```
/*****************************************************************************
;*
;*            School of Engineering
;*
;*            The Robert Gordon University, Aberdeen
;*
;*****************************************************************************
;*
;*            Filename:   Task3.c
;*            Author:     Christopher Stoddart
;*            Language:   68hc12 Assembly Language
;*            Date:       21 Nov 2017
;*            Class:
;*
;*****************************************************************************
*
* Description:
* Design and write the code to complete part of the overall display software
* including the user interface and part of the real-time controlling elements,
* making use of the hardware and software elements covered in the first part of
* the module plus additional features not yet covered.
*
/*****************************include librarys**********************************/
#include <stdio.h>
#include "lcdv2.h"
#include "DBug12New.h"
#include <mc9s12dp512.h>
/*******************************Set Values*************************************/
#define   KEYPRESS 0x10          /* 0b00010000 key pressed bit mask    */
#define   KEYVAL   0x0f          /* 0b00001111 key code value bit mask */
#define   Alarm_Value  0x3800
#define   StartAddress 0x3800
/*****************************************************************************/

 void Alarm_count (void);              /* menu 'A' function*/
 void Test_Alarms(void);               /* menu 'B' function*/
 void Simulate_Alarms(void);           /* menu 'C' function*/
 void Timer_Init(void);               /* intialize timer*/
 void TimerCh2_ISR(void);              /* set interupt*/
 unsigned char Fstring;               /* code from alarm*/
 unsigned char Prevval = 0;           /* define prevval to 0*/
 unsigned char keycode;               /* code from keypad */
 unsigned char asciicode;             /* ascii code to be displayed */
 unsigned int value;                 /* return value from subroutine*/
 unsigned long delay;                 /* Allows time delay to vary*/
 unsigned int DELVAL;                 /* set Delval to Variable*/
 unsigned string;                    /* variable string length*/
 unsigned locatestring(unsigned int,int); /* subroutine message pointer*/
```

```
 char Fstring;                    /* variable string value*/

int main(void)
{
   char buf[24];                  // Memory buffer for output string

/***************************** Set up I/O ports ******************************/
   DDRH = 0x00;                   // make Port H an input port
   PERH = 0xFF;                   // enable Port H
/*****************************************************************************/
   lcd_init();                    // Initialize the LCD panel
          lcd_clear();                    // Clear LCD display

          lcd_move(0,4);                  // move position across 4 spaces
          lcd_putstr(" Acme Alarms ");          // display on lcd
          lcd_move(1,9);                  // move position down 1,across 9
          lcd_putstr("By");               // display on lcd
   lcd_move(2,0);                 // move position down 2
          lcd_putstr("Christopher Stoddart");          // display on lcd
   lcd_move(3,5);                 // move position down 3,across 5
          lcd_putstr("ID 1705593");          // display on lcd

   for (delay = 0; delay < 500000; delay++);    // Introduce Time Delay
   if (delay = 4900000)           // clear "splash screen"
   {
                  lcd_clear();                    // clear lcd
   }

          lcd_menu();                     // main menu
          Alarm_count();                    // menu 'A'
          Test_Alarms();                  // menu 'B'
          Simulate_Alarms();                  // menu 'C'

          asm( "swi" );                   // return to D-Bug12

          return 0;                       // return
   }
/*****************************************************************************
*   Name:                          lcd_menu
*   Description:    main menu for routine that the sub menus and functions
*                branch from.
*****************************************************************************/
   void lcd_menu(void)
   {

          lcd_move(0,0);                  // start position on lcd
   lcd_putstr("A - Count Alarms");          // display on lcd
          lcd_move(1,0);                  // move position down 1
```

```
        lcd_putstr("B - Test Alarms");          // display on lcd
        lcd_move(2,0);                          // move position down 2
        lcd_putstr("C - Simulated Alarm");      // display on lcd
        lcd_move(3,0);                          // move position down 3
        lcd_putstr("Press a Key");              // display on lcd

while( 1 )                              // response for keypress
        {
  while ( ( PTH & KEYPRESS ) == 0 );    // wait for key to be pressed
        keycode = PTH & KEYVAL;                 // read latched key code

        if     ((keycode) == 0xA)               // if 'A' is pressed
        {
           Alarm_count();                       // go to alarm_count function
   }
        else if ((keycode) == 0xB)              // if 'B' is pressed
        {
           Test_Alarms();                       // go to test_alarm function
        }
        else if ((keycode) == 0xC)              // if 'C' is pressed
        {
           Simulate_Alarms();                   // go to simulate_alarms funtion
        }
        else                                    // if invalid key is pressed
        {
        lcd_clear();                            // clear lcd
        lcd_init();                             // initalise lcd
        lcd_move(2,2);                          // move position down 2,across 2
        lcd_putstr("Invalid Keypress");         // display to lcd
        for (delay = 0; delay < 500000; delay++);   // set Time Delay for message
        lcd_menu();                             // return to main menu
        }
  while ( ( PTH & KEYPRESS ) != 0 );    // wait for key to be released
  }
 }
/*****************************************************************************
*
*   Name:                           Alarm_count
*   Description:    searches subroutine for number of alarms and returns a
*   value before displaying then returns to main menu.
*****************************************************************************/
        void Alarm_count(void)
        {
        char buf[24];                           // Memory buffer for output string
        unsigned char counter = 0;              // set intial counter value to 0

        lcd_clear();                            // clear lcd
        counter = AlarmCount(Alarm_Value);      // return value from subroutine
```

```
        sprintf(buf, "%d Alarm Types", counter);      // lcd memory buffer
        lcd_move(1,3);                          // move position down 1,across 3
        lcd_putstr( buf );                      // display to lcd
        lcd_move(3,2);                          // move position down 3,across 2
        lcd_putstr( "Any key for menu" );           // display to lcd

        for (delay = 0; delay < 100000; delay++);     // Introduce Time Delay

        while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
        while ( ( PTH & KEYPRESS ) != 0 );          // wait for key to be released
        lcd_clear();                          // clear lcd
        lcd_menu();                             // go to lcd main menu
        return;
        }
/*************************************************************************
*
*  Name:                          Test_Alarms
*  Description:   searches subroutine for an entered alarms and returns an
*  address, alarm type, alarm duration and string length which is displayed on the
*  lcd.
