MICROCHIP

Regional Training Centers

Section 6
Interrupt

MICROCHIP
Regional Training Centers

What's Interrupt ?

- 在一般的情況下,CPU是按照程式的流程依序地執行。但如果此時某些周邊希望取得CPU的服務,此時就可以透過中斷,打斷目前正在執行的程式片斷,跳到該中斷的服務常式中(ISR, Interrupt Service Routine) 。
- 中斷的來源,有許多可能如Timer, ADC, UART, SPI, etc..,都可以打斷CPU。這些中斷來源在16-Bits MCU中被區分為兩大類:
  Interrupt:一般周邊產生的中斷來源,如上述的Timer, ADC, UART, SPI, 外部中斷, etc..。
  Trap:俗稱的不可遮罩中斷(NMI), 這類型的中斷發生,代表著某些可能危害程式運行的狀況出現,如除"0",Stack Overflow / Underflow等。

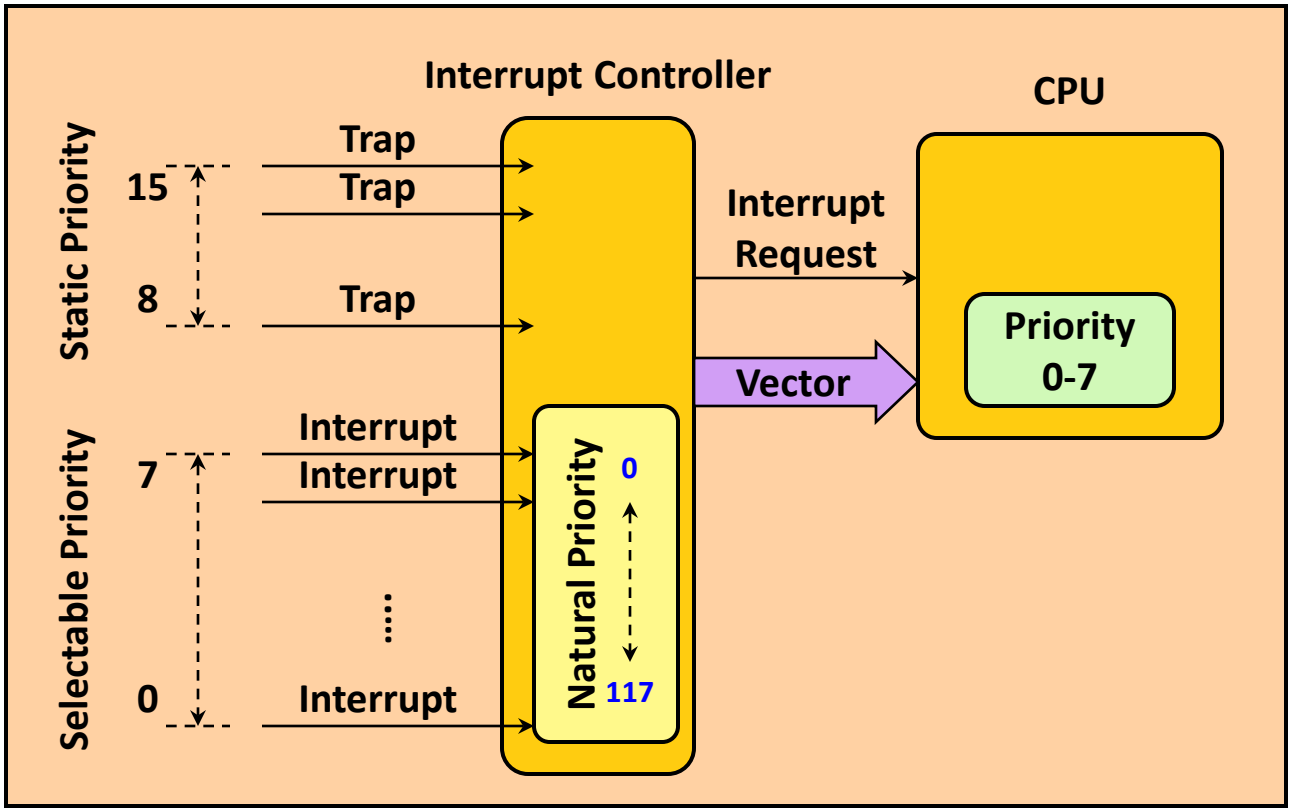
MICROCHIP
Regional Training Centers

What's Interrupt ?

- 中斷條件的成立,通常必須要滿足幾個要素:
  中斷需求(Interrupt Request):在MCU中,周邊會透過設定中斷旗標(xxIF)來提出中斷需求。
  中斷致能(Interrupt Enable):使用者可以透過中斷致能位元(xxIE)來決定是否接受周邊的中斷需求。
  簡單的說,就是當xxIF跟xxIE都為1時,才能打斷CPU的執行,跳到中斷服務常式中。
- Trap的發生代表著某些可能危害程式運行的狀況出現,因此Trap是無法拒絕的,CPU一定必須接受。16-Bits dsPIC MCU定義以下幾種Trap : Stack Error, Oscillator Fail Error, Address Error, Math Error, DMA Error。

16-Bits Interrupt Architecture

- 16-Bits的中斷架構,如圖所示。16-Bits MCU支援巢狀中斷,也就是中斷可以設定優先權,優先權高的可以打斷低的。
- Interrupt的優先權可以自行設定,0~7(IPCx)。
- Trap則固定為8~15,每個Trap都有固定的優先權。
- 另外CPU本身也有優先權0-7 。



MICROCHIP
Regional Training Centers

16-Bits Interrupt Architecture

- 在16-Bits的中斷架構中,要使中斷條件成立,除了前述的xxIF跟xxIE都必須為1以外,中斷(Interrupt)的優先權還必需大於CPU的優先權才可以成立。
- 因為CPU最大的優先權可以設定到7,但Trap從8開始,這也說明了為何CPU無法拒絕Trap。
- CPU的優先權設定位元(IPL)在狀態暫存器的Bit7-Bit5(SRL <7:5>)。中設定。 bits 透過更改CPU的優先權設定位元(IPL)的值,可以抑制較低優先權的中斷發生(IPL=7, Disable All Interrupt)。當中斷發生時,舊的CPU優先權(IPL)會被自動存到堆疊裡保存起來。
- 當兩個相同優先權的中斷”同時”發生時,CPU要先服務誰?

