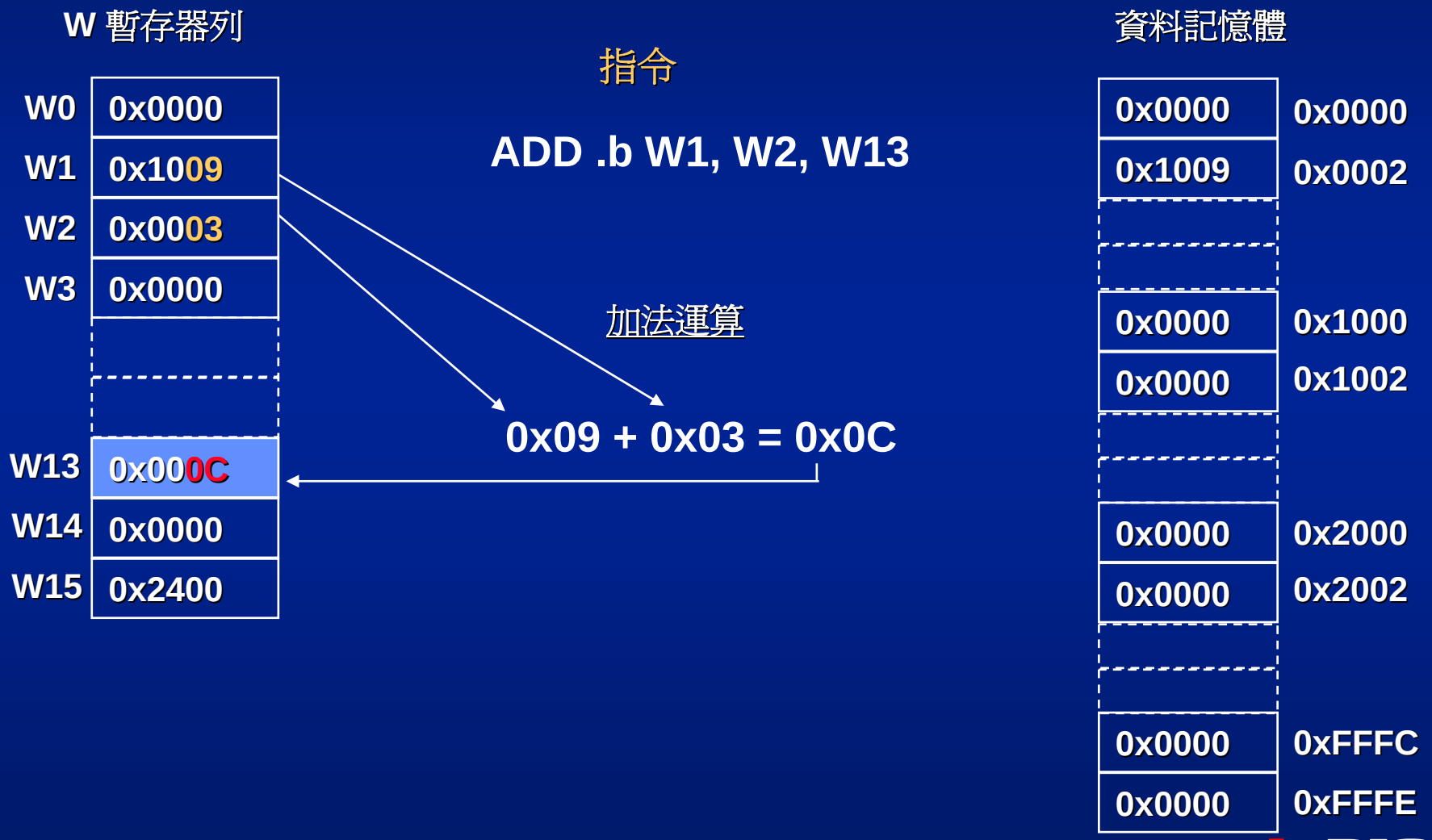
```
        MOV     0x900, WREG     ;Copy data at 0x900 to W0
        MOV     WREG, MYBUF     ;Copy W0 to MYBUF
```