*************************************************************************/

        void Test_Alarms(void)
  {
        int combine;                          // define numeric variables holding whole numbers
  unsigned FirstAddr;                   // set variable 16 bits wide
  unsigned char Code;                    // set variable character code
  unsigned char Length;                  // set variable character legnth
  unsigned char FirstMSG;                  // set variable character message
  char buf[24];                        // Memory buffer for output string

  lcd_clear();                          // clear lcd display
  for (delay = 0; delay < 50000; delay++);      // Introduce Time Delay
        sprintf(buf, "Enter Alarm:");
  lcd_putxy(0,0,buf);                     // display to lcd


//*****************************For first Digit*******************************//
  while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
  keycode=PTH & KEYVAL;                   // read latched key code
  sprintf(buf,"%x", keycode);
  lcd_putxy(0,12,buf);                    // and dipslay it
  combine = keycode*100;                  // add top bits of number
  while ( ( PTH & KEYPRESS ) != 0 );          // wait for key to be released

//*****************************For Second Digit*******************************//
  while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
  keycode = PTH & KEYVAL;                 // read latched key code
```

```
   sprintf(buf, "%x", keycode);
   lcd_putxy(0,13,buf);                    // and dipslay it
   combine += keycode*10;                  // add top bits of number
   while ( ( PTH & KEYPRESS ) != 0 );      // wait for key to be released

//*****************************For Third Digit*******************************//
   while ( ( PTH & KEYPRESS ) == 0 );      // wait for key to be pressed
   keycode = PTH & KEYVAL;                 // read latched key code
   sprintf(buf, "%x", keycode);
   lcd_putxy(0,14,buf);                    // and display it
   combine += keycode;                     // add in lower 4 bits of number
   while ( ( PTH & KEYPRESS ) != 0 );      // wait for key to be released

//*****************************Display values********************************//
 // This snippet of code is bracketed out as i was unable to make the error string
 // alarm work when the string length was exceeded.
 // FirstMSG= MSGlen(StartAddress,combine);
 // if (FirstMSG==255)
 // {
 //lcd_clear();
 //lcd_move(1,2);
 //lcd_putstr("too large");
 //}
   FirstAddr = AlarmAddr(StartAddress,combine);  // find address of alarm
   if( FirstAddr==0)                       // if alarm isnt found
  {
   lcd_clear();                            // lcd clear
   lcd_move(1,2);                          // move position down 1,across 2
   lcd_putstr("Alarm Not Found");          // display on lcd
   for (delay = 0; delay < 500000; delay++);    // Introduce Time Delay
   Test_Alarms();                          // go to test alarm menu
  }
   else                                    // if alarm is found
  {
   lcd_move(1,0);                          // move position down 1
   lcd_putstr("Addr  Type  Len  Str\r");   // display alarm headers

   sprintf(buf, "%x", FirstAddr);
   lcd_putxy(2,0,buf);                     // and display it

   Code= AlarmCode(StartAddress,combine);  // find alarm code
   sprintf(buf,"%c", Code);
   lcd_putxy(2,7,buf);                     // and display it

   Length= AlarmLen(StartAddress,combine); // find length of alarm
   sprintf(buf,"%d", Length);
   lcd_putxy(2,13,buf);                    // and display it
```

```
  FirstMSG= MSGlen(StartAddress,combine);      // returns length of message
  sprintf(buf,"%d", FirstMSG);
  lcd_putxy(2,17,buf);

  lcd_move(3,2);                          // move position down 3,across 2
  lcd_putstr("Any key for Menu");             // display to lcd

  for (delay = 0; delay < 100000; delay++);     // time delay before key is pressed

        while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
        while ( ( PTH & KEYPRESS ) != 0 );          // wait for key to be released
        lcd_clear();                          // clear lcd
        lcd_menu();                           // go to lcd main menu

  }
  }
/*****************************************************************************
*
*  Name:                          Simulated_Alarms
*  Description:   Searches subroutine for an a alarm and returns a message and code
*  value, before playing an alarm according to the database using interupts.
*
*****************************************************************************/

  void Simulate_Alarms(void)
  {

        int combine;                          // define numeric variables holding whole numbers
  unsigned FirstAddr;                       // set variable 16 bits wide
  unsigned char Code;                       // set variable character code
  unsigned char Length;                     // set variable character legnth
  unsigned char FirstMSG;                   // set variable character message
  char buf[24];                           // Memory buffer for output string


  lcd_clear();                          // clear lcd display
  for (delay = 0; delay < 50000; delay++);      // Introduce Time Delay
        sprintf(buf, "Enter Alarm:");
  lcd_putxy(0,0,buf);                       // display to lcd

//*****************************For first Digit*********************************//
  while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
  keycode=PTH & KEYVAL;                     // read latched key code
  sprintf(buf,"%x", keycode);
  lcd_putxy(0,12,buf);                      // and display it
  combine= keycode*100;                     // add top bits of number
  while ( ( PTH & KEYPRESS ) != 0 );          // wait for key to be released
```

```
//*****************************For Second Digit*****************************//
   while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
   keycode = PTH & KEYVAL;                    // read latched key code
   sprintf(buf, "%x", keycode);
   lcd_putxy(0,13,buf);                      // display to lcd
   combine+= keycode*10;                      // add top bits of number
   while ( ( PTH & KEYPRESS ) != 0 );          // wait for key to be released

//*****************************For Third Digit*****************************//
   while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
   keycode = PTH & KEYVAL;                    // read latched key code
   sprintf(buf, "%x", keycode);
   lcd_putxy(0,14,buf);                      // display to lcd
   combine+= keycode;                        // add in lower 4 bits of number
   while ( ( PTH & KEYPRESS ) != 0 );

//*************************************************************************//

         string = locatestring(StartAddress,combine);   // string pointer value from subroutine
         FirstAddr = AlarmAddr(StartAddress,combine);   // find address of alarm
         if( FirstAddr==0)                        // if alarm isnt found
  {
   lcd_clear();                           // clear lcd
   lcd_move(1,2);                         // move position down 1,across 2
   lcd_putstr("Alarm Not Found");              // display error message to lcd
   for (delay = 0; delay < 500000; delay++);     // wait fot time delay
   Simulate_Alarms();                       // display simulate alarms menu
  }
   else
  {
   sprintf(buf, "Enter Alarm:");
   lcd_putxy(0,0,buf);                      // and display on lcd
   lcd_move(1,0);                          // move position down 1
   lcd_putstr("Alarm type");                  // display to lcd
   Code= AlarmCode(StartAddress,combine);       // return alarm type
   sprintf(buf, "%c", Code);
   lcd_putxy(1,11,buf);                      // and display it

   lcd_move (2,0);                         // move position down 2
   lcd_putstr((char*)string);                // display alarm message string
   lcd_move(3,0);                          // move position down 3
   lcd_putstr("press key to stop:");           // display to lcd
  }

   if (Code == 'a')                        // when a is pressed on keyboard
  {
         DELVAL = 750;                           // set low frequency tone
         VectTimerCh2 = (unsigned)TimerCh2_ISR;       // Set up interrupt vector
```

```
Timer_Init();                       // Initialise the timer
        asm("cli");                         // sets mask interupt
        while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
keycode = PTH & KEYVAL;                     // read latched key code

        if ((keycode)!= 0)                      // if key is pressed