Natural Priority

- 自然優先權(Natural Priority)是仲裁CPU該先服務那個中斷的最後手段。
- 當數個優先權相同的中斷”同時”發生時,由於自訂的優先權無法比較出高低,因此僅能透過自然優先權來仲裁。
- 在16-Bits MCU中自然優先權,是根據中斷在中斷向量表中安排的位置而定,位址越低的優先權越高,位址越高的優先權越低。

Vector Number	IVT Address	AIVT Address	Interrupt Source
0	0x000004	0x000104	Reserved
1	0x000006	0x000106	Oscillator Failure
2	0x000008	0x000108	Address Error
3	0x00000A	0x00010A	Stack Error
4	0x00000C	0x00010C	Math Error
5	0x00000E	0x00010E	DMA Error
6	0x000010	0x000110	Reserved
7	0x000012	0x000112	Reserved
8	0x000014	0x000114	INT0 – External Interrupt 0
9	0x000016	0x000116	IC1 – Input Capture 1
10	0x000018	0x000118	OC1 – Output Compare 1
11	0x00001A	0x00011A	T1 – Timer1
12	0x00001C	0x00011C	DMA0 – DMA Channel 0
13	0x00001E	0x00011E	IC2 – Input Capture 2
14	0x000020		
15	0x000022		
16	0x000024		
17	0x000026		
18	0x000028		
19	0x00002A		
20	0x00002C		
21	0x00002E		
22	0x000030		
23	0x000032		
24	0x000034		
25	0x000036		
26	0x000038		
27	0x00003A		
28	0x00003C		
29	0x00003E		

Reset – GOTO Instruction	0x000000	
Reset – GOTO Address	0x000002	
Reserved	0x000004	
Oscillator Fail Trap Vector		
Address Error Trap Vector		
Stack Error Trap Vector		
Math Error Trap Vector		
DMA Error Trap Vector		
Reserved		
Reserved		
Interrupt Vector 0	0x000014	Interrupt Vector Table (IVT)(1)
Interrupt Vector 1		
~		
~		
~		
Interrupt Vector 52	0x00007C	
Interrupt Vector 53	0x00007E	
Interrupt Vector 54	0x000080	
~		
~		
~		

