In vectors.c, define the vector address by...

1. Add function prototype in this format: extern void near *yourISR*(void);

```
/*
*************************************************************************
*                  EXTERNAL ISR FUNCTION PROTOTYPES
*************************************************************************
*/

extern void near _Startup(void);                          /* Startup Routine.
extern void near  OSTickISR(void);                        /* OS Time Tick Routine.
extern void near  OSCtxSw(void);                          /* OS Contect Switch Routine.
extern void near  SevenSegDisp_ISR(void);                 /* Seven Segment Display ISR.
extern void near  ProbeRS232_RxTxISR(void);               /* Probe SCI ISR.
```

2. Find the appropriate vector in the tIsrFunc array (found in documentation or comments)...

```
/*
*************************************************************************
*                 INTERRUPT VECTORS
*************************************************************************
*/

typedef void (*near tIsrFunc)(void);
const tIsrFunc _vect[] @0xFF80 = {      /* Interrupt table                          */
        software_trap63,                /* 63 RESERVED                              */
        software_trap62,                /* 62 RESERVED                              */
        software_trap61,                /* 61 RESERVED                              */
        software_trap60,                /* 60 RESERVED                              */
```

… and replace the generic software trap with the function you prototyped earlier.

```
software_trap10,                /* 10 Enhanced Capture Timer channel 2       */
software_trap09,                /* 09 Enhanced Capture Timer channel 1       */
SevenSegDisp_ISR,               /* 08 Enhanced Capture Timer channel 0       */
software_trap07,                /* 07 Real Time Interrupt                    */
software_trap06,                /* 06 IRQ                                    */
```

3. "Write" an assembly file that defines the function given in the prototype. By this, copy the file sevenSegment.s and change a couple things…but do NOT CHANGE ANYTHING ELSE!!!!!!...

```
;
; Nots          : 1) This ISR should be used as a model for creating new ISRs under uC/OS-II
;                    using the MC9S12 Freescale architecture.
;
;               : 2) THIS FILE *MUST* BE LINKED INTO NON_BANKED MEMORY!
;*********************************************************************************************

NON_BANKED:       section

;*********************************************************************************************
;                                          I/O PORT ADDRESSES
;*********************************************************************************************

PPAGE:            equ    $0030         ; Addres of PPAGE register (assuming MC9S12 (non XGATE part)

;*********************************************************************************************
;                                         PUBLIC DECLARATIONS
;*********************************************************************************************

    xdef   SevenSegDisp_ISR
```

This should match the name of your function prototype, and the references circled below.

```
;*********************************************************************************************
;                                        EXTERNAL DECLARATIONS
;*********************************************************************************************

    xref   OSIntExit
    xref   OSIntNesting
    xref   OSTCBCur
    xref   SevenSegDisp_ISR_Handler
```

Change this external reference to the name of a C subroutine, which you'll add in the next step, and the references to this circled below.

```
;*********************************************************************************************
;                                          7-Segment ISR
;
; Description : This routine is the ISR for the Wytec Dragon12 7-Segment LED blocks. It calls
;               an ISR handler to perform most of the the I/O operations from C. The ISR
;               itself, like ALL other ISR's is coded in assembly so that the OS related
;               stack pointer manipulation can be performed prior to calling the associated
;               ISR Handler function.
;
; Arguments   : none
;
; Notes       : 1) All USER interrupts should be modeled EXACTLY like this where the only
;                  line to be modified is the call to your ISR_Handler and perhaps the call to
;                  the label name thisISRName1, See bne SevenSegDisp_ISR1.
;*********************************************************************************************

SevenSegDisp_ISR:
    ldaa   PPAGE                      ;  Get current value of PPAGE register
    psha                              ;  Push PPAGE register onto current task's stack

    inc    OSIntNesting               ;  Notify uC/OS-II about ISR

    ldab   OSIntNesting               ;  if (OSIntNesting == 1) {
    cmpb   #$01                       ;
    bne    SevenSegDisp_ISR1          ;

    ldy    OSTCBCur                   ;      OSTCBCur->OSTCBStkPtr = Stack Pointer
    sts    0,y                        ;  }

SevenSegDisp_ISR1:
    call   SevenSegDisp_ISR_Handler   ;  Call the Seven Segment Display ISR Handler in SevenSegDisp_BSP.c

    cli                               ;  Enable interrupts to allow interrupt nesting

    call   OSIntExit                  ;  Notify uC/OS-II about end of ISR

    pula                              ;  Get value of PPAGE register
    staa   PPAGE                      ;  Store into CPU's PPAGE register

    rti                               ;  Return from interrupt, no higher priority tasks ready.
```

4. Write a C file that defines the handler function above in the assembly file, as done in sevenSegment.c for SevenSegDisp_ISR_Handler. This function/subroutine is called from the ISR and must clear the interrupt's flag. However, since it is not the entry point for the ISR, it MUST NOT be defined with the "interrupt" keyword.

```
#ifndef SEVEN_SEG_BSP_H
#define SEVEN_SEG_BSP_H
```

Unique label for header

```
/*
*****************************************************************************************************
*                        Wytec Dragon12 Board Support Package
*
* File  : SevenSegDisp_BSP.h
* By    : Eric Shufro
*
* Notes: This file contains function prototypes for initializing and user the four, 7-Segment
*        LED blocks on the Wytec Dragon12 EVB.
*****************************************************************************************************
*/

/*
*****************************************************************************************************
*                                          PROTOTYPES
*****************************************************************************************************
*/
```

Called from a task once to initialize the interrupt (set the enable bit and perform configuration

```
void  SevenSegDisp_Init(void);
void  SevenSegWrite(INT16U num);
void  SevenSegDisp_ISR_Handler(void);
```

This must match the function declared in the assembly file.

```
#endif
```

```
void  SevenSegDisp_ISR_Handler (void)
{
   INT8U digits[4];
   INT8U currDispNum;
```

This must match the function declared in the assembly file.

```
   PTP        |=   0x0F;
   currDispNum = ((actBlockNum) % 4);
   actBlockNum++;
```

Do the work.

```
   digits[0]   =  (INT8U) (outputNum / 1000);
   digits[1]   =  (INT8U)((outputNum % 1000) / 100);
   digits[2]   =  (INT8U)((outputNum % 100)  / 10);
   digits[3]   =  (INT8U) (outputNum % 10);

   SevenSegOut(digits[currDispNum]);
   PTP        &= ~(1 << currDispNum);

#if SEVEN_SEG_OC == 0
    TFLG1 |= 0x01;
    TC0   += nbrCnts;
#endif
```

Clear the flag bit according to specific hardware.

Here are the "different" portions of code needed to get the second SCI port, SCI1, to receive from the PC via an SCI1 interrupt. These sections would go in the C file.

```
void    Sci1_Init(void) {

    SCI1CR2 = 0x00;     // disable the receiver, transmitter and interrupts
    SCI1BDH = 0x00;
    SCI1BDL = 156;
    SCI1CR1 = 0x00;     // 8 data bits, no parity, 1 stop bit
    SCI1CR2 = 0x0C;     // Enable the receiver and transmiter

    SCI1CR2 |=  SCI1SR1_RDRF_MASK;  // enable SCI Rx Interrupts
    SCI1CR2 &= ~SCI1SR1_TC_MASK;    // disable SCI Tx Interrupts
}
```

```
void    SCI_ISR_Handler (void) {

    int status  = SCI1SR1;
```

Your code here…at some point, you must read from SCI1DRL.

```
    SCI1SR1_RDRF = 0x00;
    return;
}
```