dsPIC30F Peripheral Module

How dsPIC30F Handles Interrupts

Agenda

- Get to know interrupt of dsPIC30F controller
  - dsPI30F Architecture Refresh
  - Interrupt Vector Table
  - Interrupt Priority
  - Traps
  - Interrupt Nesting
  - Context Saving
  - SLEEP and IDLE modes
  - Interrupting DO and REPEAT
  - Control Registers
  - Interrupt Timing
  - Interrupt Coding
  - Q and A discussion

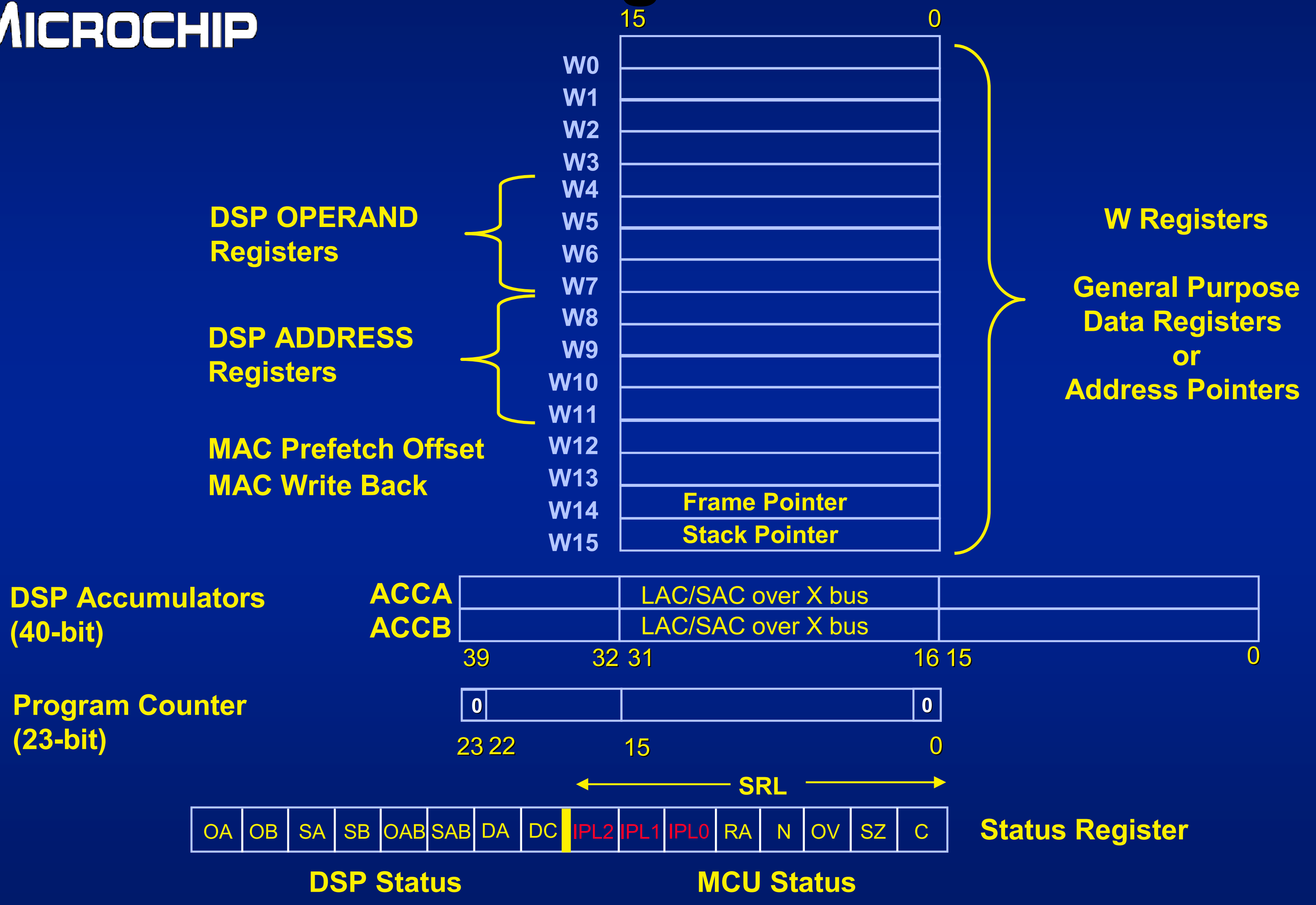
dsPIC® DSC Architecture

- Main Features
  - Single Core Integrating an MCU & a DSP
  - Modified Harvard Architecture
  - Data is 16-bit wide
  - Instruction is 24-bit Wide
  - Linear Program Memory up to 12 MB
  - Linear Data (RAM) up to 64 kB
  - True DSP Capability
  - Many Integrated Peripherals

