Working with Interrupts on the PIC 32 Processor using C32

I. Single vs multi interrupt vectors
   a. Single interrupt vectors (see text pg. 91)
      i. all interrupts routed (vectored) through vector 0
      ii. must poll individual interrupt flags to determine which interrupts occurred
      iii. each interrupt flag must be reset in software
      iv. commonly used in non-nested interrupt schemes
      v. Priorities is determine in way code is constructed
      vi. advantages:
         1. Single ISR function – context saved only once – saves time and memory
         2. consistent with low end processors and earlier Microchip microcontrollers
         3. allows use of legacy code with porting to more advanced processors
      vii. disadvantages:
         1. requires polling
         2. difficult to prioritize
         3. difficult it implement nested interrupt scheme
   b. Multi interrupt vectors
      i. individual interrupt sources are vectored through separate vectors
      ii. Some resources still share a common interrupt vector. Examples
         1. Serial communications: Rx, Tx
      iii. each interrupt flag must be reset in software
      iv. commonly used in nested interrupt schemes
      v. hardware based priorities
         1. 7 priority levels
         2. 4 sub priority levels
         3. native or hardware priority
      vi. advantages
         1. easier to prioritize
         2. lower latency for higher priority events
         3. consistent with modern processors
      vii. disadvantages
         1. must write multiple ISRs
         2. uses more memory to save context

II. Elements to use interrupts
   a. Declaration of interrupt functions – see III below for additional details
   b. Initialize resources that generate interrupts
   c. Code that will be executed in response to an interrupt (interrupt service routine – ISR)

III. Methods to declare a function and ISR – refer to Table 5.3 page 99 of text

a.
```
void __attribute__( (interrupt(ipl1), vector(_TIMER_1_VECTOR)))
Timer1Handler ( void );

void  Timer1Handler(void)
{
      /* ISR code inserted here */
}
```

b.
```
#pragma interrupt Timer1Handler ipl1 vector 4

void  Timer1Handler(void)
{
      /* ISR code inserted here */
}
```

c. This method eliminates the requirement of a function prototype. Refer to the text on page 99 for vector entries.

```
void __ISR(_TIMER_1_VECTOR, ipl2)
Timer1Handler(void)
{
      /* ISR code inserted here */
}
```

IV. Initializing Resources to generate Interrupts

a. To enable multi vectored interrupts, the following line of code must always be executed before any interrupts will be processed:

```
INTEnableSystemMultiVectoredInt();
```

b. Required: set interrupt level between 1 and 7.
c. Optional : set sub priority level
d. Priority of interrupt service: 1) IP Level, 2) Sub Priority level, 3) natural level
e. Timer Interrupt Examples:
   i. macro version - See text Table 5.2 pg. 86 for (xx)

```
1. m(xx)IntEnable();
2. m(xx)SetIntPriority(y);      // 1 <= y <= 7
3. m(xx)SetIntSubPriority(z);   // 0 <= z <= 3
```

ii. void ConfigIntTimerx(unsigned int *config*);

```
                                  // x = 1,2,3,4,5,23,45
```

config definitions

Timer interrupt enable/disable

```
Tx_INT_ON
Tx_INT_OFF
(These bit fields are mutually exclusive)
```

Timer interrupt priorities

```
Tx_INT_PRIOR_7
Tx_INT_PRIOR_6
Tx_INT_PRIOR_5
Tx_INT_PRIOR_4
Tx_INT_PRIOR_3
Tx_INT_PRIOR_2
Tx_INT_PRIOR_1
```

```
        Tx_INT_PRIOR_0
        (These bit fields are mutually exclusive)
```

Timer interrupt sub- priorities

```
        Tx_INT_SUB_PRIOR_3
        Tx_INT_SUB_PRIOR_2
        Tx_INT_SUB_PRIOR_1
        Tx_INT_SUB_PRIOR_0
        (These bit fields are mutually exclusive)
```

1. `void EnableIntTx(void);    //x = 1,2,3,4,5,23,45`
   `SetPriorityIntTx (unsigned int config);`

   config definitions

   ```
   Tx_INT_PRIOR_7
   Tx_INT_PRIOR_6
   Tx_INT_PRIOR_5
   Tx_INT_PRIOR_4
   Tx_INT_PRIOR_3
   Tx_INT_PRIOR_2
   Tx_INT_PRIOR_1
   Tx_INT_PRIOR_0
   (These bit fields are mutually exclusive)
   ```

2. `INTSetPriority(INT_VECTOR vector, INT_PRIORITY priority);`
   `INTSetSubPriority(INT_VECTOR vector, INT_SUB_PRIORITY subPriority)`

   `See Peripheral Users Guide Table 8-1 for INT_VECTOR entries`

`INT_PRIORITY entries`	`INT_PRIORITY_LEVEL_7`
	`INT_PRIORITY_LEVEL_6`
	`INT_PRIORITY_LEVEL_5`
	`INT_PRIORITY_LEVEL_4`
	`INT_PRIORITY_LEVEL_3`
	`INT_PRIORITY_LEVEL_2`
	`INT_PRIORITY_LEVEL_1`
	`INT_PRIORITY_LEVEL_0`
`INT_SUB_PRIORITY entries`	`INT_SUB_PRIORITY_LEVEL_3`
	`INT_SUB_PRIORITY_LEVEL_2`
	`INT_SUB_PRIORITY_LEVEL_1`
	`INT_SUB_PRIORITY_LEVEL_0`

V. Interrupt Service Routines
   a. Interrupt flag must be cleared prior to exiting the ISR
      i. `m(xx)ClearIntFlag();` – - See text Table 5.2 pg. 86 for (xx)
      ii. `INTClearFlag(INT_VECTOR vector);`
         1. `See Peripheral Users Guide Table 8-1 for INT_VECTOR entries`
   b. The ISR must return a type VOID and have VOID parameters passed to it.
   c. An ISR can never be call like o conventional function

VI. Disabling interrupts for code protection
   a. `mINTDisableSystemMultiVectoredInt(); - mINTEnableSystemMultiVectoredInt();`
   b. `m(xx)IntDisable(); - m(xx)IntEnable();`
   c. `INTEnable(INT_SOURCE source, unsgined int enable)`
      i. **use 1 to enable interrupt, 0 to disable the interrupt**

VII. Example code for interrupts using peripheral library code

```
/* Declaration of interrupt ISR for Timer 1 at interrupt level 2 */
void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
   /* Optional ISR code inserted here */

   /* Required one of the following to statements to clear the
   interrupt flag in the ISR */

   mT1ClearIntFlag();

   // or

   INTClearFlag(INT_VECTOR vector);

}

void sysyem_initialize(void)
{
   /* code to set up IO ports */

   /* The following instruction sets up Timer 1 to set the T1 Flag at a rate of
   1000 Hz assumming that the PBCLK is running at 10 MHz.  This instruction
   sets the PR1 value to 1250. */

   OpenTimer1((T1_ON | T1_SOURCE_INT | T1_PS_1_8), 1250);

   /* The following enables the Timer 1 interrupt priority for 3 and the
   interrupt sub priority for 1.*/

   ConfigIntT1((T1_INT_ON | T1_INT_PRIORITY_3 | T1_INT_SUB_PRIORITY_1));

   /* code to for additional interrupt initializations */

   INTEnableSystemMultiVectoredInt();      // Required to enable all interrupts

}
```