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

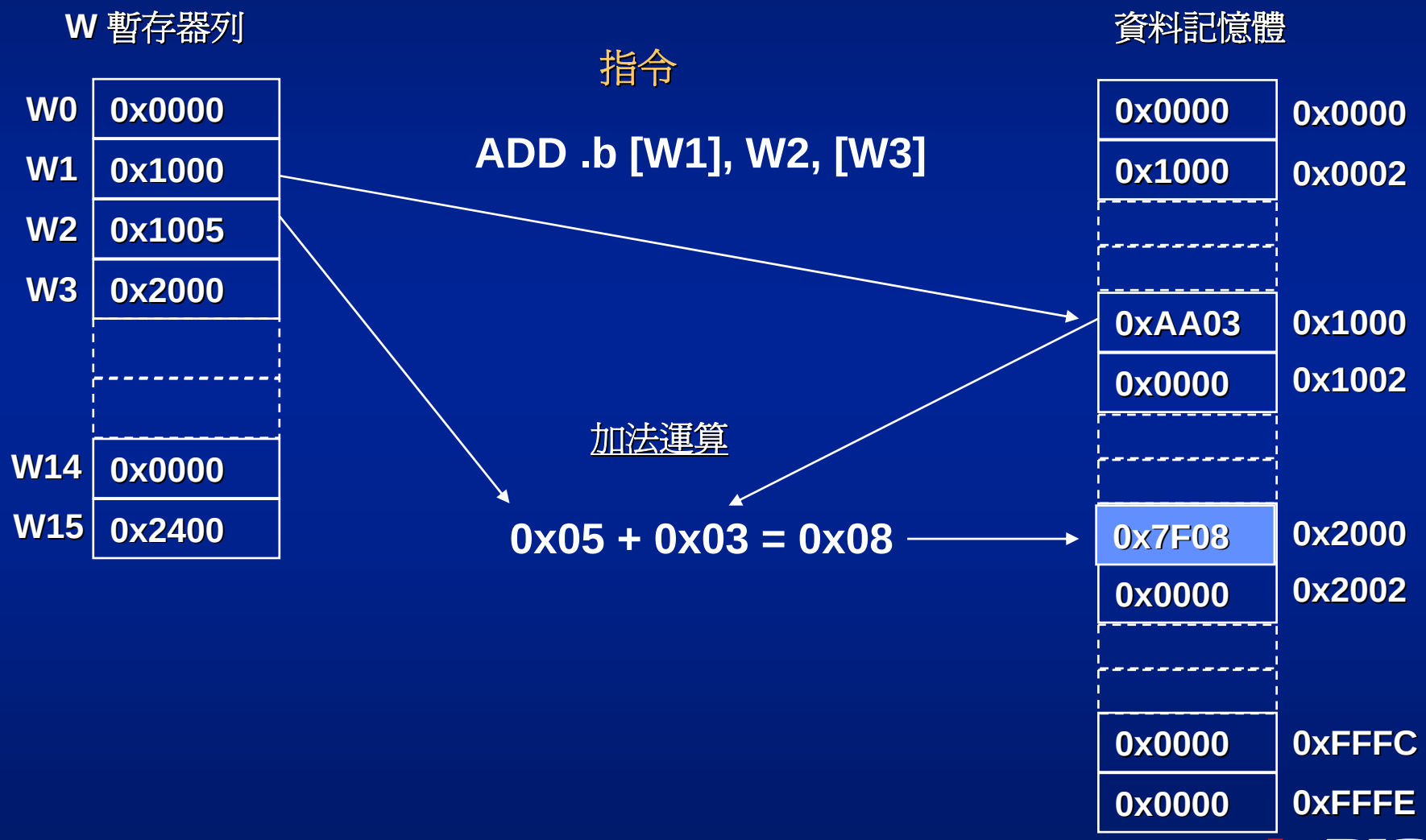


暫存器間接定址模式

- 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

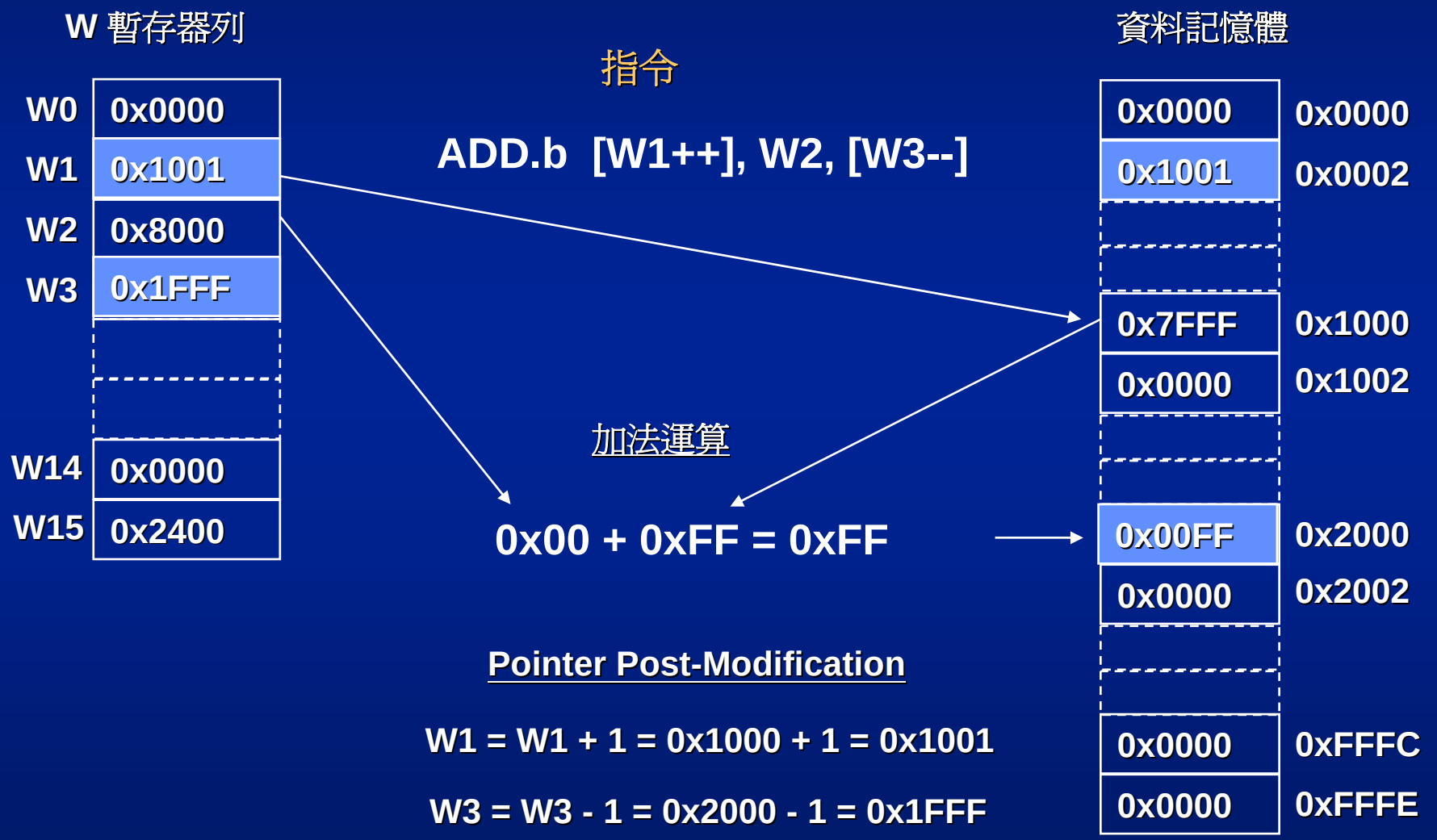


暫存器間接定址模式 – Post-Modification

- 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

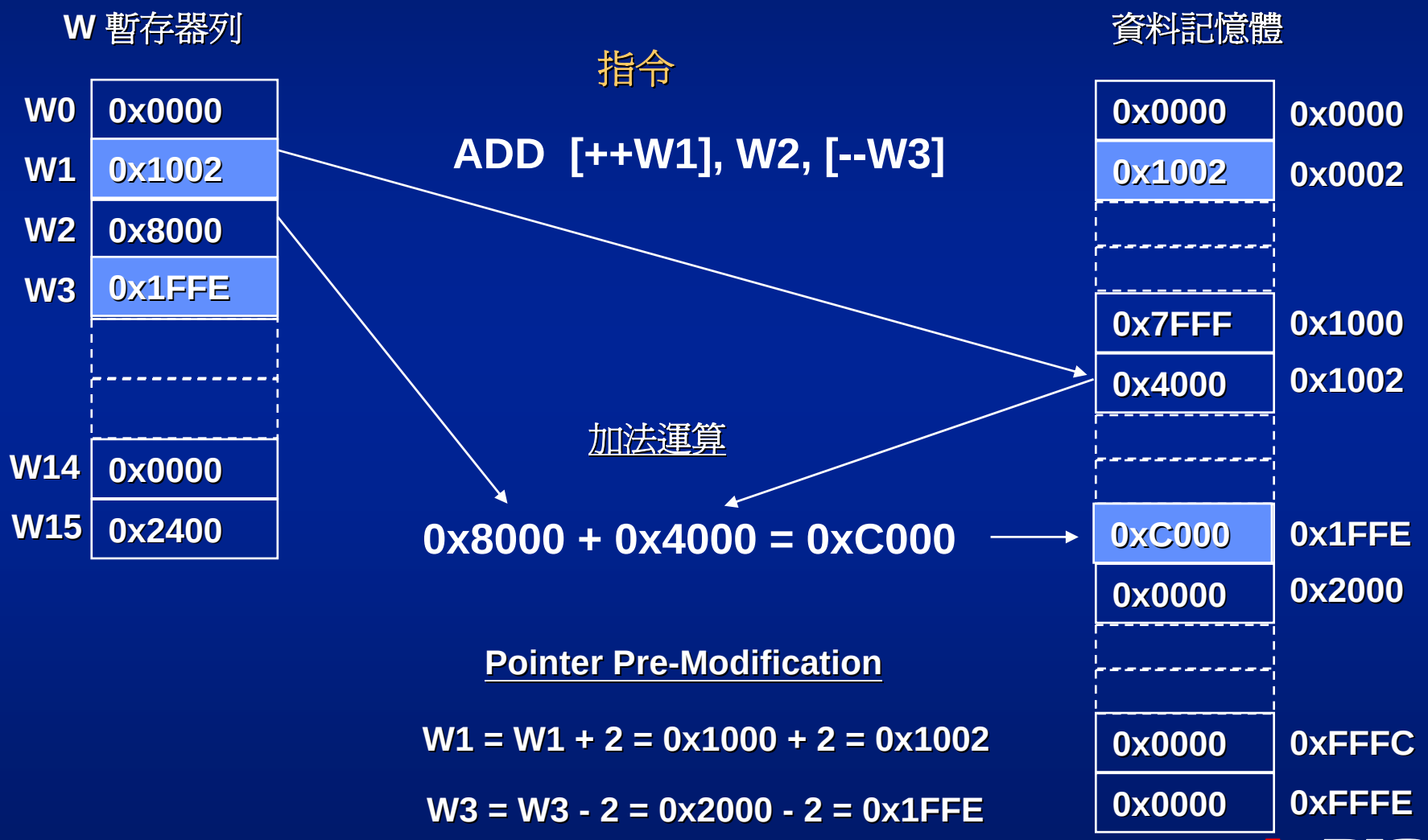
間接定址 Post-Modification: Byte Access



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

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


W 暫存器列
W0 0x0000
W1 0x1000
W2 0x0003
W3 0x0000
W13 0x101E
W14 0x0000
W15 0x2400
指令
ADD W1, #30, W13
加法運算
0x1000 + 0x1E = 0x101E
資料記憶體
0x0000 0x0000
0x1000 0x0002
0x0000 0x1000
0x0000 0x1002
0x0000 0x2000
0x0000 0x2002
0x0000 0xFFFC
0x0000 0xFFFE

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, 1023]，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
```

Modulo 定址方式：設定

- Modulo 定址方式在 MODCON 暫存器裡設定
  - XMODEN 爲 X-Modulo 致能控制位元
  - YMODEN 爲 Y-Modulo 致能控制位元
- 可以選一個 Wn 暫存器做爲 X 或 Y-DS 的定址暫存器

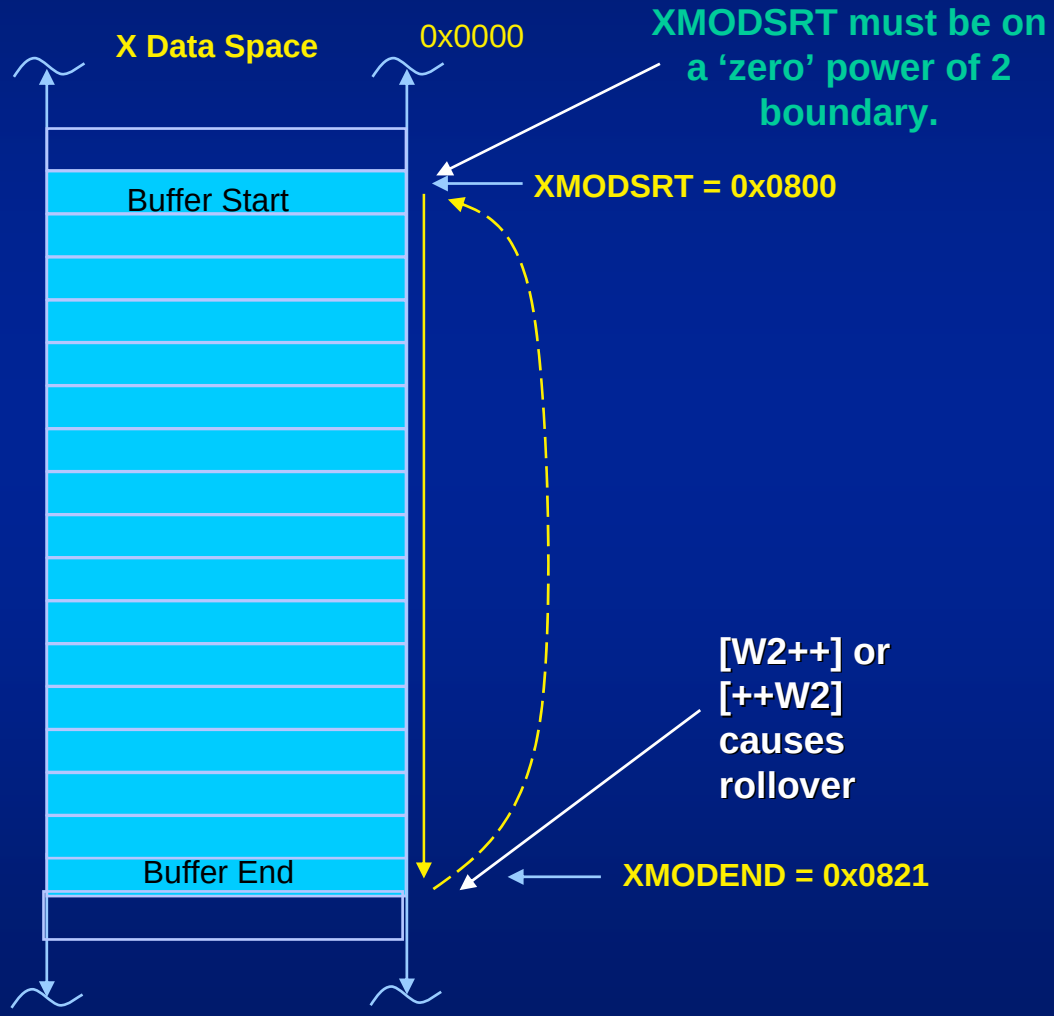
MODCON REGISTER

Modulo 定址方式：位址增加的循環模式

範例說明:

- 循環資料區長度 = 17 words (0x0800 - 0x0821 = 0x22 bytes = 17 words)
- W2 是資料位址的指標
- 使用 X Modulus
- Y Modulus 禁能
- X Bit Reversed禁能
- MODCON = 0x8FF2
- XMODSRT = 0x0800
- XMODEND = 0x0821