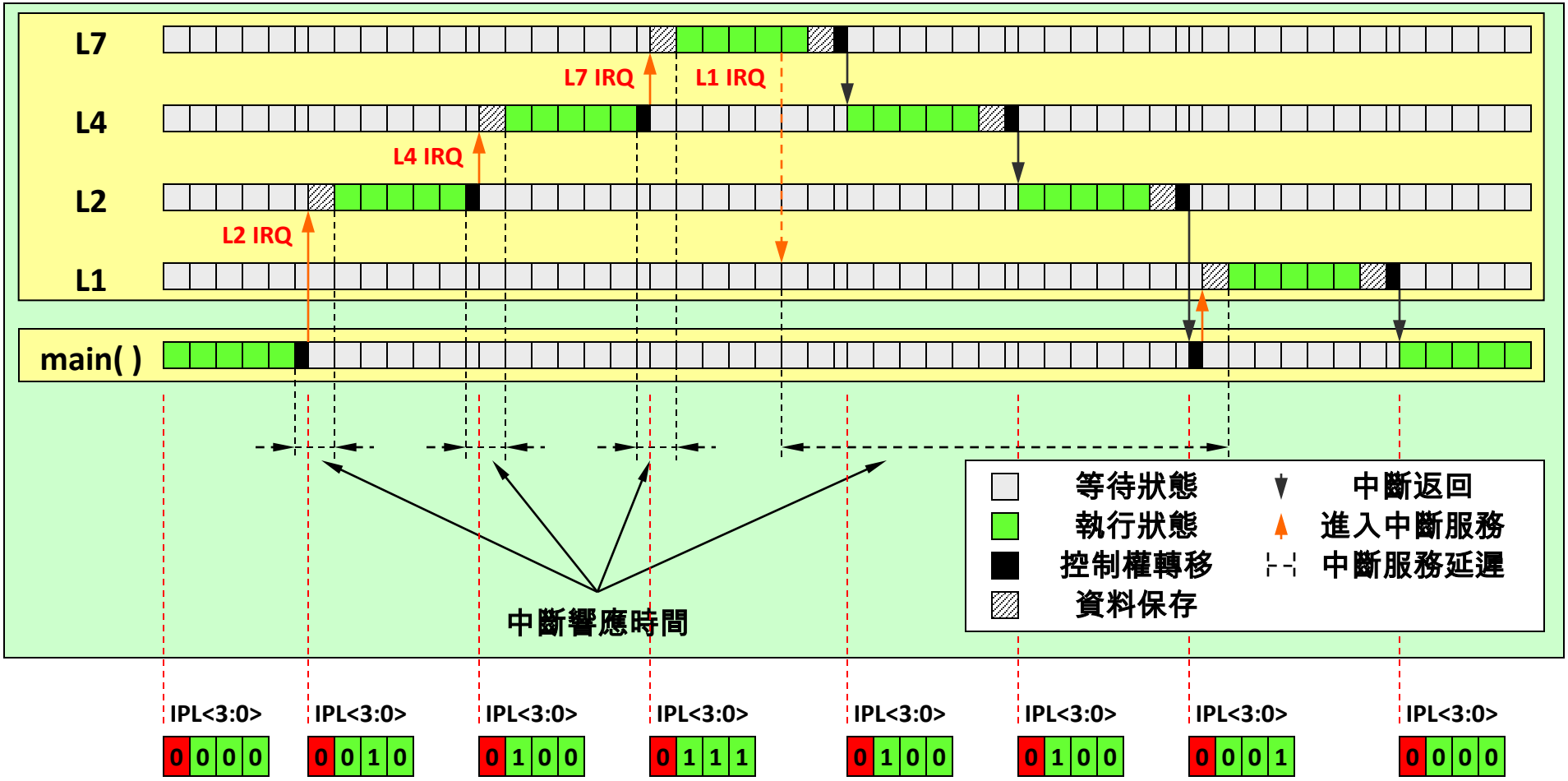
MICROCHIP
Regional Training Centers

16-Bits Interrupt Architecture

- 各個周邊可以單獨針對自己的Interrupt Request設定優先權(IPCx),最小為”0”(Disable Interrupt),最大為”7” 。
- Reset後,CPU的優先權會恢復到”0”,各個周邊的優先權則會被設定為”4” 。
- 組合語言中,提供DISI指令,用來優先權1 - 6的中斷n+1的指令週期(Tcy) 。
  Ex:DISI 16384 ;暫停優先權1 - 6的中斷16,385個指令週期。

- 實際來看一下中斷發生時,的運作流程!

Interrupt Flow



MICROCHIP
Regional Training Centers

中斷架構的程式架構

- 一個完整的中斷架構程式片斷,必須至少有三個部份:
- 中斷服務常式(Interrupt Service Routine):
  中斷需求被接受後,CPU會跳至ISR中執行程式,因此必須針對所啟用的中斷設計ISR。例如:Timer計數時,要Toggle LED。
  在MPLAB C30中,透過 __attribute__ ((interrupt, auto_psv)) 來指定ISR。
- 設定中斷優先權, 開啟中斷:
  如開頭所說,中斷必須要被致能,周邊發出需求時,才會被CPU接受,因此必須在啟用中斷時,將對應的(xxIE)設為"1" 。
  在MPLAB C30中,透過ConfigIntxxxx( )Function來設定。
- 初始化周邊模組,設定發出中斷需求的模式:
  設定周邊模組要在什麼情形下,發出中斷需求。例如:ADC每次轉換完成後,發出中斷需求。
  在MPLAB C30中,透過Openxxxx( )Function來設定。

MICROCHIP
Regional Training Centers

MPLAB C30對Interrupt的支援

- MPLAB C30透過attribute來指定ISR。

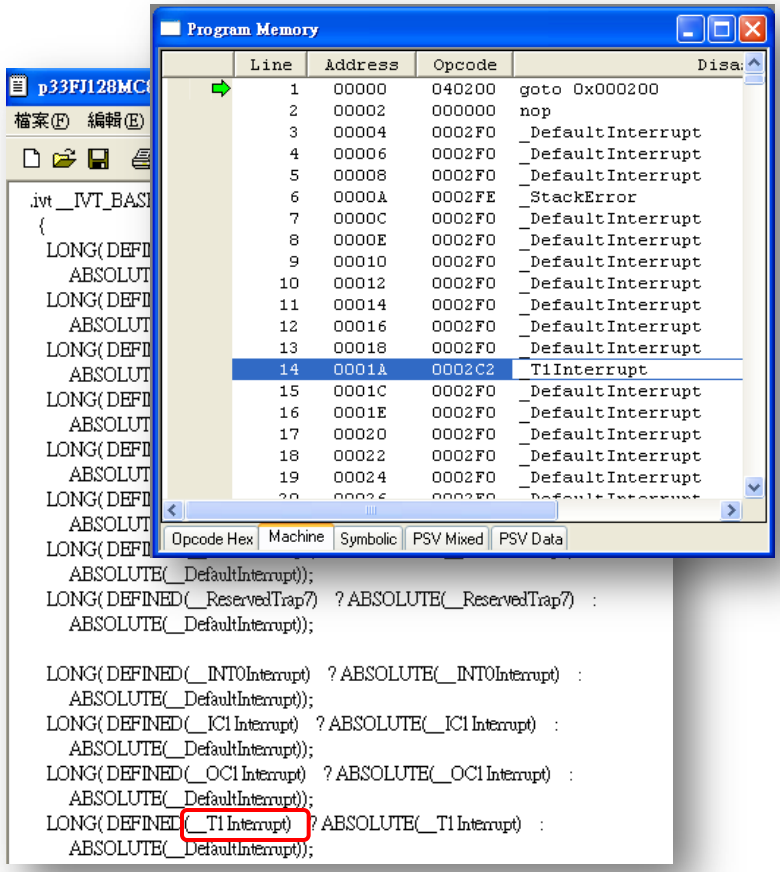
```
Ex: void __attribute__(( interrupt , auto_psv )) _T1Interrupt( void )
{
   IFS0bits.T1IF = 0;
   ....
}
```

- 宣告一個函式,作為Timer1的中斷服務函式。注意,中斷服務函式不可以有傳入跟傳回值,所以都必須設定為void型態。
- 中斷服務函式,一定要把對應的中斷旗標清除為零。在中斷的架構上,中斷旗標會由硬體設為"1",但不會自動清除,必須透過軟體手動清除。