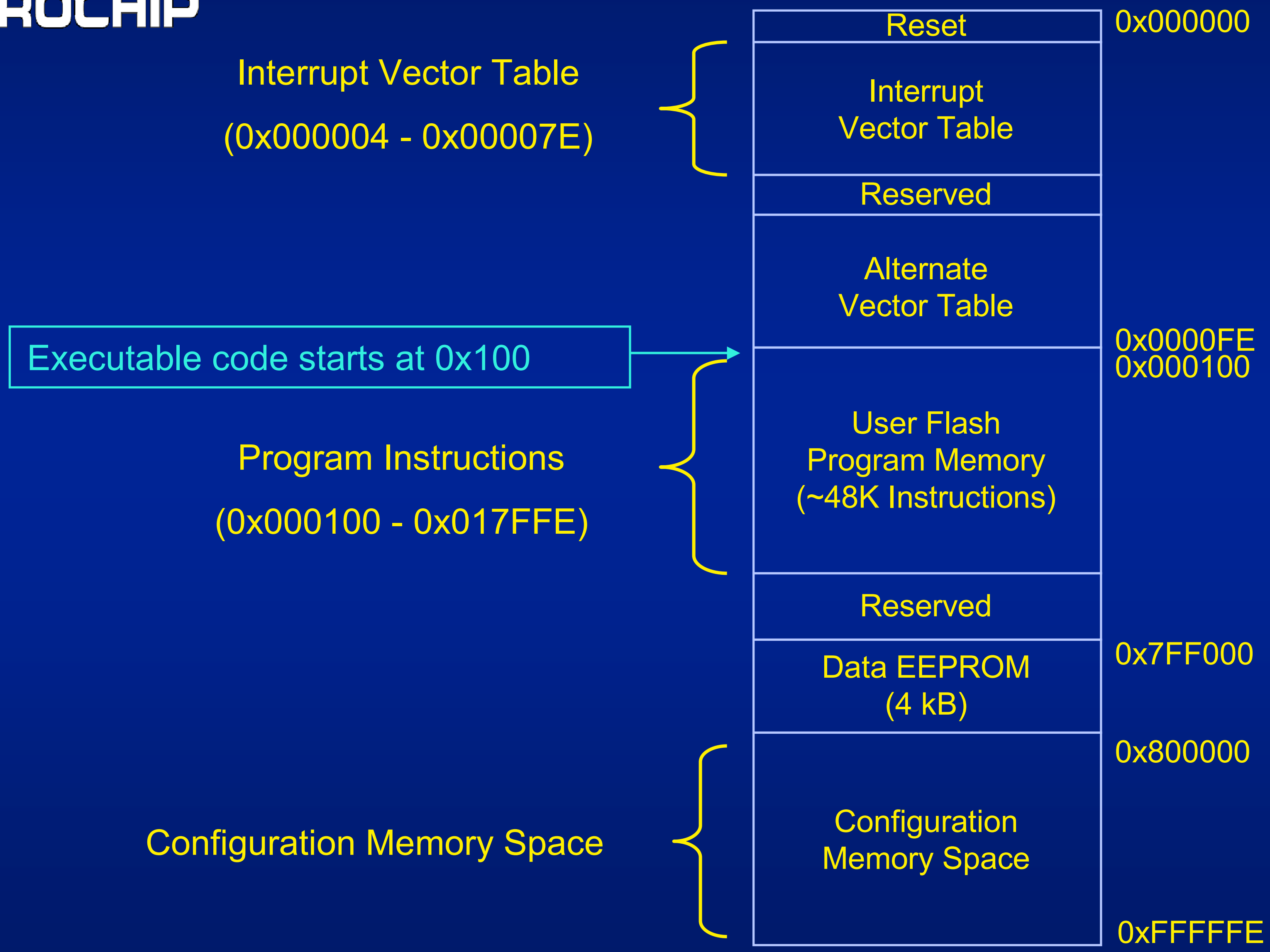
dsPIC® DSC Architecture

- Main Features (continued)
  - 16 x 16-bit Working Register Array
  - Software Stack
  - Fast, Deterministic Interrupt Response
  - Three Operand Instructions: C = A + B
  - Extensive Addressing Modes

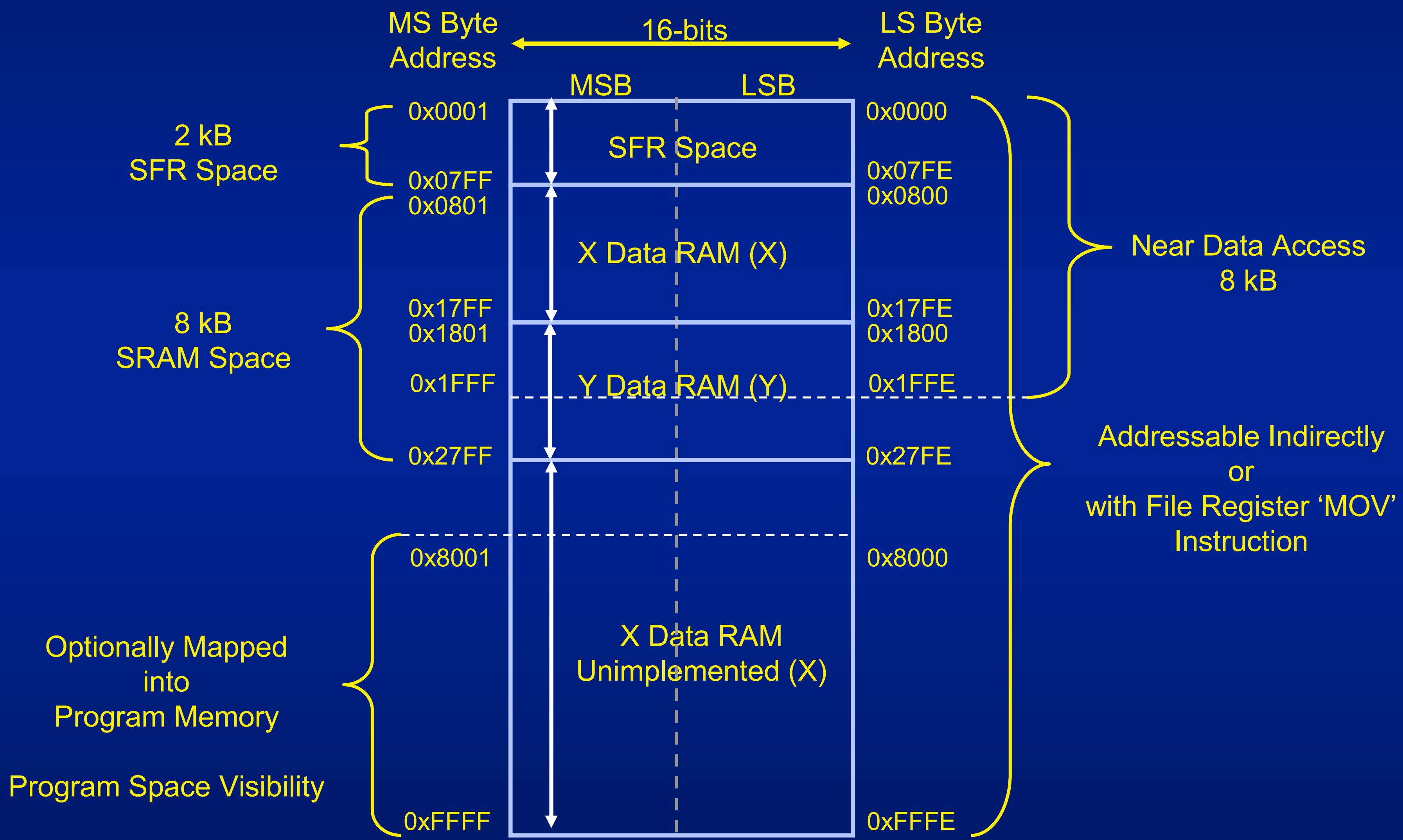
Programmers Model



Program Memory - dsPIC30F6014



Data Memory - dsPIC30F6014



dsPIC30F Interrupts Overview

- Interrupt Vector Table (IVT) with unique vector for each source
- Vector location contains ISR address
- 8 non-maskable trap vectors
- 54 interrupt vectors
- 8 user assigned priority levels
- Alternate IVT for diagnostics
- Consistent 5 cycle entry latency for all instructions
- 3 cycles RETFIE

Interrupt Vector Table

Interrupt Vector Table	Reset - GOTO Instruction	0x000000
	Reset - GOTO Address	0x000002
	Oscillator Fail Trap	0x000004
	Stack Error Trap	
	Address Error Trap	
	Arithmetic Error Trap	
	Software Trap	
	Reserved	
	Reserved	
	Reserved	
	Interrupt Vector 0	
	Interrupt Vector 1	
	Interrupt Vector 2	
	•	
	•	
	Interrupt Vector 53	0x00007E
Alternate Interrupt Vector Table	Reserved	0x000080
	Reserved	0x000082
	Oscillator Fail Trap	0x000084
	•	
	•	
	•	
	•	
	•	
	Interrupt Vector 53	0x0000FE