```
MOV     #0x0800,W0
MOV     W0,XMODSRT
MOV     #0x0821,W0
MOV     W0,XMODEND
MOV     #0x8FF2,W0
MOV     W0,MODCON
MOV     #55AA,W0
MOV     #0x0800,W2
DO      #16,Loop
Loop:
MOV     W0,[W2++]    ; W2 =0x0800 when DO loop completes
```



Modulo 定址方式：位址減少的循環模式

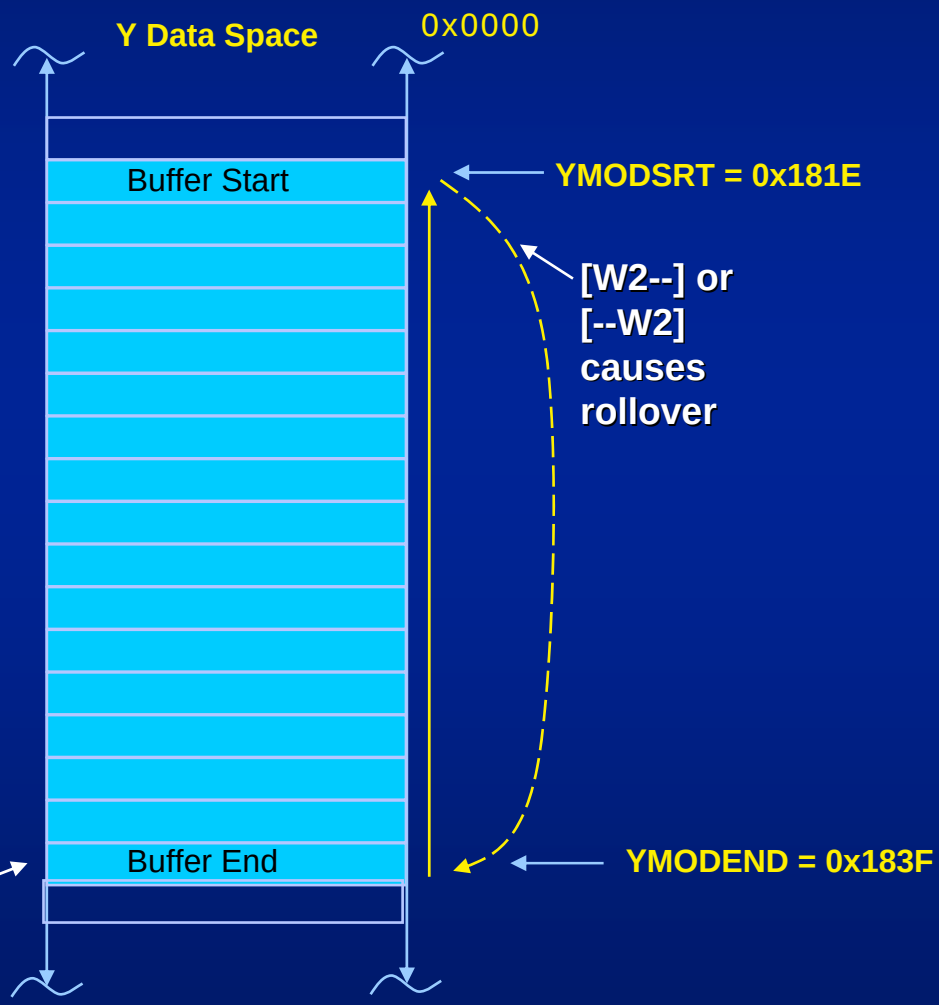
範例說明:

- 循環資料區長度 = 17 words (occupying memory region 0x181E - 0x183F)
- W2 是資料位址的指標
- Y Modulus Buffer
- Disable X Modulus
- Disable X Bit Reversed

- MODCON = 0x4F2F
- YMODSRT = 0x181E
- YMODEND = 0x183F

Note: Byte addresses are used for YMODSRT and YMODEND values.

YMODEND must be on a 'ones' power of 2 boundary.



Modulo 定址方式：雙向位址的循環模式

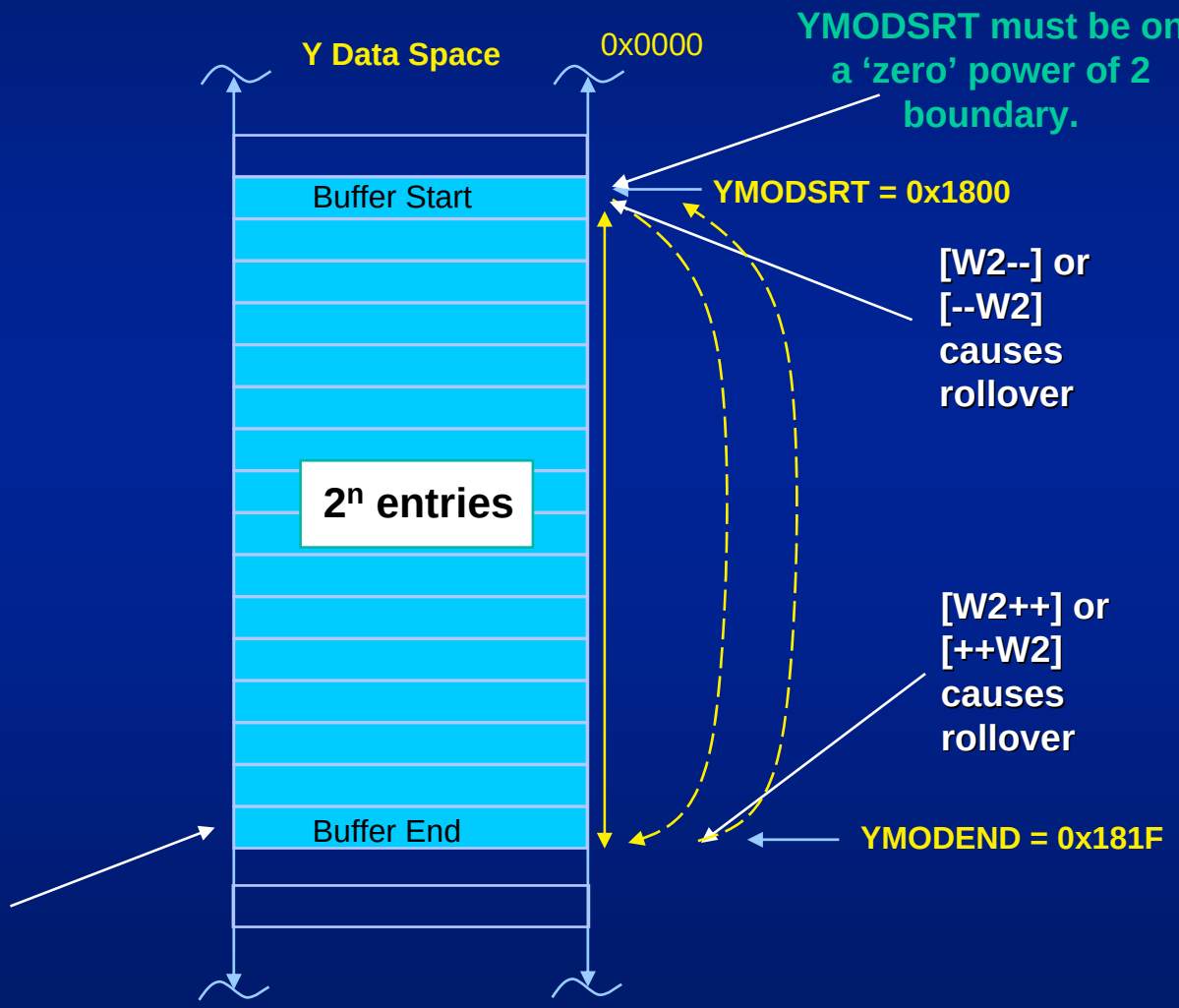
Example Specifications:

- Buffer Length = 16 words (occupying memory region 0x1800 - 0x181F)
- W2 is selected register
- Y Modulus Buffer
- Disable X Modulus
- Disable X Bit Reversed

- MODCON = 0x4F2F
- YMODSRT = 0x1800
- YMODEND = 0x181F

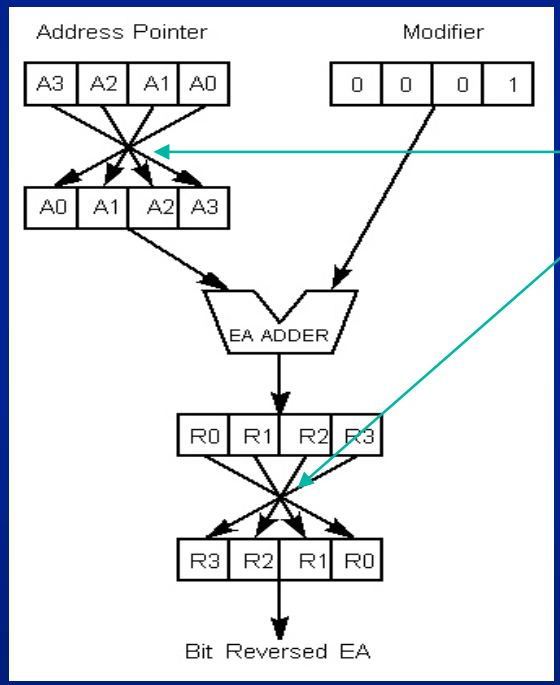
Note: Byte addresses are used for YMODSRT and YMODEND values.

YMODEND must be on a 'ones' power of 2 boundary.

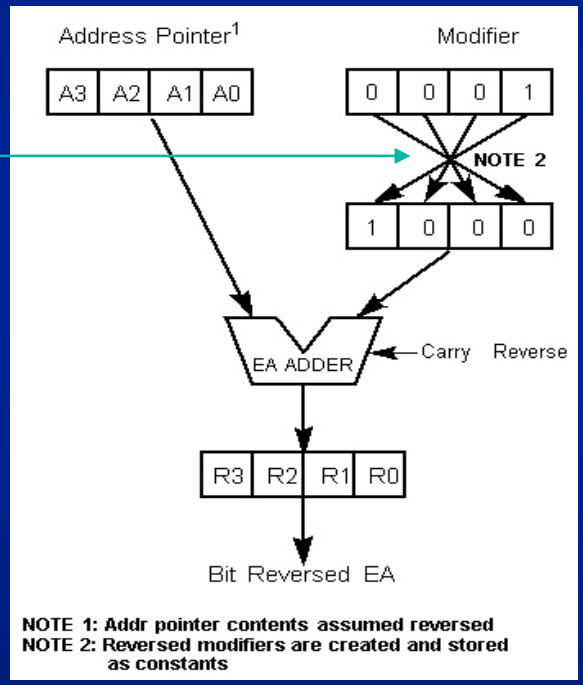


位元反轉定址模式 (Bit-Reversed)

- Bit-Reversed (BR) addressing is used to efficiently implement data addressing for radix-2 FFT



Pivot Point

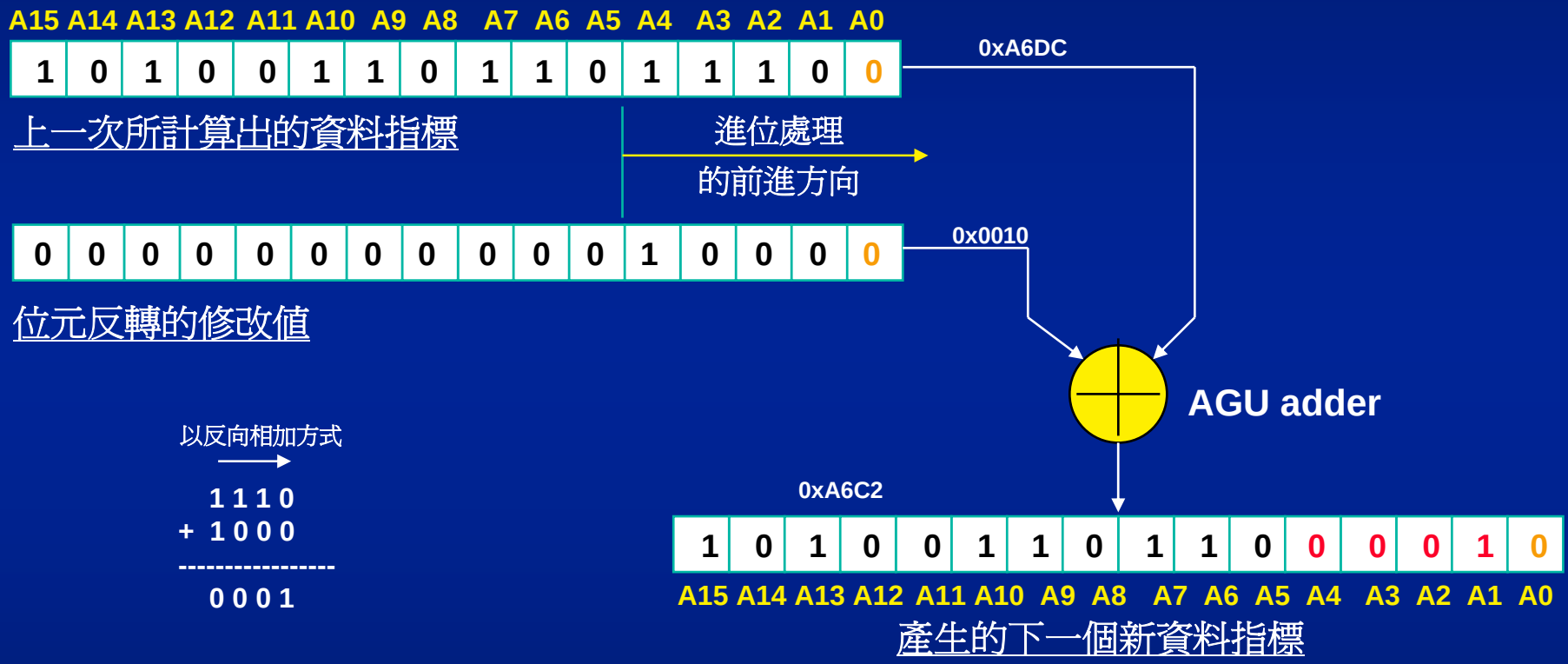


- BR modifier is added to the base W register, with direction of Carry propagation reversed, to generate the next EA

位元反轉定址模式順序 (16 - 輸入值)

Normal Address					Bit-Reversed Address				
A3	A2	A1	A0	decimal	A3	A2	A1	A0	decimal
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	8
0	0	1	0	2	0	1	0	0	4
0	0	1	1	3	1	1	0	0	12
0	1	0	0	4	0	0	1	0	2
0	1	0	1	5	1	0	1	0	10
0	1	1	0	6	0	1	1	0	6
0	1	1	1	7	1	1	1	0	14
1	0	0	0	8	0	0	0	1	1
1	0	0	1	9	1	0	0	1	9
1	0	1	0	10	0	1	0	1	5
1	0	1	1	11	1	1	0	1	13
1	1	0	0	12	0	0	1	1	3
1	1	0	1	13	1	0	1	1	11
1	1	1	0	14	0	1	1	1	7
1	1	1	1	15	1	1	1	1	15

位元反轉定址模式
16 位元的範例 (一)



- Supported only in word mode write operations carried out in X data space
- BR addressing assumes priority over Modulo addressing for write operations, when the two are simultaneously enabled

位元反轉定址模式 16 位元的範例 (二)

線性位址的順序

(0)	0xA6C0
(1)	0xA6C2
(2)	0xA6C4
(3)	0xA6C6
(4)	0xA6C8
(5)	0xA6CA
(6)	0xA6CC
(7)	0xA6CE
(8)	0xA6D0
(9)	0xA6D2
(10)	0xA6D4
(11)	0xA6D6
(12)	0xA6D8
(13)	0xA6DA
(14)	0xA6DC
(15)	0xA6DE

範例：

現在的指標 = 0xA6DC

修改值 = XB*2 = 0x0010