         {
         TSCR1 = 0x00;                      // interupt alarm
         for (delay = 0; delay < 100000; delay++);    // introduce time delay
         lcd_clear();                       // clear lcd
         lcd_menu();                          // return to main menu
         }
        }
else if (Code=='b')                     // when b is pressed on keyboard
{
        DELVAL = 400;                       // set mid frequency tone
        VectTimerCh2 = (unsigned)TimerCh2_ISR;      // Set up interrupt vector
Timer_Init();                       // Initialise the timer
        asm("cli");                         // sets mask interupt
        while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
keycode = PTH & KEYVAL;                     // read latched key code

        if ((keycode)!= 0)                      // if key is pressed
        {
         TSCR1 = 0x00;                      // interupt alarm
         for (delay = 0; delay < 100000; delay++);    // introduce time delay
         lcd_clear();                       // clear lcd
         lcd_menu();                          // return to main menu
         }
        }
else if (Code=='c')                     // when c is pressed on keyboard
{
        DELVAL = 100;                       // set high frequency tone
        VectTimerCh2 = (unsigned)TimerCh2_ISR;      // Set up interrupt vector
Timer_Init();                       // Initialise the timer
        asm("cli");                         // sets mask interupt
        while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
keycode = PTH & KEYVAL;                     // read latched key code

        if ((keycode)!= 0)                      // if key is pressed
        {
         TSCR1 = 0x00;                      // interupt alarm
         for (delay = 0; delay < 100000; delay++);    // introduce time delay
         lcd_clear();                       // clear lcd menu
         lcd_menu();                          // return to main menu
         }
}
else if (Code=='f')                     // when f is pressed on keyboard
```

```
{
	DELVAL = 700;				// start with low frequency tone
	VectTimerCh2 = (unsigned)TimerCh2_ISR;	// Set up interrupt vector
Timer_Init();				// Initialise the timer
	asm("cli");				// sets mask interupt
	for (delay = 0; delay < 150000; delay++);	// operate for 1.5s time delay
	TSCR1 = 0x00;				// interupt alarm
	for (delay = 0; delay < 50000; delay++);	// off for 0.5s

	DELVAL = 600;				// raise frequency slightly
	VectTimerCh2 = (unsigned)TimerCh2_ISR;	// Set up interrupt vector
Timer_Init();				// Initialise the timer
	asm("cli");				// sets mask interupt
	for (delay = 0; delay < 150000; delay++);	// operate for 1.5s time delay
	TSCR1 = 0x00;				// interupt alarm
	for (delay = 0; delay < 50000; delay++);	// off for 0.5s

	DELVAL = 500;				// raise frequency slightly
	VectTimerCh2 = (unsigned)TimerCh2_ISR;	// Set up interrupt vector
Timer_Init();				// Initialise the timer
	asm("cli");				// sets mask interupt
	for (delay = 0; delay < 150000; delay++);	// operate for 1.5s time delay
	TSCR1 = 0x00;				// interupt alarm
	for (delay = 0; delay < 50000; delay++);	// off for 0.5s

	DELVAL = 400;				// raise frequency slightly
	VectTimerCh2 = (unsigned)TimerCh2_ISR;	// Set up interrupt vector
Timer_Init();				// Initialise the timer
	asm("cli");				// sets mask interupt
	for (delay = 0; delay < 150000; delay++);	// operate for 1.5s time delay
	TSCR1 = 0x00;				// interupt alarm
	for (delay = 0; delay < 50000; delay++);	// off for 0.5s

	DELVAL = 300;				// raise frequency slightly
	VectTimerCh2 = (unsigned)TimerCh2_ISR;	// Set up interrupt vector
Timer_Init();				// Initialise the timer
	asm("cli");				// sets mask interupt
	for (delay = 0; delay < 150000; delay++);	// operate for 1.5s time delay
	TSCR1 = 0x00;				// interupt alarm
	for (delay = 0; delay < 50000; delay++);	// off for 0.5s

	DELVAL = 200;				// raise frequency slightly
	VectTimerCh2 = (unsigned)TimerCh2_ISR;	// Set up interrupt vector
Timer_Init();				// Initialise the timer
	asm("cli");				// sets mask interupt
	for (delay = 0; delay < 150000; delay++);	// operate for 1.5s time delay
	TSCR1 = 0x00;				// interupt alarm
	for (delay = 0; delay < 50000; delay++);	// off for 0.5s
```

```
        DELVAL = 100;
        VectTimerCh2 = (unsigned)TimerCh2_ISR;        // Set up interrupt vector
 Timer_Init();                          // Initialise the timer
        asm("cli");                             // sets mask interupt
        for (delay = 0; delay < 150000; delay++);      // operate for 1.5s time delay
        TSCR1 = 0x00;                           // interupt alarm
        for (delay = 0; delay < 50000; delay++);       // off for 0.5s

        lcd_clear();                            // clear lcd display
        lcd_menu();                             // return to main menu
        }

 else if (Code=='d')                    // when d is pressed on keyboard
{
         while ( ( PTH & KEYPRESS ) == 0 )          // wait for key to be pressed
         {
         DELVAL = 400;                          // set mid frequency tone
         VectTimerCh2 = (unsigned)TimerCh2_ISR;       // Set up interrupt vector
  Timer_Init();                         // Initialise the timer
         asm("cli");                            // sets mask interupt

         for (delay = 0; delay < 50000; delay++);     // operate for 0.5s time delay
         TSCR1 = 0x00;                          // interupt alarm

         for (delay = 0; delay < 50000; delay++);     // off for 0.5s
         TSCR1 = 0x04;                          // interupt alarm
         }
        while ( ( PTH & KEYPRESS ) != 0 )           // wait for key to be pressed
        {
 TSCR1= 0x00;                           // interupt alarm
         for (delay = 0; delay < 100000; delay++);     // time delay to menu
         lcd_clear();                           // clear lcd display
         lcd_menu();                            // return to main menu
        }
        while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
        keycode = PTH & KEYVAL;                    // read latched key code

        for (delay = 0; delay < 100000; delay++);      // time delay to menu
        while ( ( PTH & KEYPRESS ) == 0 );          // wait for key to be pressed
        keycode = PTH & KEYVAL;                    // read latched key code
        while ( ( PTH & KEYPRESS ) != 0 )           // wait for key to be pressed
        lcd_clear();                            // clear lcd display
        lcd_menu();                             // return to main menu
}

else if (Code=='e')                     // when e is pressed on keyboard
{
```

```
        while ( ( PTH & KEYPRESS ) == 0 )            // wait for key to be pressed
        {
        DELVAL = 400;                          // set mid frequency tone
        VectTimerCh2 = (unsigned)TimerCh2_ISR;        // Set up interrupt vector
  Timer_Init();                          // Initialise the timer
        asm("cli");                            // sets mask interupt

        for (delay = 0; delay < 150000; delay++);      // operate for 1.5s time delay
        TSCR1 = 0x00;                          // interupt alarm

        for (delay = 0; delay < 50000; delay++);       // off for 0.5s
        TSCR1 = 0x04;                          // interupt alarm
        }
        while ( ( PTH & KEYPRESS ) != 0 )            // wait for key to be pressed
        {
  TSCR1= 0x00;                           // interupt alarm
        for (delay = 0; delay < 200000; delay++);      // 2s delay to menu
        lcd_clear();                           // clear lcd dispaly
        lcd_menu();                              // go to main menu

        }
        while ( ( PTH & KEYPRESS ) == 0 );           // wait for key to be pressed
         keycode = PTH & KEYVAL;                  // read latched key code
         }
        for (delay = 0; delay < 100000; delay++);      // 1s delay
        while ( ( PTH & KEYPRESS ) == 0 );           // wait for key to be pressed
        keycode = PTH & KEYVAL;                  // read latched key code
        while ( ( PTH & KEYPRESS ) != 0 )            // wait for key to be pressed
        lcd_clear();                           // clear lcd display
        lcd_menu();                              // go to main menu
          }

        /*************************************************************************
*
* Name:       Timer_Init
* Parameters:  none
* Returns:     nothing
* Globals:     none
* Description: Initialises the timer using channel 2 in toggle mode.