Decreasing Natural Order Priority

MICROCHIP

__DefaultInterrupt in GLD

```
ivt            : ORIGIN = 0x04,      LENGTH = (62 * 2)
/*
** Primary Interrupt Vector Table
*/
.ivt __IVT_BASE :
  {
    LONG(DEFINED(__ReservedTrap0) ? ABSOLUTE(__ReservedTrap0) : ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__OscillatorFail)? ABSOLUTE(__OscillatorFail): ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__AddressError)  ? ABSOLUTE(__AddressError)  : ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__StackError)    ? ABSOLUTE(__StackError     : ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__MathError)     ? ABSOLUTE(__MathError)     : ABSOLUTE(__DefaultInterrupt));
  :
    LONG(DEFINED(__INT0Interrupt) ? ABSOLUTE(__INT0Interrupt) : ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__IC1Interrupt)  ? ABSOLUTE(__IC1Interrupt)  : ABSOLUTE(__DefaultInterrupt));
  :
```

```
LONG(DEFINED(__ReservedTrap0) ? ABSOLUTE(__ReservedTrap0) : ABSOLUTE(__DefaultInterrupt));
LONG(DEFINED(__OscillatorFail)? ABSOLUTE(__OscillatorFail): ABSOLUTE(__DefaultInterrupt));
```

Program Memory

Line	Address	Opcode	Label	Disassembly
1090	00882	000000		nop
1091	00884	FE0000	_DefaultInterrupt	reset
1092	00886	FFFFFF		nopr
1093	00888	FFFFFF		nopr

Opcode Hex | Machine | Symbolic | PSV Mixed | PSV Data

Interrupt Priority Control

- 8 user assigned priority levels
- IPL bits (SRL<7:5) indicate current CPU priority
- Source has to greater than CPU level can interrupt
- Interrupts can be masked by writing to the IPL bits
- CPU updates IPL bits during interrupt processing
- Old IPL value saved on stack during an interrupt
- Natural priority resolves conflicts
- User assigned priority can override natural priority

Natural Order Priority

Highest Priority

INT0 - External Int 0	IRQ 0, Vector 8
IC1 - Input Capture 1	IRQ 1, Vector 9
OC1 - Output Compare 1	IRQ 2, Vector 10
TMR1 - Timer 1	IRQ 3, Vector 11
IC2 - Input Capture 2	IRQ 4, Vector 12
OC2 - Output Compare 2	IRQ 5, Vector 13
TMR2 - Timer 2	IRQ 6, Vector 14
TMR3 - Timer 3	IRQ 7, Vector 15
⋮	⋮
CAN2 - Combined IRQ	IRQ 38, Vector 46
PWM - Period Match	IRQ 39, Vector 47
QEI - Counter Compare	IRQ 40, Vector 48
DCI - Transfer Done	IRQ 41, Vector 49
PLVD - Low Volt. Detect	IRQ 42, Vector 50
FLTA - MPWM Fault A	IRQ 43, Vector 51
FLTB - MPWM Fault B	IRQ 44, Vector 52
Reserved	IRQ 45-53, Vector 53-61

Lowest Priority

Control the Interrupt

- CPU has some priority level at all times
  - IPL<2:0> in Status Register - SR
  - IPL3 in Core Control Register – CORCON
  - After RESET IPL will be cleared
- Higher priority interrupt can interrupt CPU
  - CPU then assumes new priority
  - Saves old priority on the stack

Interrupt Priority Setting

- Interrupt Priority Level set by IPCx registers
  - Interrupt sources can be levels 0 - 7
  - A level 0 source is effectively disabled
  - Default interrupt priority is level 4
  - Each Peripheral has individual priority control
- Can change CPU priority by writing IPL<2:0>
  - IPL3 is read only - can't disable traps
  - IPL<2:0> = 111 disables all other interrupts
- DISI instruction disables level 1 - 6 interrupts

Traps for Robust Operation

- Non-maskable interrupt sources
  - Level 8 ~ Level 15
- Detect catastrophic hardware / software problems
- Adhere to natural priority in IVT
  - Level 15 – the highest priority
- Trap ISR Occupied
  - IPL3 set by H/W only can be cleared by S/W
  - User handle the trap
  - _ _DefaultInterrupt defined in the GLD file
    - Content is RESET instruction

Traps for Robust Operation

- Oscillator Failure Trap (switches to FRC) (Level 14)
- Stack Error Trap (Level 13)
- Address Error Trap (Level 12)
  - Instruction fetch from illegal program space
  - Data fetch from unimplemented data space
  - Unaligned word access from data space
- Arithmetic Error Trap (Level 11)
  - Divide by Zero
  - Unsaturated Accumulator Overflow (A or B)
  - Catastrophic Accumulator Overflow (either)

Interrupt Nesting

- Interrupt Nesting
  - Any ISR that is in progress may be another source of interrupt which higher user assigned priority level.
- Traps, by default, NOT nested
- Interrupts, by default, are nestable
- Clear the NSTDIS bit (INTCON1<15>) to enable the nesting function
- IPL<2:0> bits can be modified in the ISR to raise or lower the priority of the current task

Interrupt Disable

- DISI instruction disables level 1 - 6 interrupts
- DISI can’t disable the Level 7 & Trap event
- Independent of Interrupt Enable bit setting
- lit14 + 1 cycles --- up to 16384 cycles
- DISICNT register holds disable count value
- Interrupts re-enabled when DISICNT counts to 0
- DISICNT can be written to extend DISI time
- DISICNT can be cleared to cancel instruction
- DISI status bit (INTCON2<14>)

SLEEP and IDLE

- Device will wake from SLEEP or IDLE with any enabled interrupt source
  - Set the relate bit in the IECx (Interrupt Enable Control Register)
- Wakes from SLEEP or IDLE mode if:
  - Interrupt is greater than CPU priority level, then the execution will be branch into ISR
  - Interrupt equal or less then CPU priority level, then the execution will follow next instruction immediately.