```
    1010 0110 1101 1100   (0xA6DC)
+   0000 0000 0001 0000   (0x0010)
-------------------------------------
    1010 0110 1100 0010   (0xA6C2)
-------------------------------------
```

進位的方向 >>>>>

修正後的指標 = 0xA6C2

位元反轉的順序

(0)	0xA6C0
(8)	0xA6D0
(4)	0xA6C8
(12)	0xA6D8
(2)	0xA6C4
(10)	0xA6D4
(6)	0xA6CC
(14)	0xA6DC
(1)	0xA6C2
(9)	0xA6D2
(5)	0xA6CA
(13)	0xA6DA
(3)	0xA6C6
(11)	0xA6D6
(7)	0xA6CE
(15)	0xA6DE

位元反轉定址模式 - 設定

- 位元反轉模式可以設定 MODCON 和 XBREV 暫存器
  - BREN (XBREV<15>) 打開位元反轉定址功能
  - BWM (MODCON<11:9>) 選擇那一個 W 暫存器做爲位元反轉定址
  - XB (XBREV<14:0>) 選擇位元反轉定址的修改值 (Modifier value)

BREN - Bit Reversed Addressing Enable

XB<14:0> - X AGU Bit Reversed Modifier

XBREV REGISTER

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)


主中斷向量表位址分配
第二中斷向量表
Reset - GOTO Instruction
Reset - GOTO Address
Reserved
Oscillator Fail Trap
Address Error Trap
Stack Error Trap
Math Error Trap
Reserved
Reserved
Reserved
Interrupt Vector 0
Interrupt Vector 1
Interrupt Vector 2
Interrupt Vector 53
Reserved
Reserved
Reserved
Oscillator Fail Trap
Interrupt Vector 53
0x000000
0x000002
0x000004
15
14
13
12
11
10
9
8
非遮罩式
中斷向量
內定的優先順序
中斷優先權往下遞減
0x00007E
0x000080
0x000082
0x000084
0x0000FE

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)

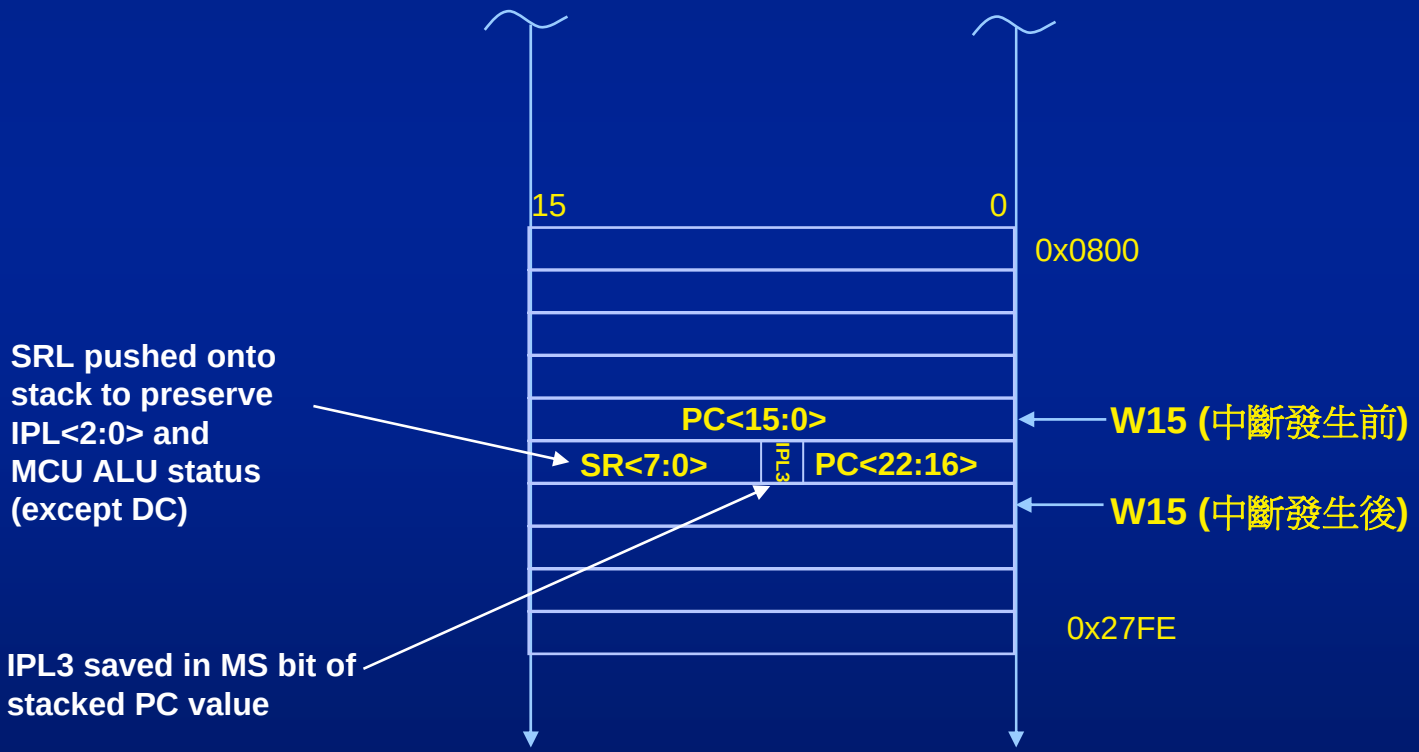
- 中斷源的優先權等級必須大於 CPU 的等級設定 (IPL<3:0>) 才有能力中斷 CPU
- IPL<2:0> 可以用軟體設定以變更 CPU 優先權
- IPL<3> 僅能被軟體讀取，它無法被禁能
- IPL 位元在中斷程式裡是可以被變更的
- 中斷發生時，原先 IPL<3:0> 的值會被推入軟體堆疊裡
- 中斷向量表有內定的先後順序設定以解決中斷相互衝突事件
- 內定的先後順序設定可以重新設定其優先權

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



中斷的巢狀式處理

- 中斷的巢狀式處理
  - 中斷巢狀設定內定是致能的
  - 中斷巢狀設定可藉由設定 NSTDIS 控制位元 (INTCON1<15>)來關閉此功能
  - 如果 NSTDIS = 1 ，CPU 中斷等級將會被設爲 7 以關閉全部的中斷輸入源

Traps 讓系統更穩定 (一)

- 軟體中斷 (Traps)
  - 中斷優先順序從 8 ~ 12
  - 可以說是固定優先權的 NMI 中斷
- 硬體錯誤中斷 (Traps)
  - 中斷優先順序從 13 ~ 15
  - 事件發生時，CPU 立即執行此中斷事件
  - Execution cannot resume until trap is acknowledged

Traps 讓系統更穩定 (二)

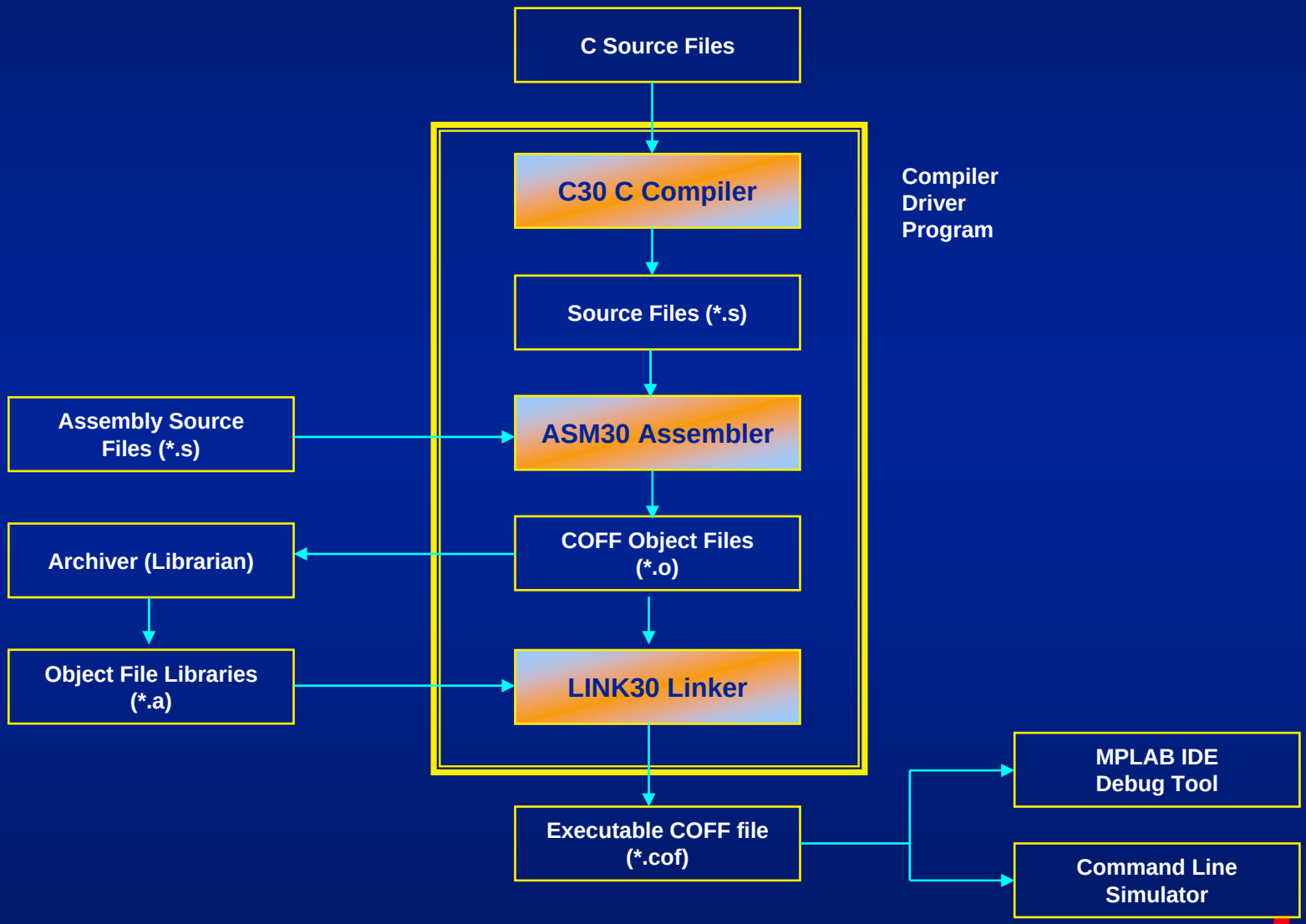
- 軟體中斷 (Traps)
  - 數學運算錯誤 (中斷等級 11)
    - 除數爲零 (商無限大)
    - 累積器移位時超出範圍
    - ACCA, ACCB 溢位 (optional)
    - 嚴重的錯誤溢位 ACCA or ACCB (optional)
  - 堆疊範圍錯誤 (等級 12)
- 硬體錯誤中斷 (Traps)
  - 位址錯誤 (中斷等級 13)
  - 振盪器功能異常 (中斷等級 14)

關閉中斷功能

- DISI 指令
  - DISI 指令可以暫時關閉等級 1 – 6 的中斷給一些特殊的需求
    - 14 位元的立即值 + 1 指令週期 --- 最多可有 16384 的禁止中斷週期
  - DISICNT 暫存器存有禁能計數值
  - 當 DISICNT 減爲 0 時，中斷功能將致能
  - DISICNT 的計數值可以更改以延長中斷禁能的時間
  - 可以直接清除 DISICNT 的計數值已脫離中斷禁能
  - DISI 狀態位元在 (INTCON2<14>)

使用 dsPIC ASM30 & LINK30

組譯工具處理流程


C Source Files
C30 C Compiler
Compiler Driver Program
Source Files (*.s)
Assembly Source Files (*.s)
ASM30 Assembler
Archiver (Librarian)
COFF Object Files (*.o)
Object File Libraries (*.a)
LINK30 Linker
Executable COFF file (*.cof)
MPLAB IDE Debug Tool
Command Line Simulator

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“
    - .include “C:\Program Files\MPLAB IDE \dsPIC_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，以最後一次定義的名稱爲準
- .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
```