*                                   The prescaler is set to 32.
*
*************************************************************************/
  void Timer_Init(void)
 {
  TIE = 0x04;     // Enable interrupt for channel 2 - disable all others
  TSCR2 = 0x05;   // Set prescaler to 32
  TCTL2 = 0x10;   // Initialise OC2 to toggle on successful compare
  TIOS = 0x04;    // Make channel 2 function as an output compare
```

```
   TC3 = DELVAL;   // Set initial compare value
   TSCR1 = 0x80;   // Enable timer
  }
/*************************************************************************/
#pragma interrupt_handler TimerCh2_ISR

void TimerCh2_ISR( void )
{
   TC2 += DELVAL;   /* Calculate the next target count */
          TFLG1 |= 0x04;   /* Clear the OC2 flag */
}
```

Lcv2.h

```
void lcd_init(void);
void lcd_clear(void);
void lcd_putcmd(char *cmd);
void lcd_putstr(char *str);
void lcd_putxy(int line, int column, char *str);
void lcd_putchar (char);
void lcd_move(int, int);
void lcd_make(char, char[]);
```

```
/*************************************************************************
;*
;*              School of Engineering
;*
;*              The Robert Gordon University, Aberdeen
;*
;*************************************************************************
;*
;*              Filename:   spilcdsubs.s
;*              Author:     Christopher Stoddart
;*              Language:   68hc12 Assembly Language
;*              Date:       21 Nov 2017
;*              Class:      EN3540 Computer Architecture
;*
;*************************************************************************
;*
;* Description: This program contains the subroutines which interface
;*           with an LCD module using the SPI port. These are:
;*              lcd_init    initialise the LCD module
;*              lcd_clear   clear the LCD module
;*              lcd_putcmd  send a command to the LCM
;*              lcd_putstr  display a string at the current cursor position
```

```
;*             lcd_putxy   display a string at the specified cursor position
;*             lcd_make    dispalys character matrix on LCD
;*******************************************************************************
;*
;* Modification History:
;*  26 Feb 2006 Modified xferSPI to read Status Register before storing data in
;*           in Data Register - required for 68HCS12DP512.
;*  10 Oct 2012 Reformatted for use in Computer Architecture module.
;*
;*******************************************************************************/
WRDATA = 0x5f         ; Value used to write data byte
WRCMD = 0x1f          ; Value used to write command byte
RDDATA = 0x7f         ; Value used to read data byte
RDCMD = 0x3f          ; Value used to read command byte
Dummy = 0x00          ; Dummy status of MOSI pin during byte read
CS_BIT = 0x80         ; Bit position for the SS

;* SPI Register addresses for 68HCS12 (SPI Port 0)
SPI0CR1 = 0xd8        ; SPI Control Register 1
SPI0CR2 = 0xd9        ; SPI Control Register 2
SPI0BR = 0xda         ; SPI Baud Rate Register
SPI0SR = 0xdb         ; SPI Status Register
SPI0DR = 0xdd         ; SPI Data Register
PORTS = 0x248         ; Port S Data Register
DDRS = 0x24a          ; Port S Data Direction Register


;         .area    text


;***************************    Subroutine    ******************************
;
; Name:       _lcd_init
; Description: Initialises the SPI LCD module
; Parameters:  none
; Results:     none
; Reg altered: D - address of initialisation string
; Reg usage:
;
;*******************************************************************************
_lcd_init::
        bset    DDRS,0x80      ; Make /SS an output
        movb    #0x03,SPI0BR   ; Set SPI rate to 1.5 MHz (divide by 16)
        movb    #0x51,SPI0CR1  ; SPE + MSTR + /CPOL + CPHA + LSBFE
        movb    #0x00,SPI0CR2  ; as default
        ldd     #init_str      ; Point D at init. string address
        jsr     _lcd_putcmd    ; Send string to display module
        rts
```

```
;***************************    Subroutine    ******************************
;
; Name:       _lcd_clear
; Description: Clears the LCD module
; Parameters:  none
; Results:    none
; Reg altered: B - command byte
; Reg usage:
;
;*******************************************************************************
_lcd_clear::
        ldab   #0x01         ; Set up the command for clear screen
        jsr    WriteCmd      ; Send string to display module
        jsr    BusyWait      ; Wait until it's written
        rts


;***************************    Subroutine    ******************************
;
; Name:       xferSPI
; Description: Transfer byte to/from SPI peripheral
; Parameters:  A - Byte sent to SPI port
; Results:    A - Byte read from SPI peripheral
; Reg altered: A - As above
; Reg usage:
;
;*******************************************************************************
xferSPI:   tst    *SPI0SR       ; Need to read SR before writing to DR
        staa   *SPI0DR       ; Send byte to transfer to SPI data register
SPIwait:   tst    *SPI0SR       ; Test SPI status register
        bpl    SPIwait       ; Wait for bit 7 to be set
        ldaa   *SPI0DR       ; Load byte from data register into Acc A
        rts


;***************************    Subroutine    ******************************
;
; Name:       WriteData
; Description: Write a data byte to the LCD module
; Parameters:  ASCII value to write (in B)
; Results:    none
; Reg altered: none
; Reg usage:   B - value to write to LCD (on entry)
;           A - code for data write
;
;*******************************************************************************
```

```
WriteData:
        ldaa    #WRDATA         ; Set up command for write data
        jsr     WriteByte       ; Call subroutine to write byte to LCD
        rts


;**************************    Subroutine    ******************************
;
; Name:       WriteCmd
; Description: Write a command byte to the LCD module
; Parameters:  Command value to write (in B)
; Results:    none
; Reg altered: none
; Reg usage:   B - value to write to LCD (on entry)
;           A - code for command write
;
;*****************************************************************************
WriteCmd:
        ldaa    #WRCMD          ; Set up command for write command
        jsr     WriteByte       ; Call subroutine to write byte to LCD
        rts


;**************************    Subroutine    ******************************
;
; Name:       WriteByte
; Description: Write a byte to the SPI peripheral (LCD). Three bytes are sent.