REPEAT Loops

- The pre-fetched within REPEAT loop will be disable
  - RCOUNT>1, disable the instruction pre-fetched
- REPEAT can be interrupted and nested
- REPEAT loop will set the RA bit (SRL<4>) when it is in progress
- RA bit push to stack with SRL register (Interrupt)
- RA bit cleared automatically on interrupt entry
- User can clear RCOUNT to quit the loop
- User must stack RCOUNT to nest REPEAT loops

DO Loops

- DO can be interrupted
- Hardware provides 1 levels of DO context saving
  - DOSTART, DOEND, DCOUNT (Saved in Shadow Reg.)
- If another DO loop is to be executed in the ISR, user must check the DL<2:0> status and save DO registers
- DL<2:0>≠'0' & DCOUNT > 1，DA bit will be set
- Must consider number of nested DO levels for interrupt processing
- Setting the EDT bit terminates DO loop execution (drops the DO level by one)
  - DL bits modified by HW to reflect new DO level

Interrupt Registers

SRL

R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
IPL<2:0>			RA	N	OV	SZ	C
bit7	6	5	4	3	2	1	bit0

INTCON1

R/W-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
NSTDIS	-	-	-	-	OVATE	OVBTE	COVTE
bit15	14	13	12	11	10	9	bit8

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
-	-	-	SWTRAP	OVRFLOW	ADDRERR	STKERR	-
bit7	6	5	4	3	2	1	bit0

INTCON2

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

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	-	INT4EP	INT3EP	INT2EP	INT1EP	INT0EP
bit7	6	5	4	3	2	1	bit0

Interrupt Registers

IEC0 (One of four Interrupt Enable Registers)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNIE	BCLIE	I2CIE	NVMIE	ADIE	U1TXIE	U1RXIE	SPI1IE
bit15	14	13	12	11	10	9	bit8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T3IE	T2IE	OC2IE	IC2IE	T1IE	OC1IE	IC1IE	INT0IE
bit7	6	5	4	3	2	1	bit0

IFS0 (one of four Interrupt Flag Status Registers)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNIF	BCLIF	I2CIF	NVMIF	ADIF	U1TXIF	U1RXIF	SPI1IF
bit15	14	13	12	11	10	9	bit8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T3IF	T2IF	OC2IF	IC2IF	T1IF	OC1IF	IC1IF	INT0IF
bit7	6	5	4	3	2	1	bit0

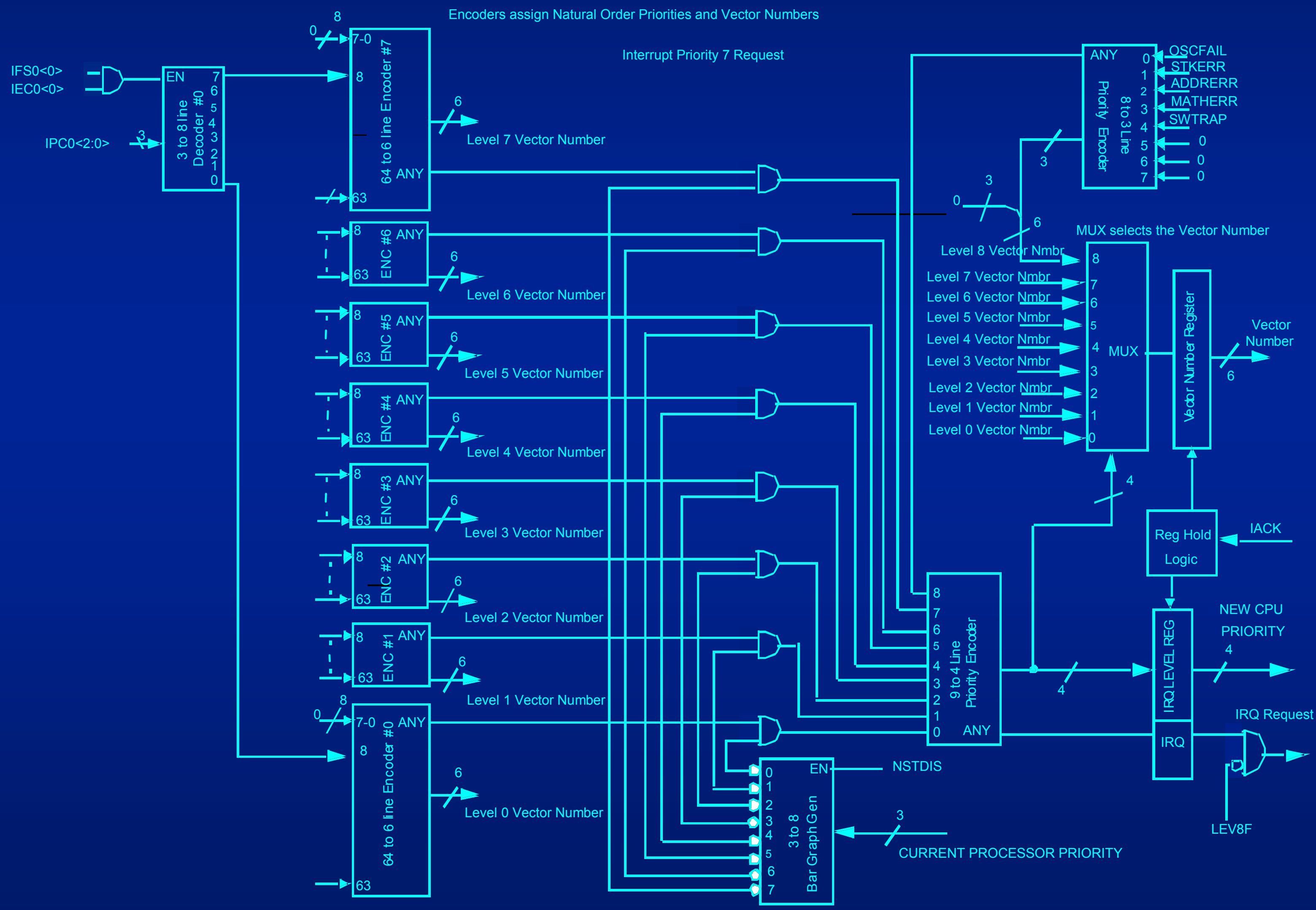
Interrupt Registers

IPC0 (One of twelve Interrupt Priority Control Registers)