- 為何知道是Timer1的中斷服務函式？

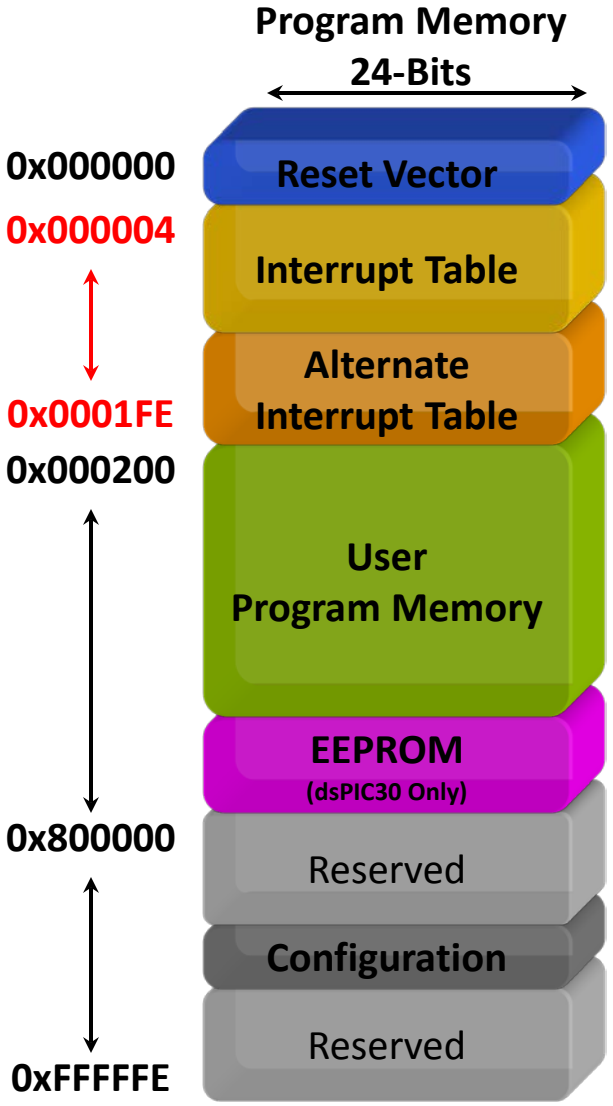
MPLAB C30對Interrupt的支援

- MPLAB C30中規定了所有中斷服務函式的名稱,每個中斷都有專屬的名稱。宣告ISR時,名稱必須一模一樣。
- 中斷服務函式的名稱可以開啟對應MCU的連結檔(Linker Script, *.gld)來尋找。
- 如果在程式中宣告的 ISR名稱無法與連結檔中相同時,組譯時Linker會出現Warning的提示。
- 如果宣告成功,Linker會自動將相關ISR的起始位寫入該中斷在中斷向量表的位址中。可以透過MPLAB IDE功能表View\Program Memory看到。



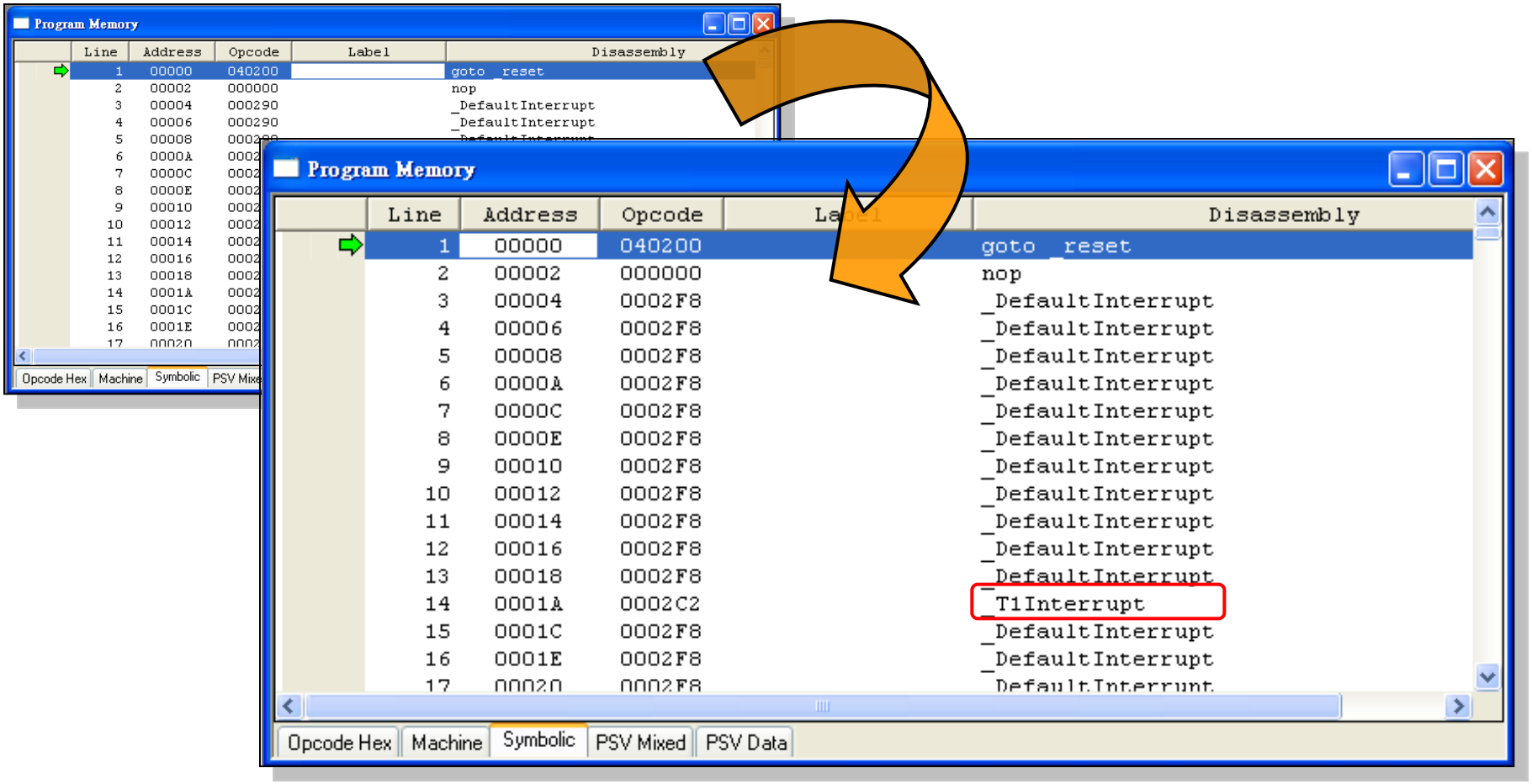
Interrupt Vector Table

- 16-Bits MCU將中斷向量表(IVT),設定在程式記憶體中0x000004-0x0001FE的區域。當中斷服務常式宣告成功後,Linker就會將中斷服務常式的進入點,填入中斷向量表中。
- 中斷發生時,如果中斷被致能,CPU就會先到中斷向量表(IVT)中取出,對應中斷的中斷服務常式的進入點,接著跳躍到ISR中。