資料節區位址的邊界點 .align

- 語法：.align　n
- 宣告以下的資料以 n 爲整除的資料節區起始位址
- N 通常是以 2 爲底的幕次方
- 適合使用在使用 modulo addressing 的起始位址設定

範例：

```
            .section .bss, "b"
            .align      8                   ; 從可以被 8 整除的位址開始
A_Var:      .space 6                        ; 保留 6 個 Bytes 的變數給 A_Var
```

程式節區位址的邊界點 .palign

- 語法：.palign n
- 宣告以下的資料以 n 爲整除的程式節區起始位址
- N 通常是以 2 爲底的幕次方

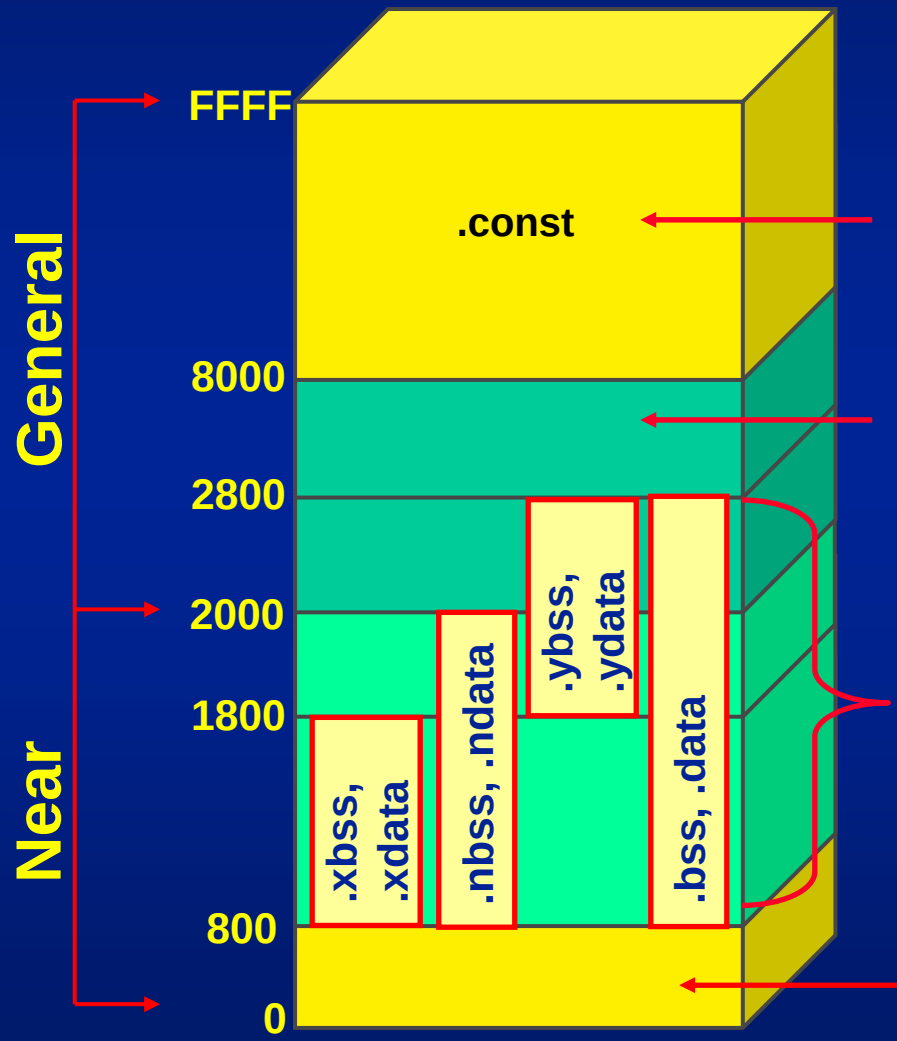
範例：

```
            .section    .My_Const_Data ,  “x"
            .palign     2                                ; 從偶數位址擺放資料
My_Dtat:    .hword      0x2000,0x3000,0x4000             ; 保留 6 個 Bytes 的變數給 A_Var
```