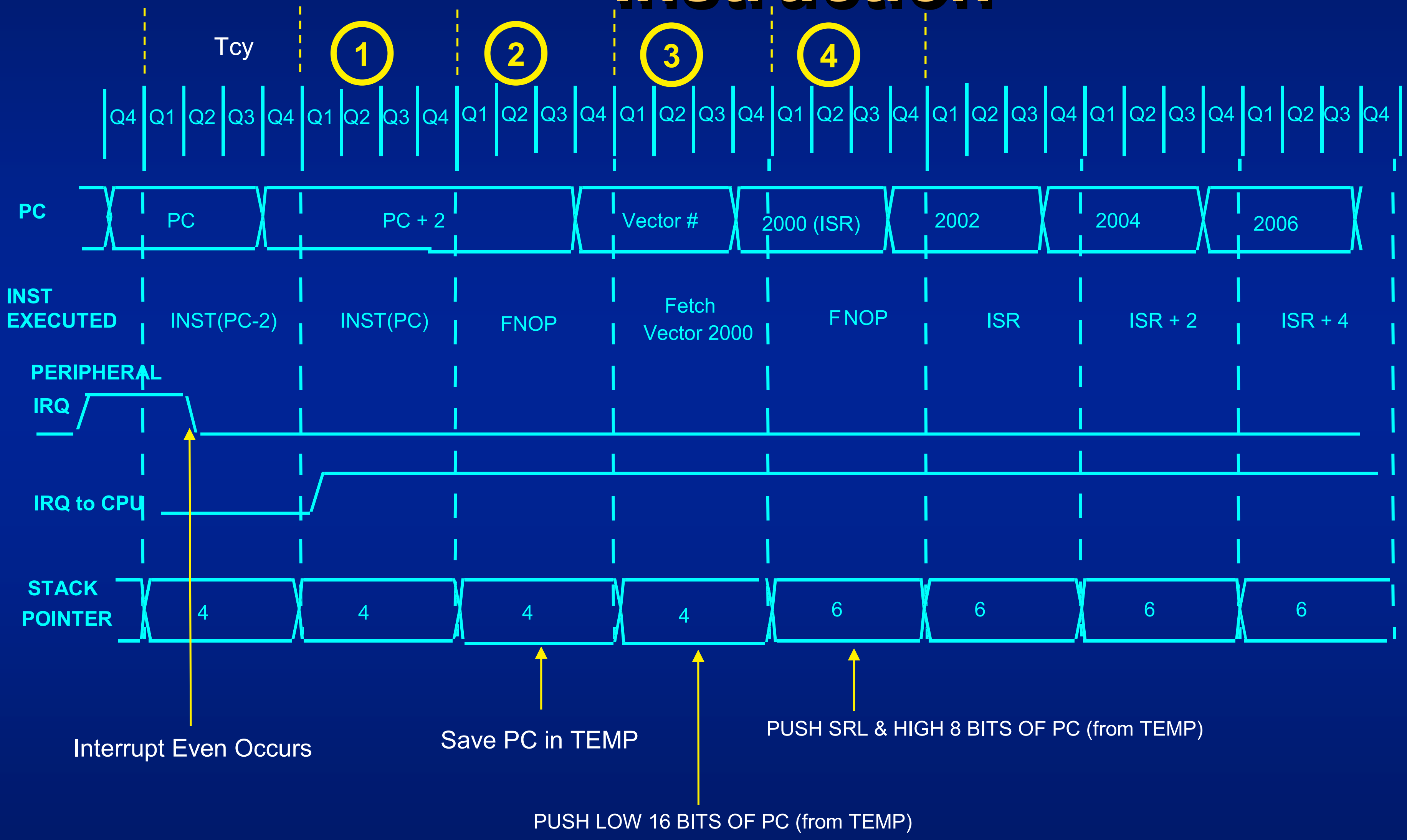
U-0	R/W-1	R/W-0	R/W-0	U-0	R/W-1	R/W-0	R/W-0
-	T1IP<2:0>			-	OC1IP<2:0>		
bit15	14	13	12	11	10	9	bit8
U-0	R/W-1	R/W-0	R/W-0	U-0	R/W-1	R/W-0	R/W-0
-	IC1IP<2:0>			-	INT0IP<2:0>		
bit7	6	5	4	3	2	1	bit0