Linker對IVT的安排

- void __*attribute__((interrupt , auto_psv))* _*T1Interrupt*(void)



MICROCHIP
Regional Training Centers

Attribute的可用宣告

- Attribute針對中斷有幾種不同形式的宣告方式:
- 宣告ISR使用Compiler managed PSV

  Ex: void __*attribute*__((*interrupt , auto_psv*)) _T1Interrupt(void)

  宣告*auto_psv*時,進入中斷時會將所用到的PSVPAG推入堆疊,保存起來。但會增加Context Saving的時間。

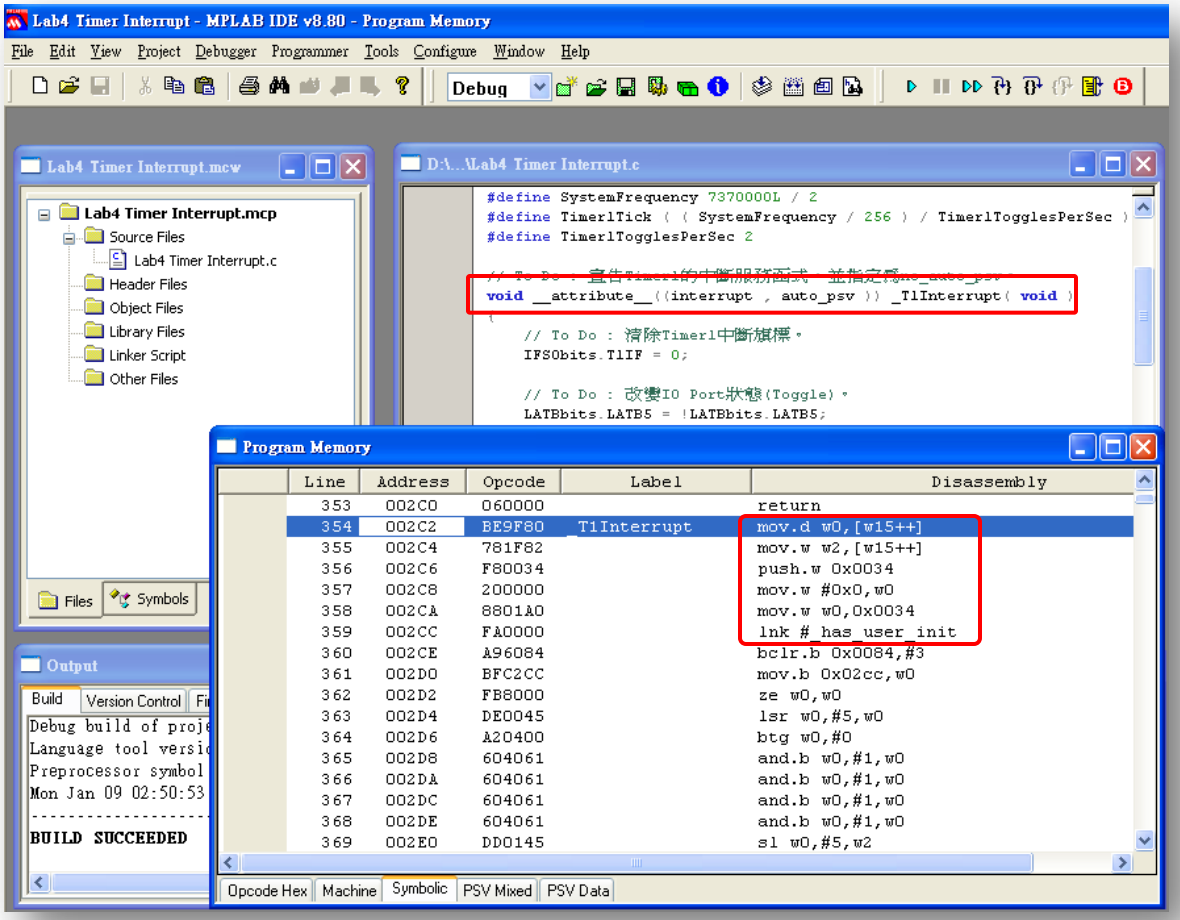
  *有在ISR中使用到const資料時就要宣告。
- 宣告ISR不會使用到Compiler managed PSV

  Ex: void __*attribute*__((*interrupt , no_auto_psv*)) _T1Interrupt(void)
- MPLAB C30 的預設值是*auto_psv*。
- 宣告要額外請Compiler幫忙保存的資料

  void __*attribute*__((*interrupt*(*save*(*var1 , var2*)))) _T1Interrupt(void)