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

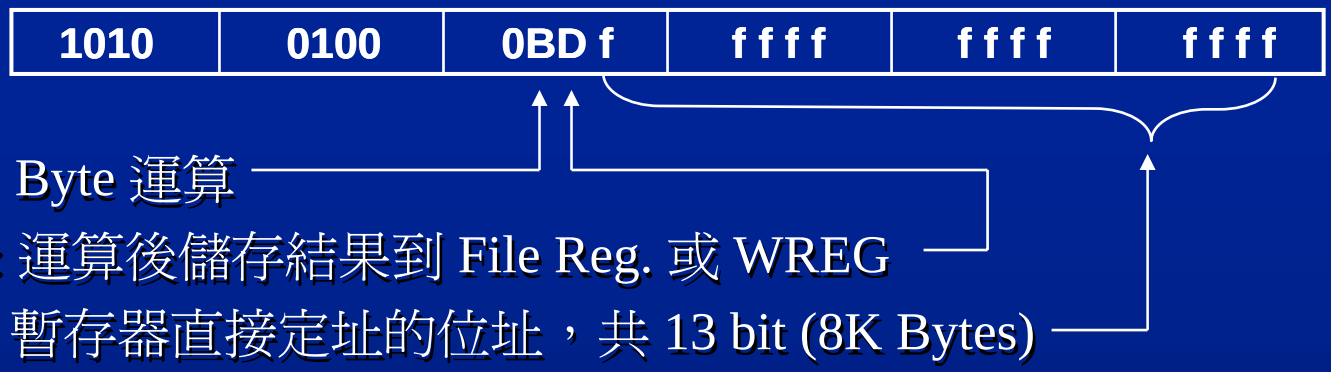
範例：

```
;.....................................................................
;          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

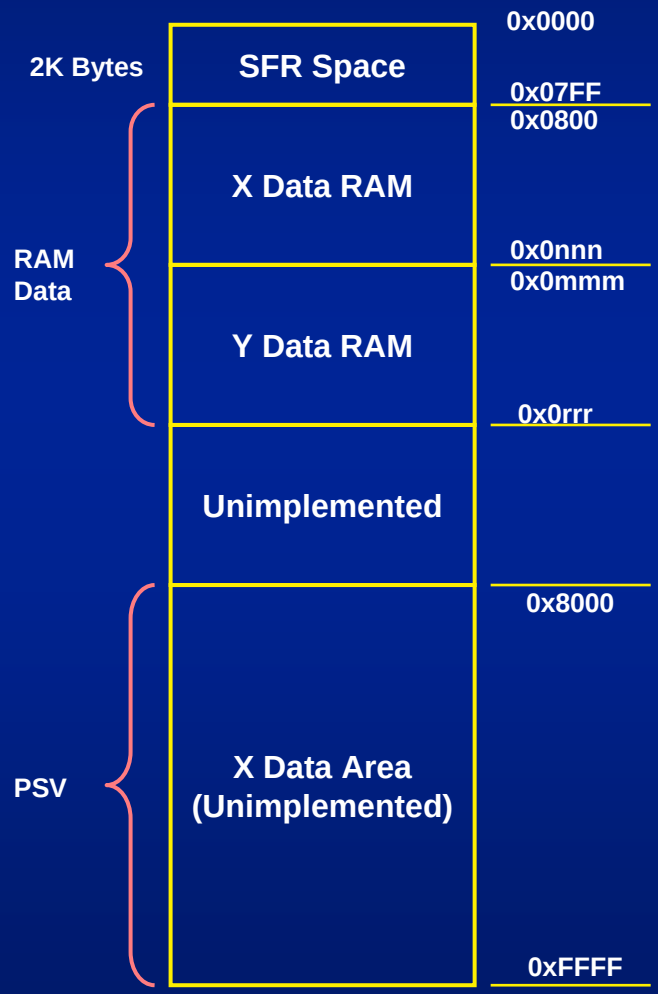
- 語法：.space *size* [,fill]
  - 保留“size”數量的 Bytes 給變數，如有給特定值 [fill] 則會被填入此保留空間

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\MPLAB IDE\dsPIC_Tools \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

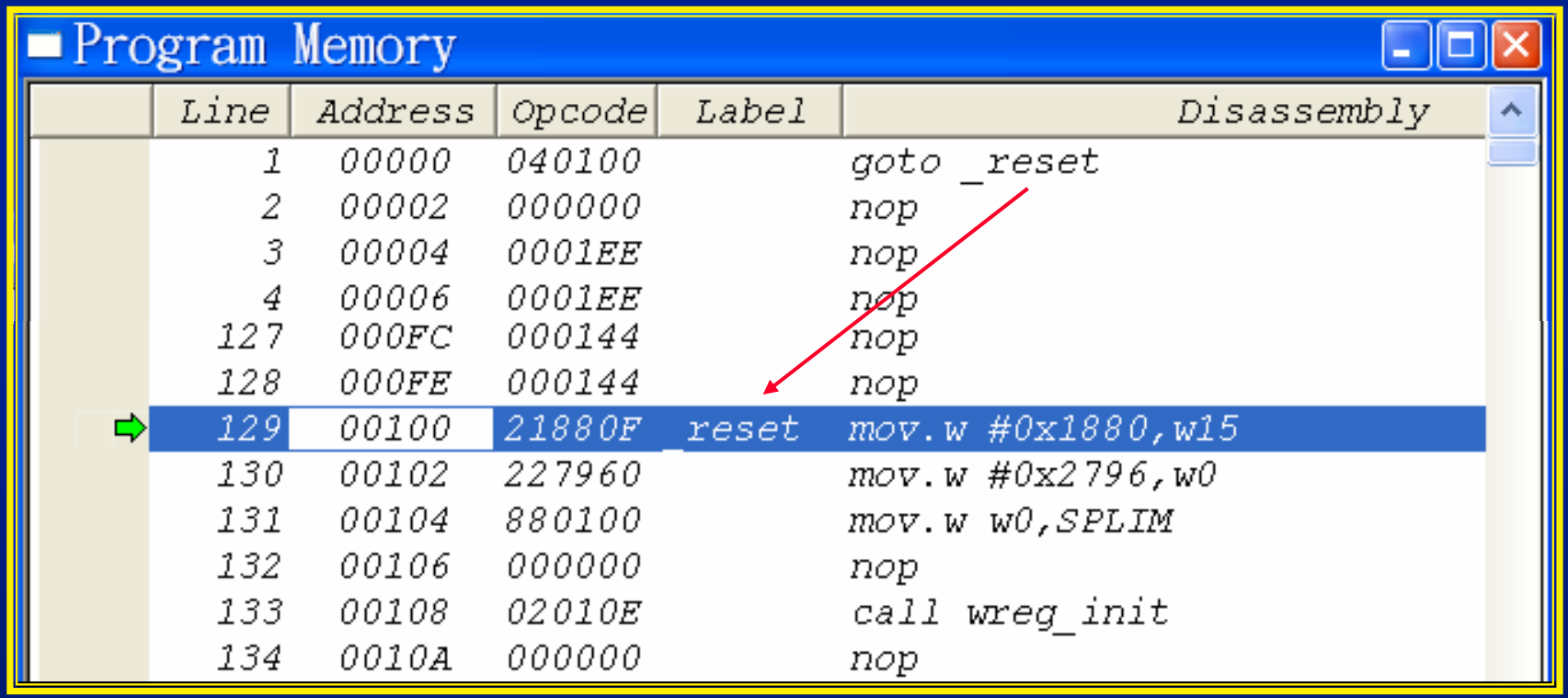
套用 tmp6010.s & tmp6014.s

- 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

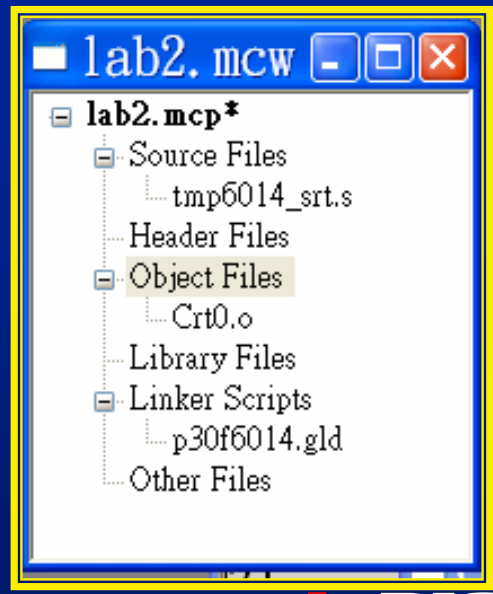
- 由下列的反組譯 (disassembly) 的顯示即可清楚的知道程式的擺放 (以上一頁程式爲例)



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.o

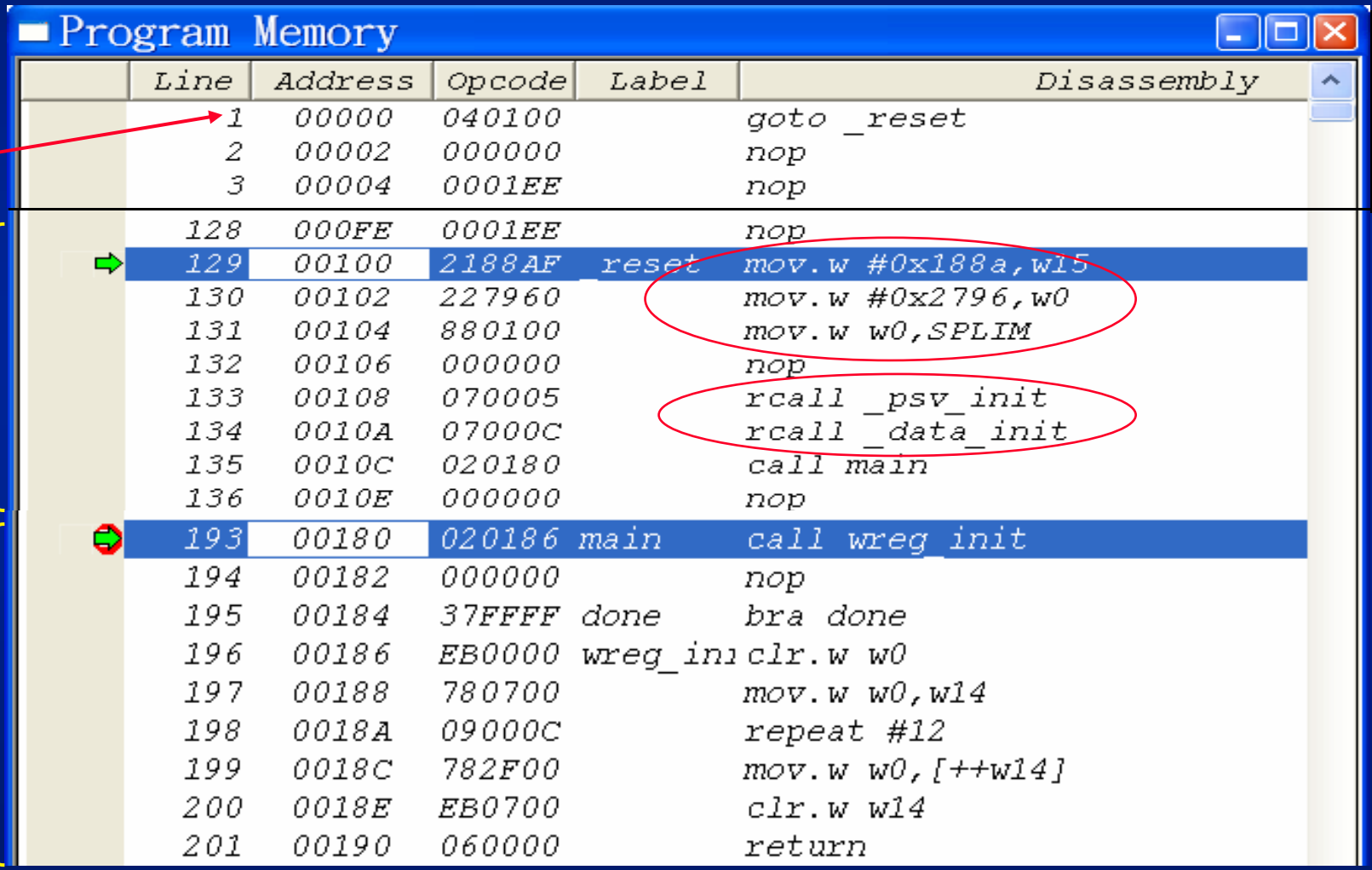
1. **Reset 後，控制權會先交給 Crt0.o 裡的 _reset**
2. 設定堆疊指標 **(W15 & SPLIM)**
3. 設定 **PSVPAG** 及 **CORCON (.const)**
4. 處理初始值的設定
5. 將控制權交給 **User’s** 程式裡的 **main** 標記



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

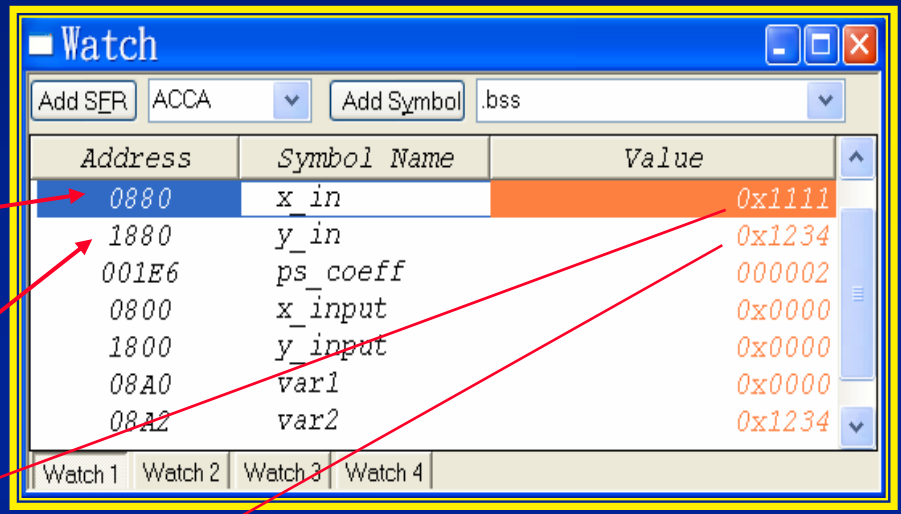
Main

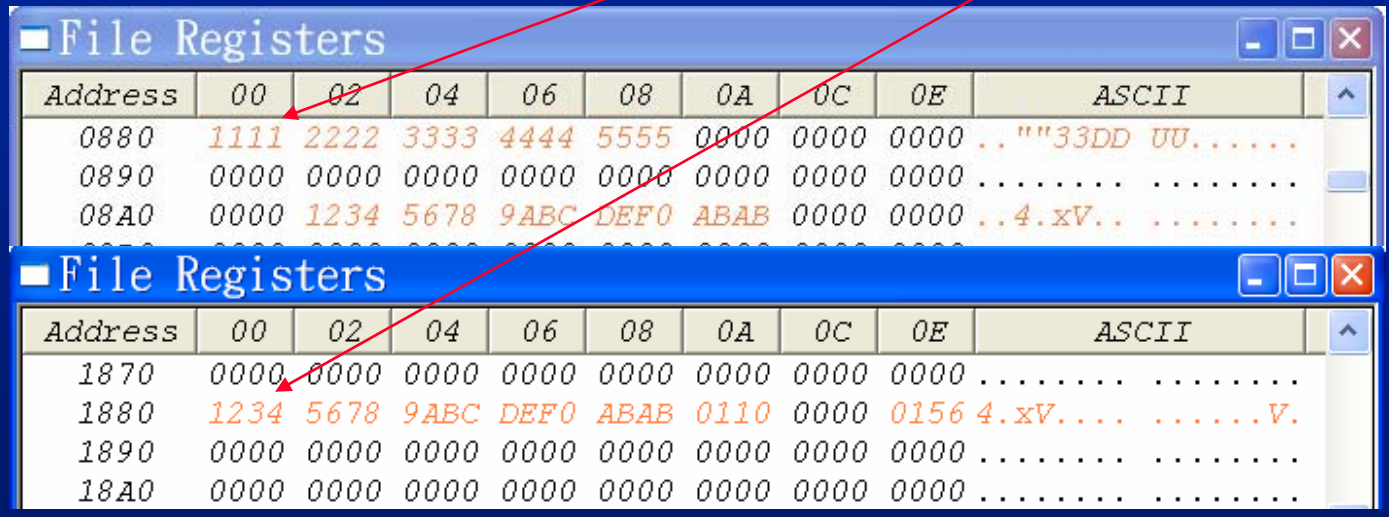
初始設定的資料安排
crt0.o

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)

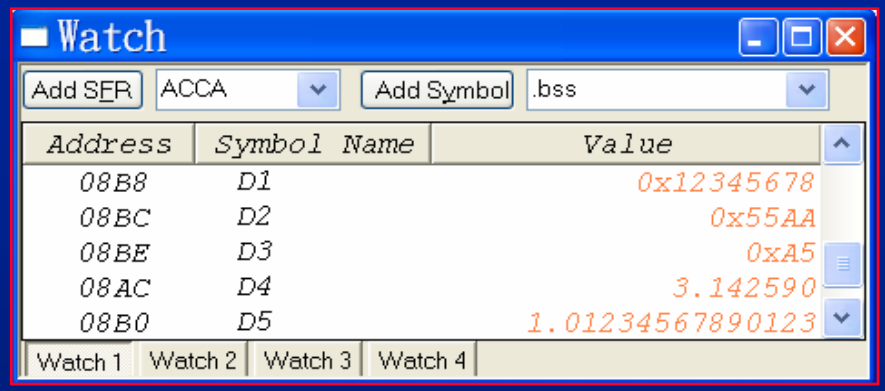




定義變數的初始值
.data & .ndata

- .section .data , “d”
  - 定義有初始值的變數在 RAM 的區域

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



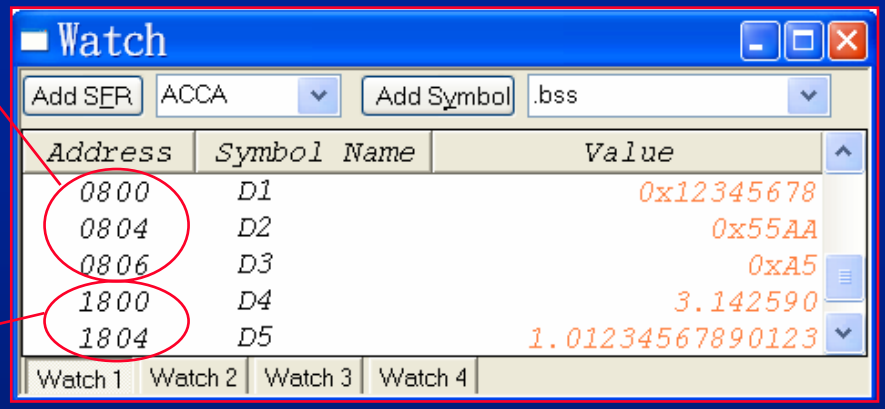
- .section .ndata , “d”
  - 定義有初始值變數在 near RAM

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

定義變數的初始值
.xdata & .ydata

- .section .xdata , “d”
  - 定義有初始值的變數在 X Data RAM 的區域

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



- .section .ydata , “d”
  - 定義有初始值變數在 Y Data RAM