Related IPCx bit default setting are 4

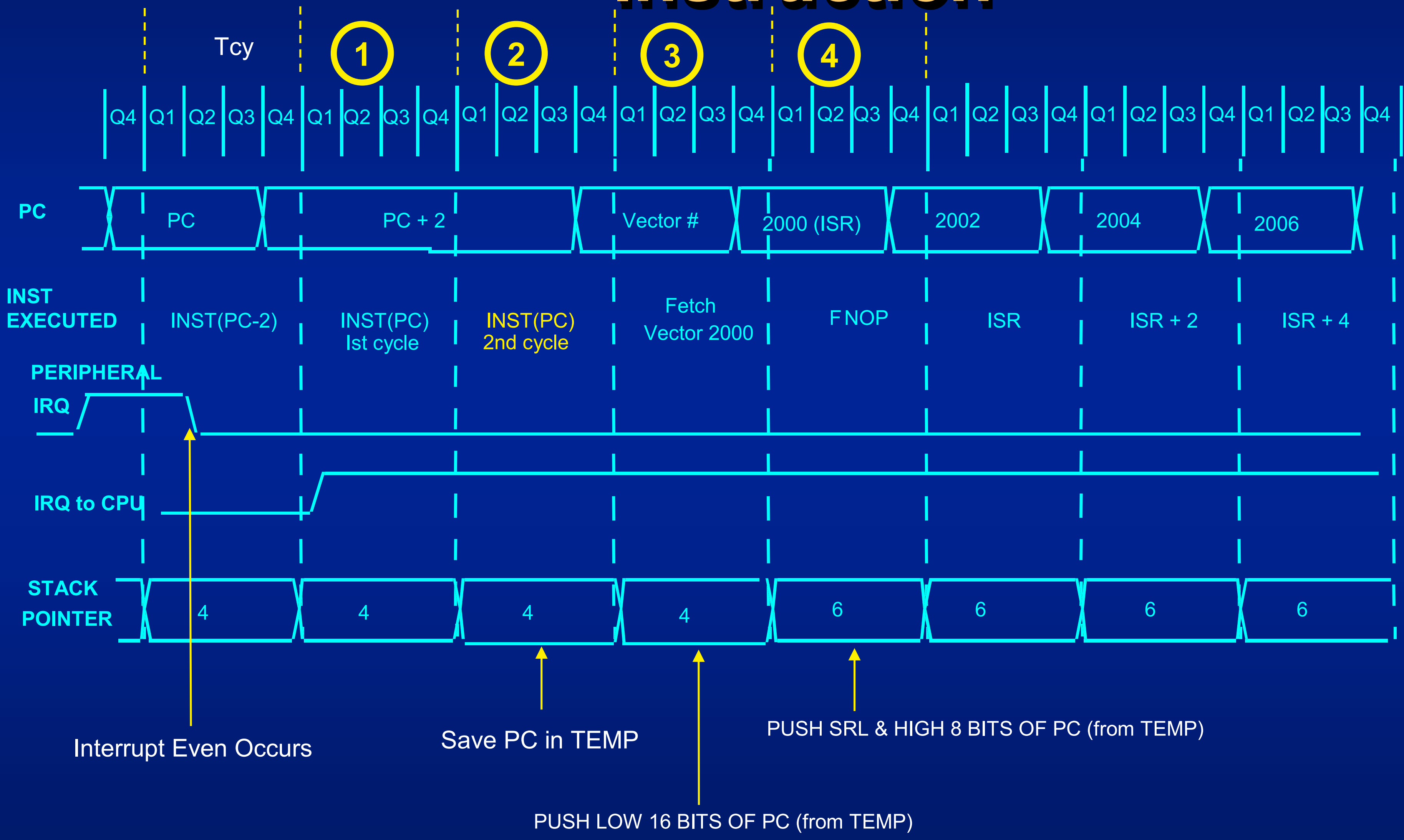
Interrupt Controller Block Diagram



Interrupt Entry: 1 Cycle Instruction

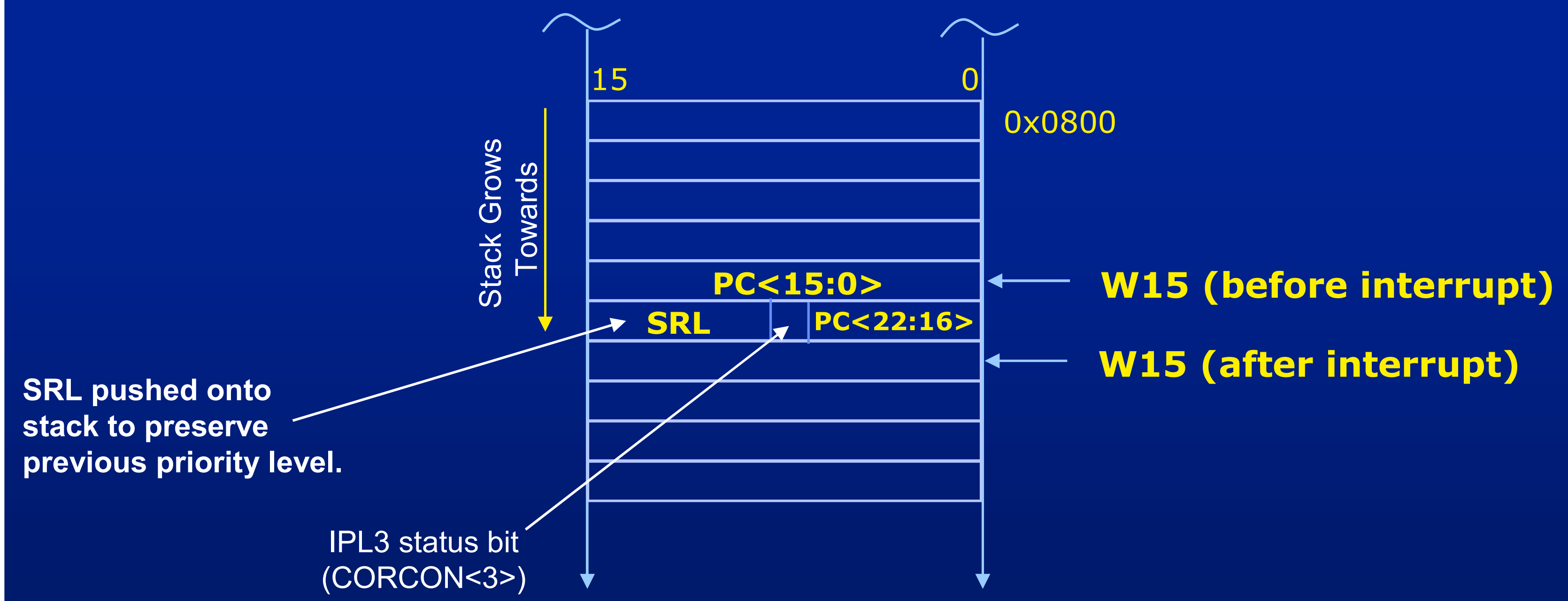


Interrupt Entry: 2 Cycle Instruction



Stack Operation

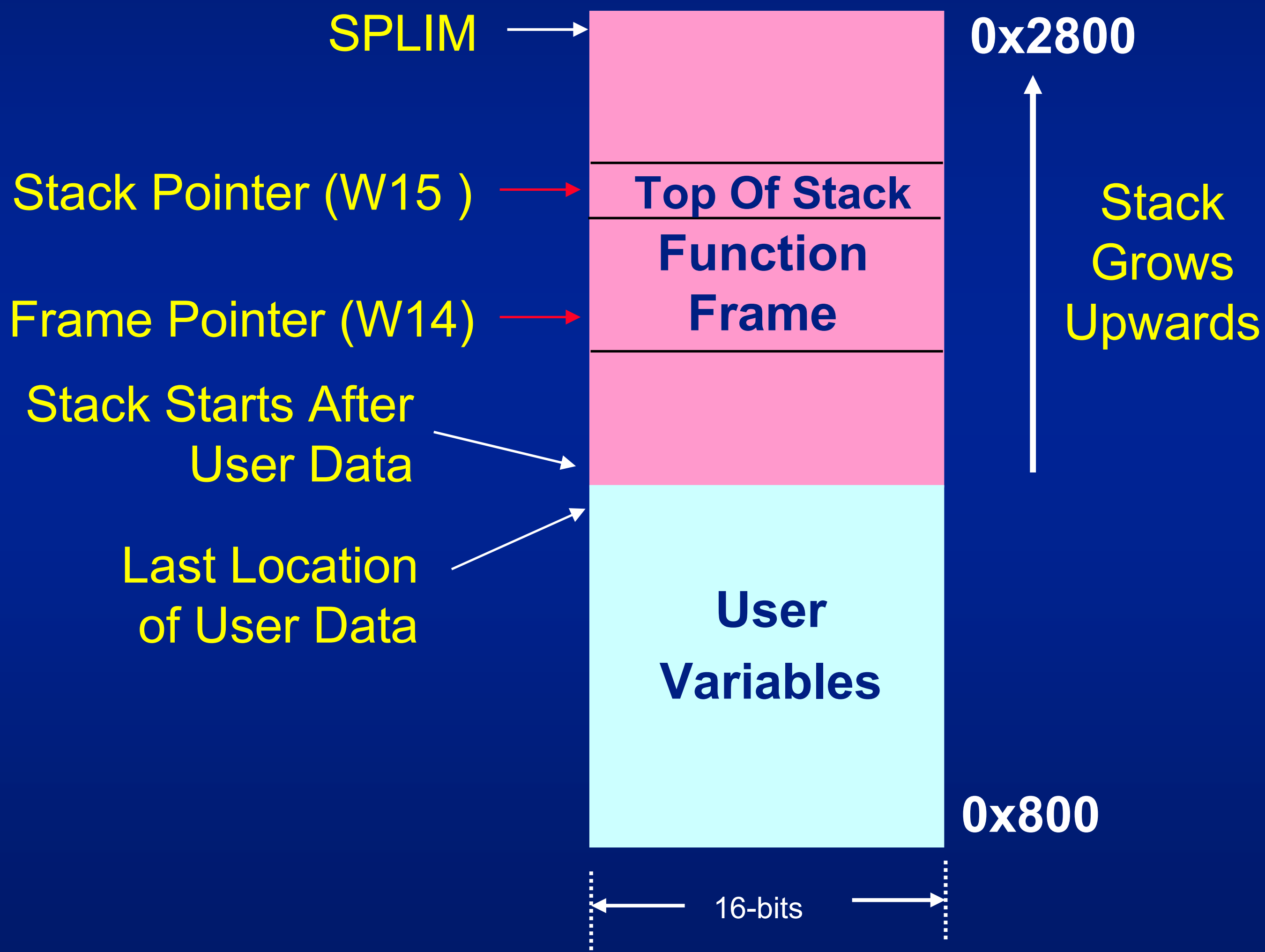
- Interrupt Context Saving
  - Push IPL3 , SRL and PC into the Stack
  - Compiler saves additional context data
  - User can specify other data to save