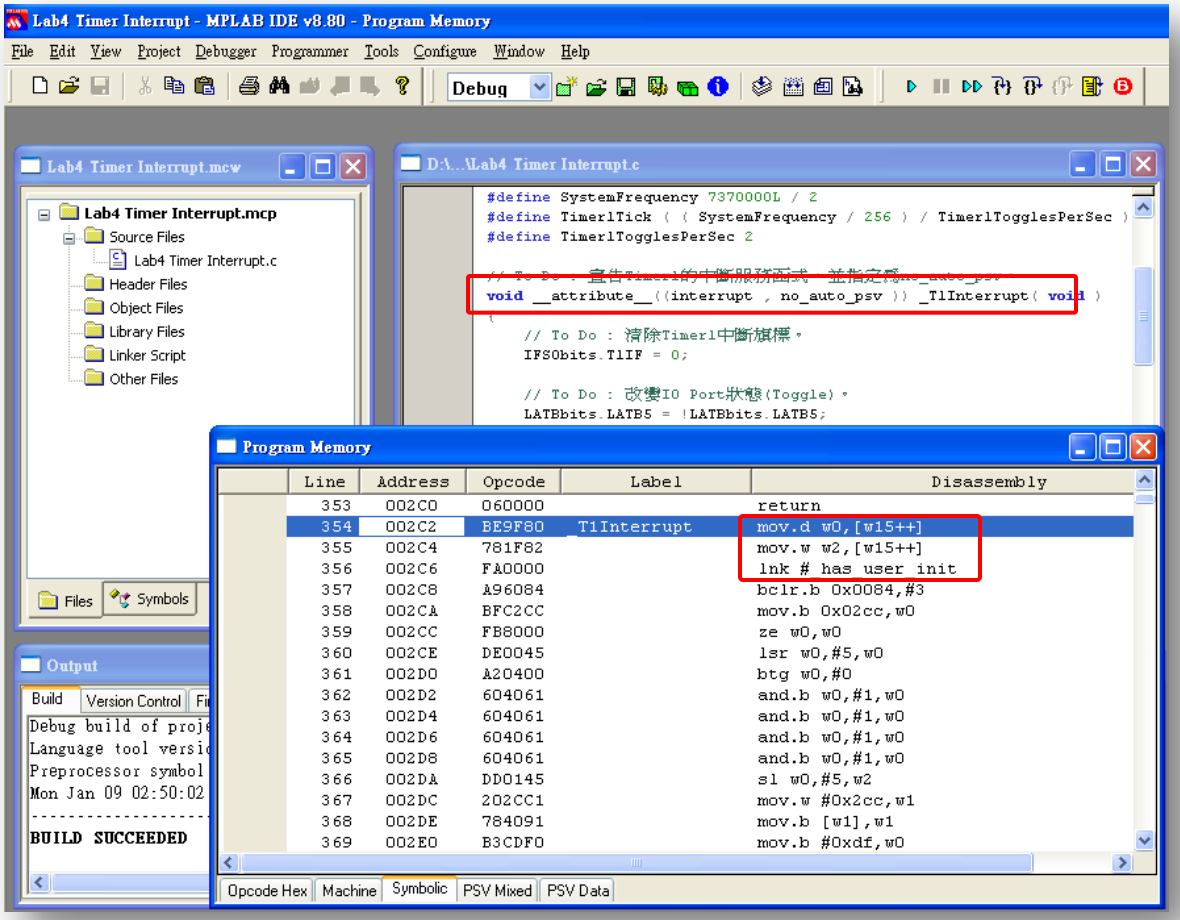
使用*auto_psv*的ISR

- 宣告*auto_psv*時,進入中斷時會將所用到的PSVPAG推入堆疊,保存起來。所以ISR會多出一段程式達成上述功能。
- 除了程式變大以外,Context Saving跟中斷響應時間都會變長。
- 記住!在ISR中使用到const 型態的時就要宣告*auto_psv* 。



使用*no_auto_psv*的ISR

- 宣告*no_auto_psv*時,進入中斷時就不會保存PSVPAG。跟所以*auto_psv* 的宣告相比除了程式較小以外,Context Saving跟中斷響應時間都相對的較短。



MICROCHIP
Regional Training Centers

MPLAB C30對Interrupt的支援

- MPLAB C30提供以下巨集,可以控制CPU的優先權,來達到抑制某些中斷需求的功能:
- Disabling所有中斷(IPL set to 7)

  Ex:

```
unsigned int ipl;
SET_AND_SAVE_CPU_IPL( ipl , 7 );
...
RESTORE_CPU_IPL( ipl );
```

- Disabling level 7以下的中斷n+1個指令週期

  Ex:

```
__builtin_disi( 16384 );
...
__builtin_disi( 0 );
```

ISR中變數的使用

volatile

- 如果在ISR與主程式中會共用到同一變數,則此變數在宣告時,必須加上*volatile*的關鍵字。

  Ex: *volatile* unsigned Ticks = 0;

- *volatile*關鍵字是用來避免C Compiler在進行最佳化時,將特定變數在最佳化的過程刪除,導致程式的流程出錯。

```
extern volatile unsigned int  PORTD __attribute__((__sfr__));
typedef struct tagPORTDBITS {
  unsigned RD0:1;
  unsigned RD1:1;
  unsigned RD2:1;
  unsigned RD3:1;
  unsigned RD4:1;
  unsigned RD5:1;
  unsigned RD6:1;
  unsigned RD7:1;
  unsigned RD8:1;
  unsigned RD9:1;
  unsigned RD10:1;
  unsigned RD11:1;
} PORTDBITS;
extern volatile PORTDBITS PORTDbits __attribute__((__sfr__));

extern volatile unsigned int  LATD __attribute__((__sfr__));
typedef struct tagLATDBITS {
  unsigned LATD0:1;
  unsigned LATD1:1;
  unsigned LATD2:1;
  unsigned LATD3:1;
  unsigned LATD4:1;
  unsigned LATD5:1;
  unsigned LATD6:1;
  unsigned LATD7:1;
  unsigned LATD8:1;
  unsigned LATD9:1;
  unsigned LATD10:1;
  unsigned LATD11:1;
} LATDBITS;
extern volatile LATDBITS LATDbits __attribute__((__sfr__));
```

一般變數的最佳化,結果符合原意!

```
B=A;
C=B;
最佳化後
C=A;
```

SFR的最佳化, 結果變成TMR1完全消失,不符合原意!

```
TMR1=A;
C=TMR1;
最佳化後
C=A;
```

!