;           1. Data/command write byte
;           2. 4 LSBs of byte (upper 4 bits set to zero)
;           3. 4 MSBs of byte in 4 LSB positions (upper 4 bits set to zero)
; Parameters:  Byte to write (in B); code for data/command write (in A)
; Results:    none
; Reg altered: none
; Reg usage:   B - byte to write to LCD (on entry)
;           A - code for data/command write
;
;*****************************************************************************
WriteByte:
        bclr    PORTS,#CS_BIT   ; Enable LCD SPI device
        jsr     xferSPI         ; Send byte to SPI LCD module
        tfr     b,a             ; Move value to Acc A (as parameter)
        anda    #0x0f           ; Clear upper 4 bits
        jsr     xferSPI         ; Send 4 LS bits to LCD (& 4 MS bits as 0)
        tfr     b,a             ; Move value to Acc A (as parameter)
        lsra                    ; Shift 4 MSBs to 4 LSB
        lsra
        lsra
        lsra
```

```
        jsr     xferSPI         ; Send 4 LS bits to LCD (& 4 MS bits as 0)
        bset    PORTS,#CS_BIT   ; Disable LCD SPI device
        rts


;****************************      Subroutine      ************************
;
; Name:         ReadData
; Description:    Send a read data command - and return the byte read
; Parameters:     none
; Results:       A = value read
; Reg altered:    A
;
;*******************************************************************************
ReadData:
        ldaa    #RDDATA
        jsr     ReadByte
        rts


;****************************      Subroutine      ************************
;
; Name:         ReadCmd
; Description:    Send a read command - and return the byte read
; Parameters:     none
; Results:       A = value read
; Reg altered:    A
;
;*******************************************************************************
ReadCmd:
        ldaa    #RDCMD
        jsr     ReadByte
        rts


;****************************      Subroutine      ************************
;
; Name:         ReadByte
; Description:    Read a byte - and return the byte read
; Parameters:     Read data/command byte (in A)
; Results:       A = value read
; Reg altered:    A
;
;*******************************************************************************
ReadByte:
        bclr    PORTS,#CS_BIT   ; Activate CS
        jsr     xferSPI         ; Send byte to SPI LCD module
        ldaa    #Dummy          ; Shift in the data LCD module
```

```
        jsr    xferSPI       ; Read byte from SPI peripheral
        bset   PORTS,#CS_BIT  ; Disable CS
        rts


;**************************    Subroutine    ******************************
;
; Name:      _lcd_putstr
; Description: displays a string to the LCM at the current cursor position
; Parameters:  start address of string (in D register)
; Results:    none
; Reg altered: AccA
; Reg usage:   AccA - char to display
;          X - pointer to next char to display
;
;*******************************************************************************
_lcd_putstr::
        pshx
        tfr    d,x          ; Transfer address of string to X
        ldaa   #WRDATA       ; Set up command for write data
        jsr    WriteBytes    ; Call subroutine to write byte to LCD
        pulx
        rts


;**************************    Subroutine    ******************************
;
; Name:      _lcd_putcmd
; Description: sends a command string to the LCD module
; Parameters:  start address of command (in D register)
; Results:    none
; Reg altered: AccA
; Reg usage:   AccA - command byte sent to display
;          X - pointer to next command byte
;
;*******************************************************************************
_lcd_putcmd::
        pshx
        tfr    d,x          ; Transfer address of string to X
        ldaa   #WRCMD        ; Set up command for write command
        jsr    WriteBytes    ; Call subroutine to write byte to LCD
        pulx
        rts


;*********************************Subroutine**********************************
;
; Name:      _lcd_putxy
```

```
; Description: displays a string on the LCM at the specified line & column
;           position
; Parameters:  line (in Acc B) - in range 0..3
;           column (on stack) - in range 0..19
;           string to be displayed (on stack)
; Results:    none
; Reg altered: D
; Reg usage:   Y - address of start of table of display data addresses
;
;*******************************************************************************
_lcd_putxy::
        pshy
        ldaa    5,sp          ; Fetch column value from stack
        ldy     #addr_tab     ; Point to DD address table
        adda    b,y           ; Calculate address of first character
        tfr     a,b           ; Copy DD address into B
        jsr     WriteCmd      ; Move cursor to that position
        jsr     BusyWait      ; Wait for busy flag to be cleared
        ldd     6,sp          ; Fetch string address
        jsr     _lcd_putstr   ; Display the string at that position
        puly
        rts

;*********************************Subroutine*************************************
;
; Name:       WriteBytes
; Description: Write a sequence of bytes. Address of first byte in X, data/
;           command byte in A.