```
        .section    .ydata,"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

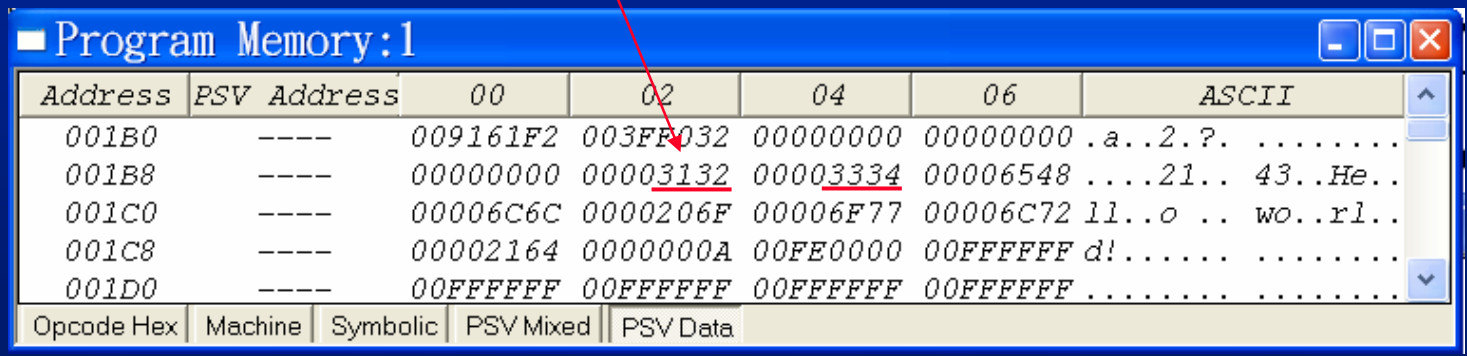
- 區域標記 (Local Label)
  - 定義近程的區域跳躍標記
  - 共10個標記可供使用
    - 0 ~ 9
  - 往前跳躍要加 b
  - 往後跳躍則要加 f

```
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 64KB 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
```

系統的可靠性設計：
振盪時序
電源管理

系統管理功能的特色

- 振盪模式的選擇 (可以設定Configuration Bits)
  - 內部最高工作頻率爲120MHz（相當於 30MIPS)
- 監視計時系統 (Watchdog Timer)
  - 只能在燒錄的時候決定要致能或禁能
  - 獨立專屬的內部 RC 振盪器
  - 可程式化溢時設定：2 ms - 16 sec
- 電源開機重置 (POR)
  - 可程式化開機延遲：0, 4, 16, 64ms
- 異常電壓的偵測 (BOR)
  - 可程式化異常電位偵測點的設定
- 低電壓偵測 (LVD)
  - 可程式化電壓位準設定及中斷能力

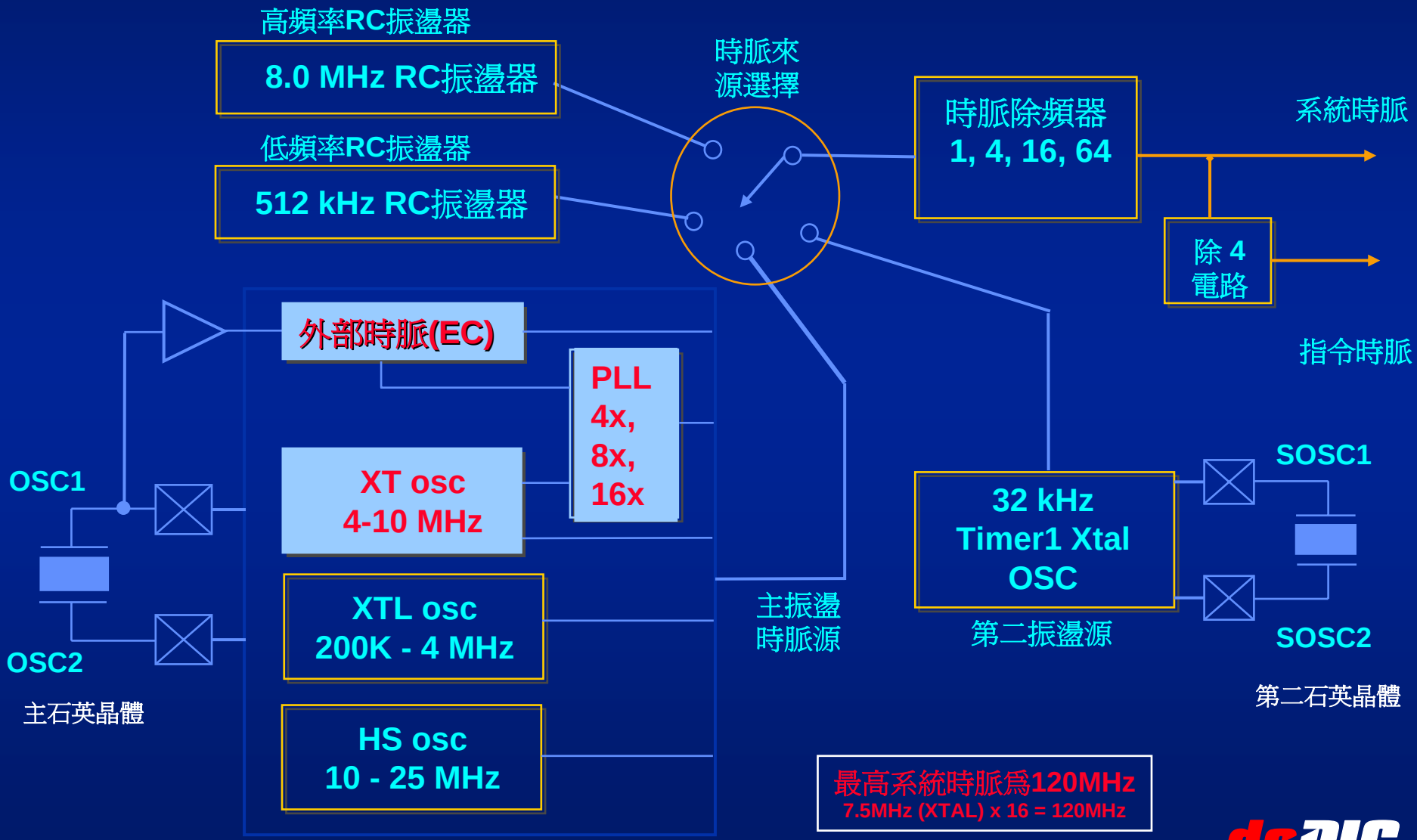
系統管理功能附加選項

- 即時的振盪時序的切換
- 振盪時脈除頻器 : 1, 4, 16, 64 (輸出爲系統時脈)
- 睡眠模式 (SLEEP mode) :
  - 系統時脈停止工作
  - CPU 停止工作
  - 週邊停止工作 ，除了一些特殊電路(WDT, INT ... 等)
  - ICD2 暫停工作
- 閒置模式 (IDLE mode) :
  - 系統時脈繼續工作
  - CPU 停止工作
  - 週邊繼續工作，可以利用設定相對應週邊的 SIDL 位元的方式將該周邊功能關閉

系統時脈來源

- 指令週期 = 系統時脈 / 4
- 系統時脈來源：
  - 外部石英振盪晶體選擇：
    - XTL: (低頻率石英振盪器) 200 KHz - 4 MHz
    - XT: (中頻率石英振盪器) 4 MHz - 10 MHz
    - HS: (高頻率石英振盪器) 10 MHz - 25 MHz
  - 外部 RC Up to 4 MHz
  - 外部時脈輸入 (EC): Up to 40 MHz
  - 內部高速 RC (FRC): 8 MHz
  - 內部低速低功耗 RC (LPRC): 512 KHz
  - 外部第二低速石英晶體振盪源： 32 KHz
- 內部鎖相倍頻電路 (PLL) (可選擇)：
  - XT 或 EC 輸入到 PLL: 4 MHz - 10 MHz 範圍
  - 4X , 8X or 16X: 16 MHz - 120 MHz 輸出範圍
  - 有些元件可以用 FRC 模式做 4X/8X/16X PLL 倍頻輸出