Interrupt Context Save/Restore

- Only SRL , IPL3 and PC is saved automatically
- Use PUSH(.D) and POP(.D) to save/restore in Stack
- PUSH.S and POP.S allow fast context save in Shadow
  - W0…W3
  - SR (N,OV,Z,C,DC bit only)
- Be careful when using W shadows for different priority tasks...
  - The PUSH.S instruction will overwrite the contents previously saved in the shadow.
  - The Shadow are only one level in depth.
  - User save the low priority task when higher priority task is happened

Software Stack in Data RAM



Software Stack

- The stack is used for…
  - Function Calls/Interrupts/Traps
  - Context Saving
  - Passing arguments to functions
  - Local variables
- Stack size limited by the amount of free RAM
  - Linker places stack after user's data
- For protection, a Stack Trap occurs when…
  - Stack Pointer > SPLIM register (overflow)
  - Stack Pointer < 0x800 (underflow)

"C" Run-Time Startup

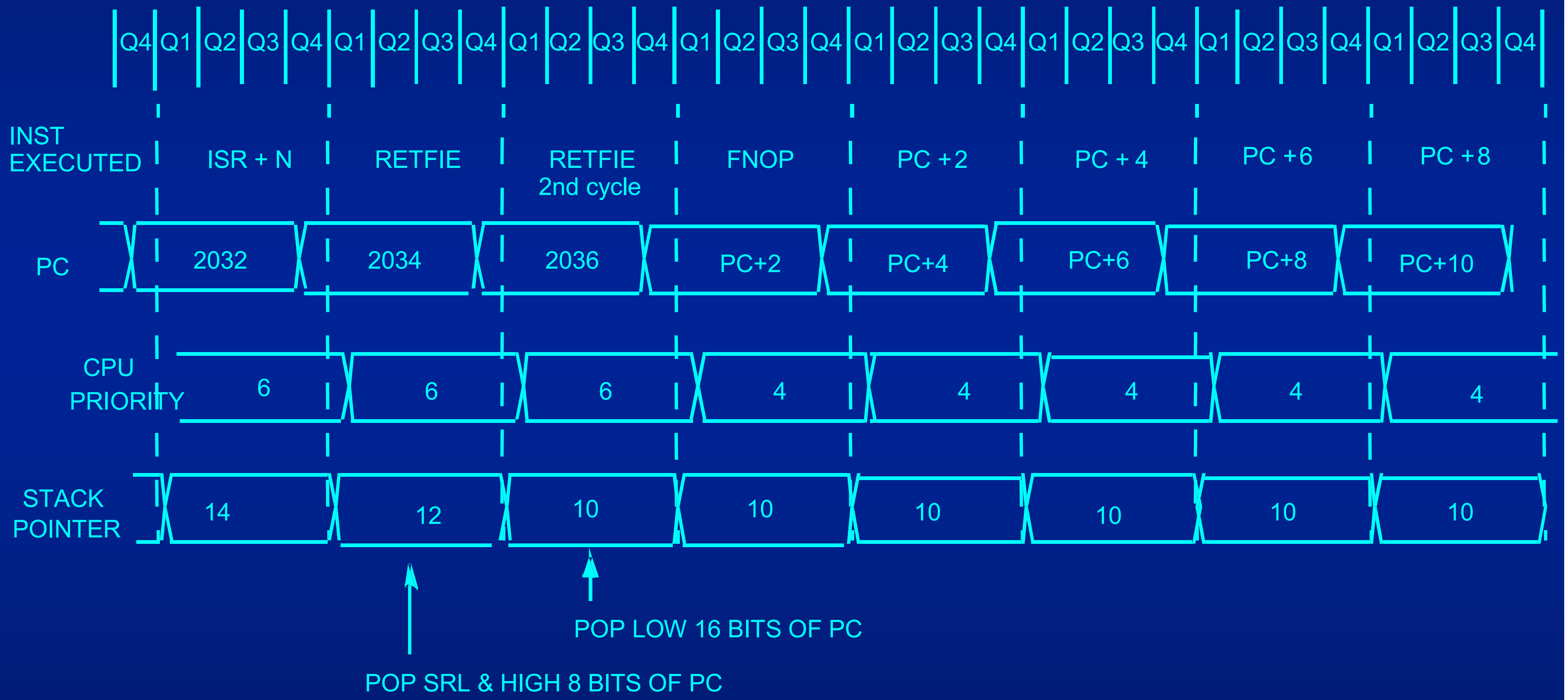
Program Memory

Address	Label	Disassembly
00100	_reset	mov.w #0x8fc,w15
00102		mov.w #0x2796,w0
00104		mov.w w0,SPLIM
00106		nop
00108		rcall _psv_init
0010A		rcall _data_init
0010C		call main
0010E		nop

Opcode Hex | Machine | Symbolic | PSV Mixed | PSV Data

Stack Setting

Return from Interrupt Timing



中斷的使用

Using MPLAB C30

定義中斷函數
使用C30

- 中斷函數 (ISR) 不可帶有任何的參數傳遞
- 中斷函數 (ISR) 也不可以有回傳值
- 中斷函數 (ISR) 必須考慮到背景資料的儲存/取回
- C30 支援 ...
  - 快速的背景存取 (使用 PUSH.S / POP.S 指令)
  - 詳盡的變數存取
  - 中斷向量位址的設定

範例:

```
void _ISR _INT0Interrupt (void);
void _ISR _TIMER2Interrupt (void);
void _ISR _AltINT1Interrupt (void);
```

C30 的中斷函數宣告

- `void __attribute__((interrupt,(options))) fname(void)`
  - `shadow` 選項提供快速的背景存取
  - `save` 選項提供一般變數的背景儲存 / 取回
  - `no_auto_psv` 選項：不處理 PSVPAG 暫存器的設定
- 基本 ISR 函數的宣告
  - `void __attribute__((interrupt))_INT0Interrupt (void)`
- 使用快速背景存取的 ISR 函數宣告
  - `void __attribute__((interrupt,shadow)) _INT0Interrupt (void)`
- 使用一般變數背景存取的 ISR 函數宣告
  - `void __attribute__((__interrupt__(__save__ (var1,var2))) _INT0Interrupt (void)`