; Parameters:  X - address of first byte in sequence
;           A - data/command byte
; Results:    none
; Reg altered: none
; Reg usage:   A - data/command byte
;           B - next byte from sequence
;           X - address of next byte in sequence
;
;*******************************************************************************
WriteBytes:
        ldab    1,x+          ; Fetch next byte
        beq     WriteDone     ; If it's a NULL, exit loop
        psha                  ; Save data/command byte
        jsr     WriteByte     ; Write the character
        jsr     BusyWait      ; Wait until it's written
        pula                  ; Restore data/command byte
        bra     WriteBytes
WriteDone:  rts
```

```
;*************************    Subroutine    ****************************
;
; Name:       BusyWait
; Description: Wait for LCD busy flag to clear
; Parameters:  none
; Results:    none
; Reg altered: none
; Reg usage:   A - port manipulation
;
;*****************************************************************************
BusyWait:
        jsr    ReadCmd
        bita   #0x80
        bne    BusyWait
        rts

init_str:   .byte   0x34,0x09,0x30,0x0e,0x06,0   ; LCD Initialisation codes
;               0x34 - 8-bit data length, set RE extension bit to 1
;               0x09 - 4 line mode
;               0x30 - 8-bit data length, set RE extension bit to 0
;               0x0e - display on, cursor on, no cursor blink
;               0x06 - auto-increment cursor
addr_tab:   .byte   0x80,0xa0,0xc0,0xe0    ; DD RAM Address Table
;               Rows actually start at 0x00, 0x20, 0x40 & 0x60 but MSB needs
;               to be set for command


;*************************     Subroutine     ************************
;
; Name:         lcd_putchar()
; Description: Display a single character at the current cursor position
; Parameters:  B - character to display
; Results:    none
; Reg altered: A - during subroutine calls
; Reg usage:   none
;*****************************************************************************
_lcd_putchar::


                jsr     WriteData     ;Call subroutine to write byte to LCD
                jsr     BusyWait      ;Call subroutine to Wait for LCD busy flag to clear

                rts

;*************************     Subroutine     ************************
;
; Name:         lcd_move::
; Description: Move the cursor to the specified position
```

```
; Parameters:  line (in Acc B) - in range 0..3
;            column (on stack) - in range 0..19
; Results:     none
; Reg altered: A, B
; Reg usage:   Y - address of start of table of display data addresses
;*******************************************************************************

_lcd_move::

                pshy                    ; push y
                ldaa        5,sp        ; Fetch value from Stack
                ldy         #addr_tab   ; Point to DD address table
                adda        b,y         ; calculate address of 1st character
                tfr         a,b         ; copy DD address into B
                jsr         WriteCmd    ; Move cursor to position
                jsr         BusyWait    ; Wait for busy flag to be cleared
                puly
                rts

;*********************************Subroutine*********************************
;
; Name:          lcd_make::
; Description:     Displays the character to the Lcd at the current cursor position
; Parameters:      Character number (in accumulator B) in the range 0-7 array of bytes
; Results:         None
; Register Altered: Accumulator B
; Register Usage:   Accumulator B to LCD Display
;                x register points to next char to display
;*******************************************************************************

_lcd_make::

            pshx                ; Push x register on to stack
            tfr         d,b     ; Transfer from d to b register
                lslb                ;
                lslb                ;
                lslb                ; shifts accumulator left 3 bits in total
                addb        #0x40   ; ADD 40 to start Address
                jsr         WriteCmd    ; Call subroutine to write byte to LCD
                jsr         BusyWait    ; Call subroutine to Wait for LCD busy flag to clear
                ldy         4,sp    ; load y register with address length from stack
                ldx         #8      ; Start of x value

          Loop:

                ldab        1,y+    ; Load byte into y accumulator and increment counter
                jsr         WriteData   ; Move cursor into position
                jsr         BusyWait    ; Wait for busy flag to clear
```

```
            dex                 ; Decrement the x register by a value of 1
            beq       Done      ; End when x is equal to 0
            jmp       Loop      ; Loop continues if x value is not 0

      Done:
        pulx              ; Restore the register
            rts               ; Return to C program


;*************************************************************************
;*                  School of Engineering
;*           The Robert Gordon University, Aberdeen
;*************************************************************************
;*           File name: alarm_subroutine.s
;*           Author:    Christopher Stoddart
;*           Created:   21th Of November 2017
;*           Revised:   21th Of November 2017
;*           Class:     EN3540 Computer Architecture
;*           Group:
;*************************************************************************
;*           M68HC12  Assembler Source File
;*
;* Description:
;*
;* The sample program copied from the Laboratory sheet:
;*
;*  Laboratory - Linking C and 68HC12 Assembly Language (Two Parameters)
;*
;*************************************************************************


;**************************    Subroutine    ****************************
;
; Name:       _AlarmCount
; Description: count number of alarms
; Parameters:  address of string (in x), search character (on stack), return value
; Results:     number of alarms (in x)
; Reg altered: x, Y, d
; Reg usage:   x - address of string on entry, position of character on exit
;
; It takes one parameter and returns a value. The parameter is the base address of
; the data structure, given as an unsigned 16 bit value. The return value is the
; count of the number of alarm records found given as an unsigned 8-bit value.
;
; The start address of the string is loaded into x.
; The search character is loaded into accumulator A.
; The index of the current character being tested is in the X register.
; On completion, the result is in the D accumulator.
```

```
;*******************************************************************************



_AlarmCount::
        pshx            ; value of X on is now saved on to the stack
                        pshy            ; value of y is now saved onto task
        tfr     d,x     ; transfer d to x register
        ldaa    #0      ; load contents of accumulator address

Loop:   ldy     1, x    ; Fetch next character; increment x
        adda    #1      ; increment by 1
                        cpy     #0x0000     ; compare to y register
                        ble     Match       ; Branch to Match if equal or less
                        tfr     y,x         ; Transfer y to x register
                        bra     Loop        ; branch to Loop

Match:  pulx            ; Restore X register from stack
        puly            ; Restore y register from stack
                        tfr     a,d         ; transfer a register to d register
        rts             ; Return to C program

;*************************    Subroutine    ****************************
;
; Name:      _AlarmAddr
; Description: find alarm address
; Parameters:  address of string (in x), search character (on stack), return value
; Results:    start address of alarm (in x)
; Reg altered: x, Y, d
; Reg usage:   x - address of string on entry, position of character on exit
;
; searches for the start address of the selected alarm. It takes two
; parameters and returns a value. The first parameter is the base address of the
; data structure, given as an unsigned 16 bit value. The second parameter is the
; alarm number, given as an unsigned 8 bit value. The return value is the start
; address of that alarm, given as an unsigned 16 bit address value.
;
; The start address of the string is loaded into x.
; The alarm number is loaded into accumulator A.
; The index of the current address being compared to is the x,y register.
; On completion, the result is in the D accumulator.
;*******************************************************************************
_AlarmAddr::
        pshx            ; value of X on is now saved on to the stack
                    tfr     d,x         ; transfer d to x register
                    ldaa    5,sp        ; load accumulator from stack pointer location

Loop1:
```

```
        ldab   0,x       ; Use (X) as address to get value to put in b
                          ldy    1,x       ; increment by 1 and load to y register
                cba              ; Compare A to memory
        beq    Match1    ; If match found, exit loop
                          cpy    #0x0000   ; copy to y register
                          beq    NoMatch1  ; if match, branch to NoMatch1
                          tfr    y,x       ; transfer y register to x
                          bra    Loop1     ; Repeat loop
NoMatch1:
        pulx              ; Restore X register from stack
                          ldd    #0                ; load d register with address
                          rts             ; Return to C program
Match1:
        tfr    x,d        ; Transfer x to regiter D
        pulx              ; Restore X register from stack
        rts             ; Return to C program

;*************************    Subroutine    ****************************
;
; Name:      _AlarmCode
; Description: find first instance of a parameter
; Parameters:  address of string (in x), search character (on stack), return value
; Results:    returns alarm type
; Reg altered: x, Y, d
; Reg usage:   x - address of string on entry, character type on exit
;
; This subroutine finds the alarm type. The return value will range from (a-f).
; They take two parameters and return a character. The first parameter is the base
; address of the data structure, given as an unsigned 16 bit value. The second
; parameter is the alarm ID number, given as an unsigned 8 bit value. Correct
; identification of the alarm returns an 8-bit quantity representing an ASCII
; character.
;
; The start address of the string is loaded into x.
; The alarm number is loaded into accumulator A.