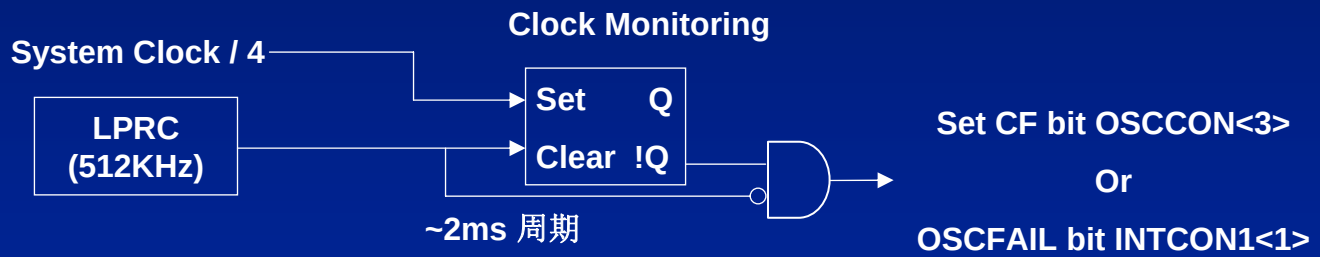
系統時序方塊

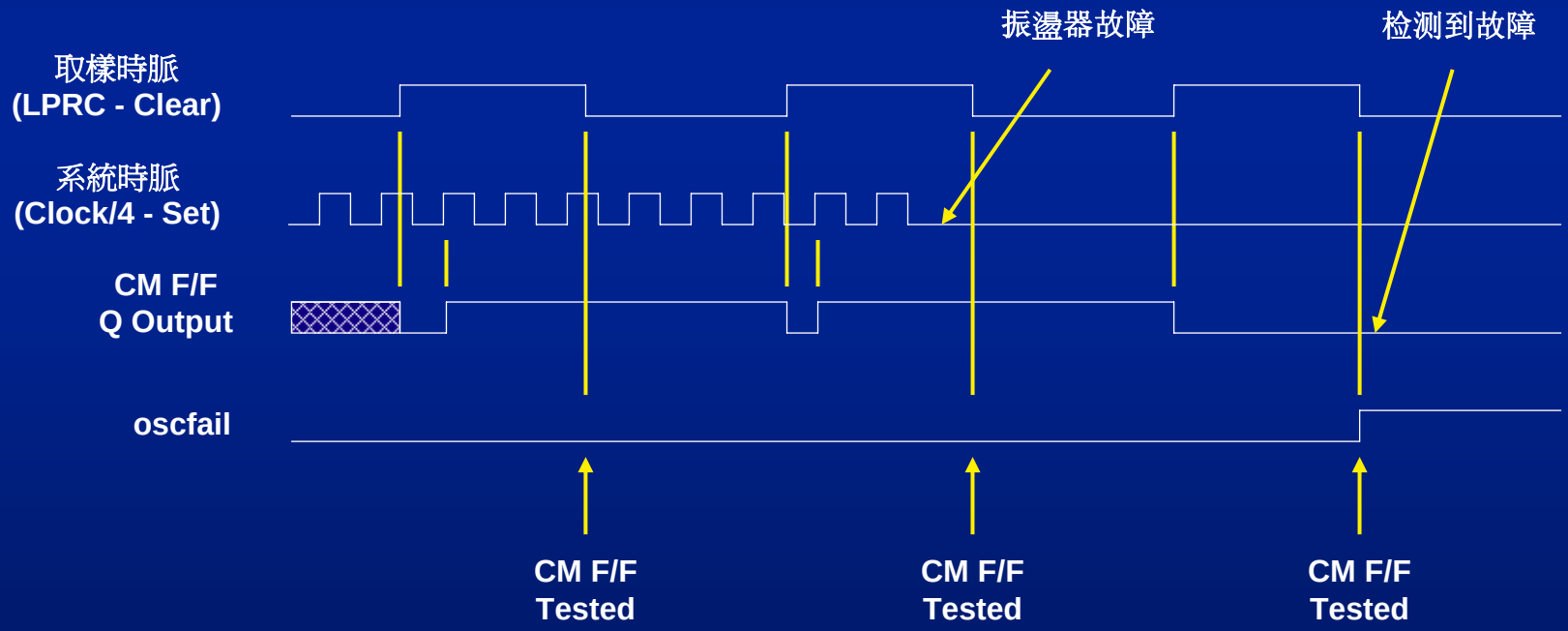

高頻率RC振盪器
8.0 MHz RC振盪器
低頻率RC振盪器
512 kHz RC振盪器
時脈來源選擇
時脈除頻器
1, 4, 16, 64
系統時脈
除 4 電路
指令時脈
外部時脈(EC)
PLL 4x, 8x, 16x
XT osc 4-10 MHz
XTL osc 200K - 4 MHz
HS osc 10 - 25 MHz
OSC1
OSC2
主石英晶體
主振盪時脈源
32 kHz Timer1 Xtal OSC
第二振盪源
SOSC1
SOSC2
第二石英晶體
最高系統時脈爲120MHz
7.5MHz (XTAL) x 16 = 120MHz

Fail-Safe 時脈監測器 FSCM

- FSCM 隨時監測系統的時脈是否故障
- 燒錄時 FCKSM 位元(FOSC<15>) 決定 FSCM 功能是否要啓動
- FSCM啓動後，內部 512KHz (LPRC) 將會持續工作
  - 除了在睡眠模式下
- FSCM 允許主時脈故障時切換到另一組振盪電路，並產生時脈錯誤的中斷 (Oscillator Fail Trap)
  - NOSC<1:0> 選擇故障時新時脈來源 (OSCCON Reg.)

FSCM: 它是如何工作的?

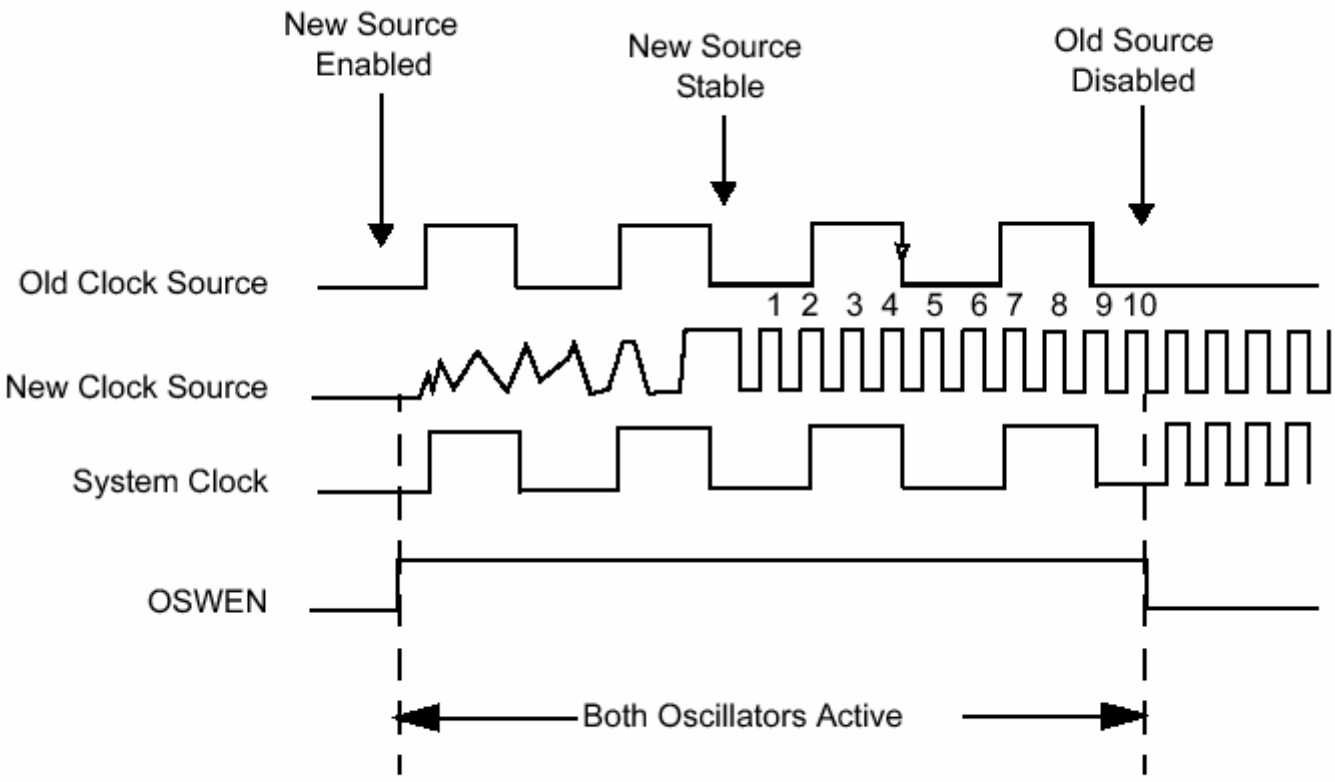




時脈的切換

- 時脈切換功能可以讓系統有執行效能與功耗有最佳的管理方案，mWatt/MIPS
- 時脈的切換可以獲得：
  - 獲取 CPU 最大的工作效率
  - 電流消耗的理想曲線
- Can not switch directly between PLL modes
  - Switch first to the FRC Oscillator then to new PLL
- 在時脈切換的過程中程式碼能繼續執行

時脈切換的時序圖



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

時脈的切換動作

- 確認時脈切換功能是致能的
  - FCKSM 位元是在 FOSC configuration register
- 時脈切換的動作需求
  - “Unlock” 先向 OSCCONH 暫存器發出解鎖命令
  - 寫入欲切換的新時脈選擇(NOSC bits OSCCON<8:9>)
  - “Unlock” 再向 OSCCONL 暫存器發出解鎖命令
  - 設定 OSWEN bit = 1 (OSCCON<0>)
  - 等待 OSWEN bit 被硬體清爲零
    - 如果 OSWEN 無法被清爲零的話，要跳離切換動作的話只要將 OSWEN 位元清除即可

OSCCON 解鎖的動作順序

```
DISI   #14                    ; 暫時停止中斷功能 (Level 0-6)
mov.b   #0x??,  w0
mov     #OSCCONH, w1
mov     #0x78,  w2            ; 載入OSCCONH 的解鎖值
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!

時脈及時監視器 (FSCM) 提供高可靠度的運作

- 當系統偵測到時脈發生錯誤？
  - 如果 FSCM 是致能的話，時脈自動切換到內部的 FRC 振盪器
  - FSCM 可藉由在 FOSC暫存器的 FCKSM bits 開啓功能
  - 產生時脈故障中斷
  - 程式的執行權交給時脈故障中斷向量 ，返回時必須完成：
    - Clear the Clock Fail bit, CF, in OSCCON
    - Clear the OSCFAIL trap flag in INTCON1
    - Execute a RETFIE

振盪器控制暫存器

- OSCCON
  - COSC<1:0> ⇒ 目前振盪器工作狀態顯示
  - NOSC<1:0> ⇒ 新振盪器切換選擇
  - POST<1:0> ⇒ 振盪器輸出除頻器選擇 (除 1,4,16,64)
  - LOCK ⇒ 鎖相 (PLL) 穩定狀態指示位元
  - CF ⇒ 時脈故障狀態指示位元
  - LPOSCEN ⇒ 32 KHz LP 振盪器致能
  - OSWEN ⇒ 振盪器切換置能控制位元

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

其它可靠性功能

- 計時監視器使用獨立的內部 RC 振盪器及燒錄設定
- 重置電路：
  - POR, Watch Dog Timer, BOR, Illegal Operation (Program Address)
  - RCON 暫存器會紀錄上一次的重置原因
- Traps:
  - Oscillator fail, Data Address, Stack, and Math errors
- Low VDD Detect (LVD) Interrupt
  - Programmable battery voltage level detect

振盪器燒錄選擇設定

- FOSC（Configuration 暫存器)
  - FCKSM<1:0> ⇒ 時脈切換功能設定
  - FOS<1:0> ⇒ 電源重置時，振盪器的來源選擇設定
  - FPR<3:0> ⇒設定主振盪器模式

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

主振盪器設定方式

- 使用 p30f6014.inc 所提供的巨集指令 config __FOSC 來定義 FOSC 暫存器的設定
  - 範例：

    config __FOSC, CSW_FSCM_ON & EC_PLL8
  - 詳細定義名稱請參考 p30F6014.inc

或

- 在 MPLAB IDE 的環境下選擇 Configure --> Configuration Bits 的對話視窗以設定工作模式