使用 C30 撰寫中斷函數

- 已經在 h 檔裡定義了中斷函數的屬性
  - #define _ISR __attribute__((interrupt))
  - #define _ISRFAST __attribute__((interrupt, shadow))
    - Compiler saves/restores the ISR context
- “shadow” 可於一個指令週期快速存取 W0-W3
  - 使用 PUSH.S/POP.S 指令 (注意只支援一層深度)
  - `void _ISRFAST _SPI1Interrupt (void);`
- 在中斷使用到的Global 變數需加上 volatile 的宣告
  - `volatile unsigned global_flag;`
- ISR 離開前必須清除相關的中斷旗號
  - `IFS0bits.T1IF = 0;`
- 不要在中斷裡再呼叫其他的函數

Interrupt Service Routines in Assembly Language

- To be placed in the Vector Table by the Linker, the function name must use 2 underscores with the Vector Name
  - `__TMR2Interrupt` for Timer2 ISR
  - `__OC3Interrupt` for OutputCompare3 ISR
- Prologue code must store all utilized Working Registers and pertinent SFRs (TBLPAG, etc…) to the stack before ISR processing
  - Use PUSH.D to efficiently save Working Registers
- Epilogue code must restore all registers stored in the Prologue code after ISR processing
  - Use POP.D to efficiently restore Working Registers
- ISRs must return using RETFIE

Assembly Language Example

```
; ISR function __INT0Interrupt
; Purpose:  Increment global counter "Count" by 1
; Used registers W0, W1
__INT0Interrupt:
        ; save W0 and W1 to stack
        PUSH.D          W0
        ; increment Count variable
        MOV             Count, W0
        INC             W0, W1
        MOV             W1, Count
        ; restore W0 and W1
        POP.D           W0
        ; clear the interrupt flag
        BCLR.B         IFS0,#INT0IF
        ; return from interrupt
        RETFIE
```

中斷的禁忌

- 一般在處理中斷函數時，底下所列的指令不要使用中斷程式及巢狀式中斷處理
  - DO 指令，建議不要超過三層的巢狀處理 (nesting)
    - 2 層巢狀處理是可以的 (第一層 DO 在主程式處理，另一個 DO 在中斷函數處理是可以的)
    - 超過兩層以上的 DO 指令需要自行儲存 DO 指令相關德暫存器
  - PUSH.S / POP.S
    - Shadow 暫存器只支援一層
  - REPEAT
    - RCOUNT 暫存器必須手動儲存
- 中斷旗號如沒有被清除，該中斷將持續發生

中斷支援

- 使用標準函數名稱
  - 所有中斷定義的函數名稱都使用單一的 ”_”
    `_T2Interrupt`
    `_INT1Interrupt`
  - 中斷使用參考 “MPLAB® C30 User’s Guide”
  - 中斷函數名稱定義參考 C:\Program Files\Microchip\MPLAB C30\docs
    - hlpMPLABC30.chm
  - 使用錯誤的中斷函數名稱，C30 會出現警告訊息
  - C30 會自動安排 ISR 中斷向量位址

中斷範例

- Use the interrupt attribute

```
#included <p30f4011.h>
:
void __attribute__((interrupt, no_auto_psv)) _T2Interrupt(void)
{
    :
    Processing your code here
    :
    :
    // Clear timer2 interrupt
    IFS0bits.T2IF = 0;
 }
```

改變中斷優先權方式

- Disabling Level 6 Interrupts

```
/* 先記住 CPU 的優先權等級 */
unsigned int ipl = SRbits.IPL;

 SRbits.IPL = 6; /* 變更 CPU 優先權等級到 6 */
 /*
 ** Protected code here
 */

/* 取回原先 CPU 的優先權等級 */
 SRbits.IPL = ipl;
```

在一段時間內關閉中斷功能

- 暫時關閉等級 6 以下的中斷

```
 DISICNT = 16383;
/*
** Protected code here
*/
DISICNT = 0;
```

動手做中斷練習

基本中斷設定 (LAB1)

&

中斷優先權控制 (LAB 2)

APP020 Plus 實驗板

- CPU : dsPIC30F4011-30I/P
- Fosc : 使用 7.372800 MHz 石英晶體
- DIP SW 位置設定 (LAB1) :
  - DSW1 : On, On, Off, Off 燒錄及除錯都使用同一組腳位 (PGD & PGD)，關閉 UART1
  - DSW2 : Off, Off, On, On 選擇 UART2 作爲 RS-232 通訊
  - DSW3 : On, On, Off, Off 選擇開啓 VR1 & VR2 的電壓輸入
  - DSW4 : On, On, Off, Off 連接 QEI 訊號產生器 (因 QEI Index 與按鍵 SW5 共用 RB3，所以要將其 Off)
  - DSW5 : On, On, Off, Off 選擇 RF0 & RF1 爲 LCD 驅動腳位
- JEN1 : Closed for LED driver
- JEN2 : Closed for Pull-Up resister

中斷實驗 Lab 1

- 使用四組Timer 的中斷設定 LED1, LED2, LED3 及 LED4 閃爍時間
    - Timer1 for LED1 (_T1Interrupt), 2 Sec.
    - Timer2 for LED2 (_T2Interrupt), 1 Sec.
    - Timer3 for LED3 (_T3Interrupt), 500mS
    - Timer4 for LED4 (_T4Interrupt), 100mS
- LEDx 在中斷裡轉態， LCD 在 main( ) 顯示
- 設定 CPU IPL<0:2> CPU 的中斷優先權爲最低的 level 0 (開啓所有的中斷)

中斷實驗 Lab 1 的提示

- Lab 1 的步驟
  1. Enable TxIE interrupts
     - IECx - Interrupt Enable Control
       - T1IE, T2IE, T3IE & T4IE bits
  2. Create 4 Timer interrupt function
     - Use labels defined in Linker Script file (GLD)
  3. Clear interrupt flag to allow a new interrupt
  4. Remind …
     - CPU interrupt priority default level 0
     - Peripheral Priority level 4

中斷優先權實驗 Lab 2

- 按鍵 SW5 & SW6 調整 CPU 的優先權
  - 初上電時，內定 CPU 優先權等級爲 0
  - 4 個 Timer 個別設定不同的中斷優先權
    - Timer1 is Level 7,Timer2 is Level 5
    - Timer3 is Level 3,Timer4 is Level 1
  - 按下 SW5 增加 CPU 優先權等級 + 1
  - 按下 SW6 減低 CPU 優先權等級 - 1

T1=7, T2=5, T3=3
T4=1, CPU Core=0

LCD 顯示所有的中斷優先權控制狀態

Q & A

- How do I nest interrupts?
- What are the options for handling a non-maskable trap event?
- What should I do with unused vector locations?
- A GOTO instruction is required at the RESET vector. Why?
- How would I mask all interrupts below a given priority level?
- Two level 3 interrupt sources occur simultaneously. Which source interrupts the CPU first?
- Can one execute a TRAP instruction from an ISR?

Thank You