*所有的SFR在MCU標頭檔中,都已經宣告為volatile,但記住!自訂的共用變數必須自行加上。(p24fj128gb106.h)

MICROCHIP
Regional Training Centers

Lab4 Interrupt Timer

- 以Lab3的程式基礎,將原本Polling Timer1中斷旗標的架構,改成中斷架構, 由中斷服務函式來負責RD0的Toggle。
- 先閱讀Timer Function的說明文件,了解Timer Function中,有關中斷的啟用與優先權設定方式。至少必須了解ConfigIntTimer1( )的用法。
- 與Lab3相同,以軟體模擬(MPLAB SIM)為工具,系統時脈(Fosc)設為8MHz。

Lab4 Interrupt Timer Step1

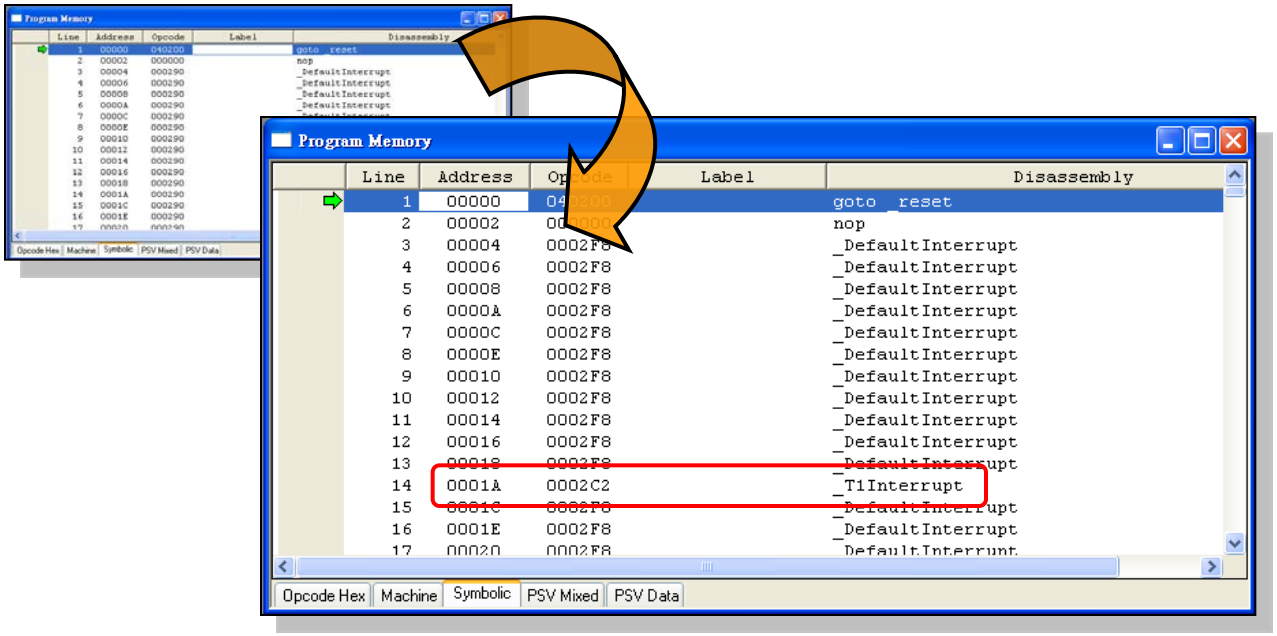
- 程式要如何改才能用改中斷架構？

  加入Timer1的中斷服務常式。將RD0的Toggle程式移入ISR中,記住還必須清除T1IF。
  利用ConfigIntTimer1( )啟用中斷,設定優先權。中斷優先權設定為”4”。
  將主程式的片段刪除,只留下while(1),主程式完全不做事情,RD0的Toggle的功能也不再在主程式內執行。只等待Timer1中斷發生。