; The index of the current address being compared to is the x,y register.
; On completion, the result is in the D accumulator.
;*******************************************************************************
 _AlarmCode::
        pshx              ; value of X on is now saved on to the stack
                          tfr    d,x        ; transfer d to x register
                    ldaa   5,sp       ; load accumulator A with effective address

Loop2:
        ldab   0,x            ; load accumulator A address
                          ldy    2,x        ; increment by 2 and load to y register
                          cba               ; Compare A to memory
                          beq    AddrFound  ; If match found, exit loop to AddrFound
```

```
                        cpy    #0x0000    ; copy to y register
                        beq    NoMatch2   ; if match, branch to NoMatch2
                        ldy    1,x        ; load y register with x value +1
                        tfr    y,x        ; transfer y to x register
                        bra    Loop2      ; Repeat loop

NoMatch2:
        pulx            ; Restore X register from stack
        rts                 ; Return to C program

AddrFound:
        tfr    y,d      ; Transfer index in D
                        pulx            ; Restore X register from stack
                        rts             ; Return to C program

;**************************    Subroutine    ****************************
;
; Name:       _AlarmLen
; Description: find alarm duration
; Parameters:  address of string (in x), search character (on stack), return value
; Results:    returns alarm duration
; Reg altered: x, Y, d
; Reg usage:   x - address of string on entry, position of character on exit
;
; This subroutine finds the alarm duration. The return value will range from (0-5).
; They take two parameters and return a character. The first parameter is the base
; address of the data structure, given as an unsigned 16 bit value. The second
; parameter is the alarm ID number, given as an unsigned 8 bit value. Correct
; identification of the alarm returns an 8-bit unsigned value.
;
; The start address of the string is loaded into x.
; The alarm number is loaded into accumulator A.
; The index of the current address being compared to is the x,y register.
; On completion, the result is in the D accumulator.
;*******************************************************************************
_AlarmLen::
        pshx            ; value of X on is now saved on to the stack
                        tfr    d,x      ; transfer d to x register
                    ldaa   5,sp      ; load accumulator A with effective address
Loop3:
        ldab   ,x       ; load to accumulator B
                        ldy    3,x      ; increment by 3 and load x value to y register
                        cba             ; Compare A to memory
                        beq    LgthFound  ; If match found, exit loop to AddrFound
                        cpy    #0x0000    ; copy to y register
                        beq    NotFound   ; if match, branch to NotFound
                        ldy    1,x      ; increment by 1 and x value to y register
                        tfr    y,x      ; transfer y to x register
```

```
                        bra     Loop3       ; Repeat loop

LgthFound:
                        tfr     y,d         ; transfer y to d register
                        pulx                ; Restore X register from stack
                        rts                 ; Return to C program
NotFound:
        pulx            ; Restore X register from stack
                        rts                 ; Return to C program

;**************************    Subroutine    ****************************
;
; Name:       _MSGlen
; Description: returns length of scrolling message
; Parameters:  address of string (in x), search character (on stack), return value
; Results:     returns lenght of message to display
; Reg altered: x, Y, d
; Reg usage:   x - address of string on entry, length of message on exit.

; This subroutine returns the length of the (scrolling) message. It takes two
; parameters and returns a value. The first parameter is the base address of the
; data structure, given as an unsigned 16 bit value. The second parameter is the
; alarm number, given as an unsigned 8 bit value. The return value is the length
; of the string given as an unsigned 8 bit value. If the length is greater than
; 100 characters then the value 255 should be returned.
;
; The start address of the string is loaded into x.
; The alarm number is loaded into accumulator A.
; The index of the current address being compared to is the x,y register.
; On completion, the result is in the D accumulator.
;*******************************************************************************
_MSGlen::
        pshx            ; value of X on is now saved on to the stack
                        tfr     d,x         ; transfer d to x register
                    ldaa    5,sp        ; load accumulator A from effective address
Loop4:

                        ldab    ,x          ; load to accumulator B
                        ldy     4,x         ; increment by 4 and load x value to y register
                        cba                 ; Compare A to memory
                        beq     StrngFound  ; If match found, exit loop to AddrFound
                        cpy     #0x0000     ; copy to y register
                        beq     NoMatch4    ; if match, branch to NoMatch4
                        ldy     1,x         ; increment by 1 and x value to y register
                        tfr     y,x         ; transfer y to x register
                        bra     Loop4       ; Repeat loop

NoMatch4:
```

```
          pulx            ; Restore X register from stack
                          rts             ; Return to C program

StrngFound:
        leax    5,x       ; add 5 to x
                          ldd     #0        ; load d register

Overloop:   tst     1,x+      ; Test the source, auto increment x
        beq     Exit      ; if match, branch to exit
                          addd    #1        ; add memory to d
                          cmpd    #100
                          beq     Plus      ; if match, branch to plus
                          bra     Overloop    ; Repeat loop

Plus:
        pulx            ; Restore X register from stack
                          ldd     #255      ; load d register with value
                          rts             ; Return to C program

Exit:
        pulx            ; Restore X register from stack
                          rts             ; Return to C program

;***************************    Subroutine    ****************************
;
; Name:       _locatestring
; Description: find first instance of a parameter
; Parameters:  address of string (in x), search character (on stack), return value
; Results:     alarm string location
; Reg altered: x, Y, d
; Reg usage:   x - address of string on entry, position of pointer on exit
;
; This subroutine returns the location of the string to display. It takes two
; parameters and returns a value. The first parameter is the base address of the
; data structure, given as an unsigned 16 bit value. The second parameter is the
; alarm number, given as an unsigned 8 bit value. The return value is the location
; of the string given as an unsigned 8 bit value.
;
; The start address of the string is loaded into x.
; The alarm number is loaded into accumulator A.
; The index of the current address being compared to is the x,y register.
; On completion, the result is in the D accumulator.
;*******************************************************************************
_locatestring::
        pshx            ; value of X on is now saved on to the stack
                          tfr     d,x       ; transfer d to x register
                      ldaa    5,sp      ; load accumulator A from effective address
Loop5:
```

```
        ldab    ,x          ; load to accumulator B
                            ldy     4,x         ; increment by 4 and load x value to y register
                            cba                 ; Compare A to memory
                            beq     LgthFound5  ; if match, branch to LgthFound5
                            cpy     #0x0000     ; copy to y register
                            beq     NotFound5   ; if match, branch to NotFound5
                            ldy     1,x         ; increment by 1 and x value to y register
                            tfr     y,x         ; transfer y to x register
                            bra     Loop5       ; Repeat loop

LgthFound5:
        leax    5,x
                            tfr     x,d         ; transfer x to d register
                            pulx                ; Restore X register from stack
                            rts                 ; Return to C program
NotFound5:
        ldd     #0
        pulx            ; Restore X register from stack
                            rts                 ; Return to C program


;*****************************************************************************
;*                      School of Engineering
;*              The Robert Gordon University, Aberdeen
;*****************************************************************************
;*              File name: Alarm_Table_1.s
;*              Author:    G Dunbar
;*              Created:   19 Oct 2017
;*              Revised:
;*              Class:
;*              Group:
;*****************************************************************************
;*              M68HC12  Assembler Source File
;*
;* Description:
;*
;* Data table for Coursework C2
;*
;* Information is stored in the form of a linked list of data records.
;* Assembly language labels are used to calculate the linked list addresses.
;* The last record contains a Null (zero) link so the record search can be
;* terminated.
;*
;* The format of the data is as follows:
;*
;* one:     .byte   1       ; one is a label marking the record start
;*                  ; 1 is a value representing the alarm number
;*                          .word   two     ; two is a label representing the link to next record
```

```
;*                          .byte   'c     ; a character representing the alarm sounder code
;*                    .byte   3      ; a constant representing the Duration/Repeat length
;*                          .asciz  "testing... " ; a string to be displayed
;*
;* Note:
;* Alarm records will be listed in increasing numerical order, but the list
;* might not be contiguous, i.e. it may skip values e.g. 1,2,5,6,37,255
;* Data files will be supplied that include display strings of varying lengths:
;*   - simple data (strings less than one LCD line length)
;*   - more complex (strings longer than one LCD line that will need to be
;*     scrolled to display properly
;*
;* Sounder codes:
;* a - low pitch continuous tone
;* b - mid pitch continuous tone
;* c - high pitch continuous tone
;* d - mid pitch short duration, i.e. beep, beep, beep .. etc
;* e - mid pitch long duration, i.e. beeep, beeep ..etc.
;* f - rising low to high long duration
;*
;* Duration/Repeat lengths
;* n - n seconds continuous or n repeats
;* a value of zero indicates endless duration or repeats
;* short - two beeps per second (on 0.25 s off 0.25 s)
;* long  - one beep per 1.5 s (on 1 s, off 0.5 s)
;*
;*****************************************************************************

data  =   0x3800          ; Alarm Storage Address
null  =   0x0000          ; last alarm pointer


        .area memory(abs)
        ;.area   text
                          .org    data

Alrm0:              .byte   0       ; Alarm: test
                          .word   Alrm1    ; Link
                          .byte   'd      ;
                          .byte   3       ; Duration/Repeat length
                          .asciz  "testing... "

Alrm1:              .byte   1       ; Alarm: fault condition
                          .word   Alrm3    ; Link
                          .byte   'a      ;
                          .byte   3       ;
                          .asciz  "system error"
```

```
Alrm3:          .byte   3       ; Alarm: power supply fluctuation
                .word   Alrm8    ; Link
                .byte   'b      ;
                .byte   5       ;
                .asciz  "power fluctuation"

Alrm8:          .byte   8       ; Alarm:
                .word   Alrm10   ; Link
                .byte   'c      ;
                .byte   5       ;
                .asciz  "too hot"

Alrm10:         .byte   10      ; Alarm:
                .word   Alrm57   ; Link
                .byte   'd      ;
                .byte   5       ;
                .asciz  "heater failure"

Alrm57:         .byte   57      ; Alarm:
                .word   Alrm213  ; Link
                .byte   'e      ;
                .byte   5       ;
                .asciz  "pump fault"

Alrm213:        .byte   213     ; Alarm:
                .word   null    ; Link
                .byte   'f      ;
                .byte   0       ;
                .asciz  "warp core breach!"
```

```
/* Vector table for DBug-12 version 4 for the Motorola 68HCS12DP512.
 * The address of an Interrupt Service Routine (ISR) can be set up using:
 *     VectTimerCh2 = (unsigned)TimerCh2_ISR;
 * where VectTimerCh2 is the vector for Timer Channel 2
 *       TimerCh2_ISR is the ISR (a C function) preceded by the command:
 * #pragma interrupt_handler TimerCh2_ISR
 *
 * Written by Grant Maxwell
 * 7 May 2005
 */

#define _INTVECT_BASE      0x3e00

#define _VP(off)       *(unsigned short volatile *)(_INTVECT_BASE + off)

#define VectRsrv0x80 _VP(0)
#define VectRsrv0x82 _VP(2)
#define VectRsrv0x84 _VP(4)
```

```
#define VectRsrv0x86 _VP(6)
#define VectRsrv0x88 _VP(8)
#define VectRsrv0x8A _VP(10)
#define VectPWMShDn _VP(12)
#define VectPortP _VP(14)
#define VectMSCAN4Tx _VP(16)
#define VectMSCAN4Rx _VP(18)
#define VectMSCAN4Errs _VP(20)
#define VectMSCAN4Wake _VP(22)
#define VectMSCAN3Tx _VP(24)
#define VectMSCAN3Rx _VP(26)
#define VectMSCAN3Errs _VP(28)
#define VectMSCAN3Wake _VP(30)
#define VectMSCAN2Tx _VP(32)
#define VectMSCAN2Rx _VP(34)
#define VectMSCAN2Errs _VP(36)
#define VectMSCAN2Wake _VP(38)
#define VectMSCAN1Tx _VP(40)
#define VectMSCAN1Rx _VP(42)
#define VectMSCAN1Errs _VP(44)
#define VectMSCAN1Wake _VP(46)
#define VectMSCAN0Tx _VP(48)
#define VectMSCAN0Rx _VP(50)
#define VectMSCAN0Errs _VP(52)
#define VectMSCAN0Wake _VP(54)
#define VectFlash _VP(56)
#define VectEEPROM _VP(58)
#define VectSPI2 _VP(60)
#define VectSPI1 _VP(62)
#define VectIIC _VP(64)
#define VectDLC _VP(66)
#define VectSCME _VP(68)
#define VectCRG _VP(70)
#define VectPAccBOv _VP(72)
#define VectModDwnCtr _VP(74)
#define VectPortH _VP(76)
#define VectPortJ _VP(78)
#define VectAtoD1 _VP(80)
#define VectAtoD0 _VP(82)
#define VectSCI1 _VP(84)
#define VectSCI0 _VP(86)
#define VectSPI0 _VP(88)
#define VectPAccEdge _VP(90)
#define VectPAccOvf _VP(92)
#define VectTimerOvf _VP(94)
#define VectTimerCh7 _VP(96)
#define VectTimerCh6 _VP(98)
#define VectTimerCh5 _VP(100)
```

```
#define VectTimerCh4 _VP(102)
#define VectTimerCh3 _VP(104)
#define VectTimerCh2 _VP(106)
#define VectTimerCh1 _VP(108)
#define VectTimerCh0 _VP(110)
#define VectRTI _VP(112)
#define VectIRQ _VP(114)
#define VectXIRQ _VP(116)
#define VectSWI _VP(118)
#define VectTrap _VP(120)
```