Lab4 Interrupt Timer Step2

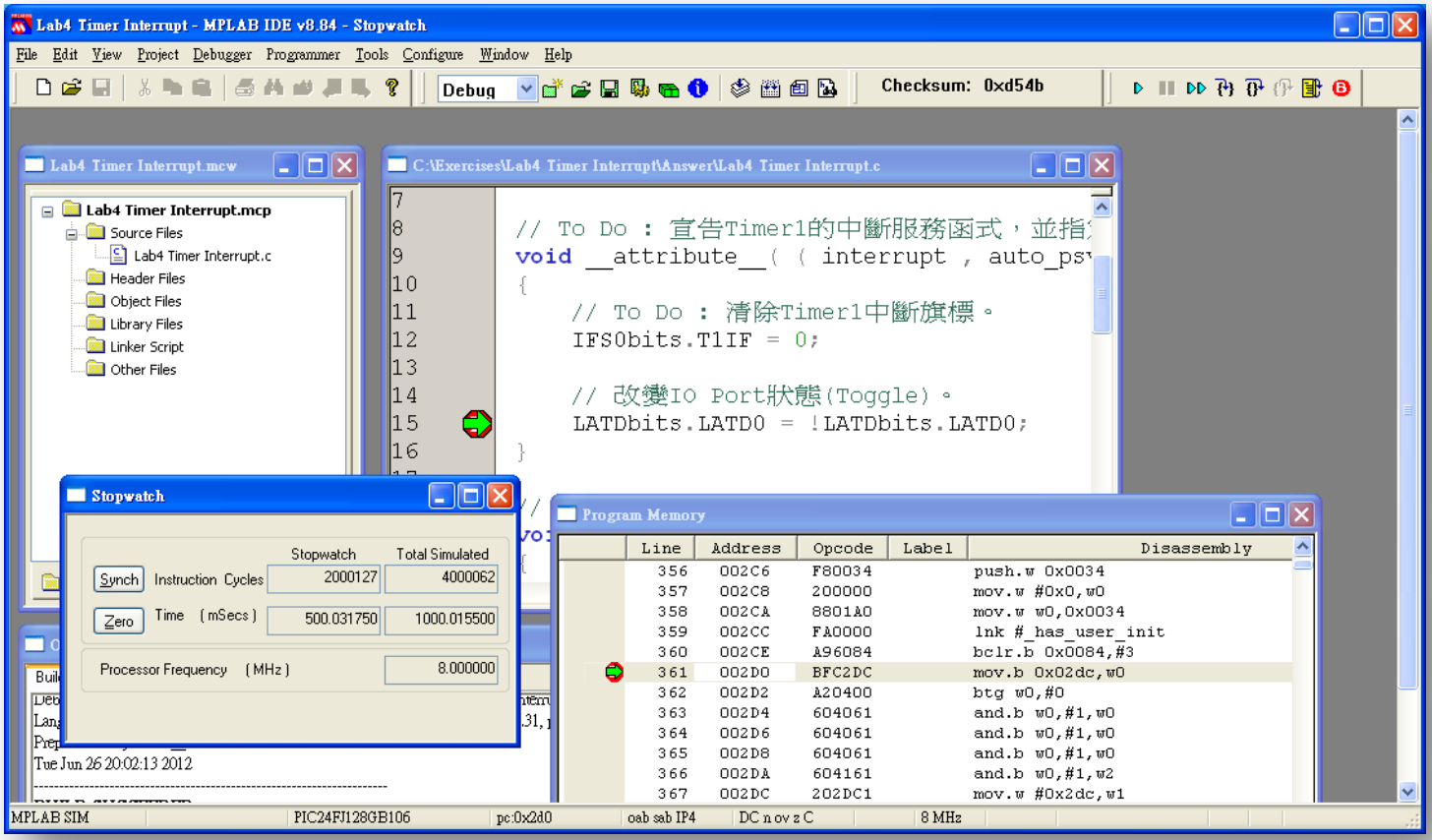
- 如何確認ISR有正確宣告？

利用MPLAB IDE功能表\view\program memory window來觀察。



Lab4 Interrupt Timer Step3

- 使用Watch Window跟Stopwatch來觀察程式的變化以及Timer的計數狀態。



MICROCHIP
Regional Training Centers

Default Interrupt

- 沒有被宣告的中斷,為何在中斷向量表中有ISR？

  在MPLAB C30中,預設將所有沒有宣告ISR的Iinterrupt給予預設的ISR, _DefaultInterrupt( )。
  _DefaultInterrupt( )的內容,只有一行指令reset。 也就是說,如果開啟了一個中斷,卻沒有正確的宣告ISR,那中斷發生時,就會跳到_DefaultInterrupt( ),直接Reset CPU。

- 如何得知發生了一個非預期的中斷？

  可以改寫DefaultInterrupt( ),讓CPU不要直接Reset。

MICROCHIP
Regional Training Centers

Lab4 Interrupt Timer Step4

- 請嘗試改寫_DefaultInterrupt( ),改寫方式跟一般的ISR宣告方式一樣,只是名字改成*_DefaultInterrupt*。

Ex:

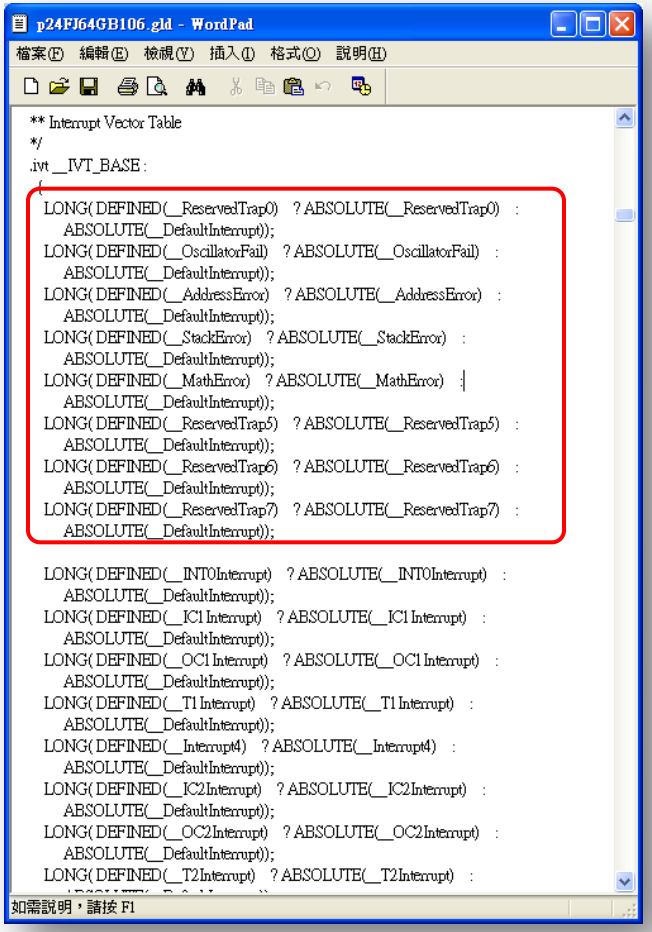
```
void __attribute__(( interrupt , auto_psv )) _DefaultInterrupt( void )
{
   while( 1 );
}
```

- 改寫_DefaultInterrupt( )有何好處?

  預設的_DefaultInterrupt( ),在發生非預期中斷時,會直接Reset,這樣便成無法觀察MCU發生什麼事情,只知道一值Reset。如果改寫成上面的程式,當程式停在while(1)時就知道有問題了,這時候可以透過Debug的工具,去檢查相關暫存器,來看看發生什麼事。

My Trap Service Routine

- Trap Service Routine沒有宣告時,一樣會套用DefaultInterrupt( ),當Trap發生時,直接Reset CPU。
- 可以對每個Trap都改寫自訂的Trap Service Routine,如此一來,當Trap發生時,就會跳到各自的Trap Service Routine,而不會直接Reset CPU。
- MPLAB C30中對Trap Service Routine名字的定義,一樣可以在MCU的連結檔(Linker Script, *.gld)找到。



Lab4 Interrupt Timer Step5

- 請嘗試加入自己的Stack Error Trap Service Routine,改寫方式跟一般的ISR宣告方式一樣,只是名字改成*_StackError*。

  Ex:

```
void __attribute__(( interrupt , auto_psv )) _StackError( void )
{
   while( 1 );
}
```

- 改寫Stack Error Trap Service Routine有何好處?

  改寫成上面的程式,當程式停在while(1)時就知道Stack發生錯誤了,有可能是Overflow或Underflow,這時候可以透過Debug的工具,去檢查相關暫存器,來看看發生什麼事。

Lab4 Interrupt Timer Step6

- 在Lab4中,使用Timer1的ISR來Toggle RD0。改寫 _DefaultInterrupt( ),並且宣告Timer1 ISR及Stack Error TSR。