ECE 375 LAB 6

External Interrupts

Lab Time: Tuesday 4-6

Owen Markley

INTRODUCTION

The purpose of the lab is to learn how to use interrupts, and when they can be used. This is built upon the previous lab, or Lab 1 with the Bump Bot. Rather than polling for a button press or whisker bump, an interrupt signal is sent when one of the first four buttons is pressed or either of the whiskers is hit.

The lab provides experience with using the General Interrupt Enable bit, EICRA (External Interrupt Control Register A), EIMSK (External Interrupt Mask Register), EIFR (External Interrupt Flag Register). It was also necessary to use the ATmega128 data sheet to set up the board.

A lot of this program is built off of the existing BumpBot code and the LCDDisplay code.

PROGRAM OVERVIEW

The program first defines the various registers and constants what will need to be used. These include counters for the left and right whisker interrupts, mpr, counters for wait, loop counters, and the locations of various important bits. Macros are also defined for moving forwards, moving backwards, turning left and right, and halting.

Interrupt vectors are also then set up for each of the routines to be triggered by the interrupts.

Initialization then takes place, initializing the Stack Pointer, Port B, Port D, external interrupts, the External Interrupt Mask. Counters are then cleared and the LCD display is initialized.

Main simply loads the move forwards command, and loops forever. Interrupts will trigger the HitRight, HitLeft, RightClear, and LeftClear routines. Pressing a button will activate the interrupt signal which corresponds to these routines. With this implementation, the bot will continue forwards until one of the whiskers is hit. Based upon which whisker is hit, the bot will back up, and then turn left or right for a second each. After this has been done, the bot will continue forwards.

INITIALIZATION ROUTINE

The initialization routine provides a one-time initialization of key registers that allow the BumpBot program to execute correctly. First the Stack Pointer is initialized, allowing the proper use of function and subroutine calls. Port B was initialized to all outputs and will be used to direct the motors. Port D was initialized to inputs and will receive the whisker input. Next the external interrupts are initialized ensuring that only the falling edge of the interrupt signal will activate something. The External Interrupt Mask is then set up by loading a 1 into the first four bits, allowing these interrupts to be used. Left and Right count are cleared, and the LDCInit routine is called to prepare the display and its associated routines. Finally, sei is called to activate the Global Interrupt Bit.

MAIN ROUTINE

The main routine is a very short loop that continuously loads the mpr with move forwards command, and outputs this to POTB. This loop repeats infinitely, and will only stop for the interrupt signals which the board is listening for.

HITRIGHT ROUTINE

First, the rightcnt register is incremented. After this, it is loaded into the mpr using mov, this is because mpr has to hold the value to be converted from Binary to ASCII String. The high and low of X register then has to be loaded with the address to the LCDLn1 string. Once this is done, BIN2ASCII can be called, and then with the new string in X, the value can be written to the string using LCDWrLn1. Mpr is then pushed as well as waitcnt, then in mpr, SREG is called to save the program state. Then the Flag register is set so that more signals cannot come through. The code to move the bot back is then called, and the proper pins are activated, and output to PORTB. Wait is then called for a second so that the bot moves backwards for the proper amount of time. The pins to turn the bot left is then loaded, after this is done, another second is allowed to pass. The Flag Register is then set to be all ones. Once this is complete, the program state is restored and returns to the main.

HitLeft Routine

The HitLeft routine is identical to the HitRight routine, except that a Turn Right command is sent to PORTB instead. This then fills the requirement for the basic BumpBot behavior.

ClearLeft Routine

Performs the same operations as the HitRight, in order to prevent other interrupt signals from being received by the board. Between these operations, the left count register is cleared, and the LDCClrLn1 function is run.

ClearRight Routine

This is the same as the above routine, however instead of right, the leftcnt is cleared, and line 2 of the display is cleared.

Wait Routine

The Wait routine requires a single argument provided in the *waitcnt* register. A triple-nested loop will provide busy cycles as such that $16 + 159975 \cdot waitcnt$ cycles will be executed, or roughly $waitcnt \cdot 10\text{ms}$. In order to use this routine, first the *waitcnt* register must be loaded with the number of 10ms intervals, i.e. for one second, the *waitcnt* must contain a value of 100. Then a call to the routine will perform the precision wait cycle. (Function remained same as in the Lab1)

Additional Questions

1) *As this lab, Lab 1, and Lab 2 have demonstrated, there are always multiple ways to accomplish the same task when programming (this is especially true for assembly programming). As an engineer, you will need to be able to justify your design choices. You have now seen the BumpBot behavior implemented using two different programming languages (AVR assembly and C), and also using two different methods of receiving external input (polling and interrupts).*

   Explain the benefits and costs of each of these approaches. Some important areas of interest include, but are not limited to: efficiency, speed, cost of context switching, programming time, understandability, etc.

   Personally, I liked coding in C a lot better, seeing as I am more familiar with it. I think that C is easier to learn based on the syntax. However, since Assembly is lower level, it is good for directly programming to the hardware. It also allows me to work with registers, interrupts, and memory easier.

Polling has the benefit of only taking data when the code is in the perfect state to handle it, while Interrupts can occur at any time. However, interrupts can be more efficient due to the fact that no time is wasted checking when there is nothing to retrieve. I think that in this instance interrupts are better.

2) *Instead of using the Wait function that was provided in BasicBumpBot.asm, is it possible to use a timer/counter interrupt to perform the one-second delays that are a part of the BumpBot behavior, while still using external interrupts for the bumpers? Give a reasonable argument either way, and be sure to mention if interrupt priority had any effect on your answer.*

   Yes, the timer has the ability to trigger an interrupt upon reaching a specified number. I see no reason that this interrupt should not be able to be used alongside the external interrupts. I think that the programmer should be able to use the Mask Register to allow for only the timer interrupt to be received when performing another subroutine. Or the programmer might be able to have the timer interrupt be queued for once the current routine is finished

DIFFICULTIES

For some reason some of the registers would not allow me to use the LDI operations. To remedy this I used the clear function to reset the counters, and used mov instead of ldi to move the data into the MPR. There is also a light that will not go off when using the LCDDriver file.

CONCLUSION

In this lab we were required to take the existing Bump Bot code, and configure the code to use interrupts rather than polling. Each interrupt also had to be counted and displayed on the different lines of the ATMega128 LCD display. Buttons would trigger both the interrupts for the “whiskers” as well as the clear lines.

SOURCE CODE

Source code for Lab 6, includes both the main file, and the LCD Driver file from Lab 4 which I re-used in this lab.

```
;***********************************************************
;* Owen_Markley_Lab4_sourcecode.asm
;*
;* Basic Bumpbot program has been reconfigured to respond to interupt
;* as opposed to polling for inputs. Each time a bump occurs and a whisker
;* is triggered, a counter will increment on the LCD Display either representing
;* the count of collisions on the left or right whisker.
;*
;* Lab 6
;*
;***********************************************************

.include "m128def.inc"                          ; Include definition file

;***********************************************************
;* Variable and Constant Declarations
;***********************************************************
.def    rightcnt = r1               ; right interrupt counter
.def    leftcnt = r2                ; left interrupt counter
.def    mpr = r16                       ; Multi-Purpose Register
```

```
.def   waitcnt = r23              ; Rest Loop Counter
.def   ilcnt = r24                      ; Inner Loop Counter
.def   olcnt = r25                      ; Outer Loop Counter


.equ   WTime = 100                      ; Time to wait in wait loop

.equ   WskrR = 0                        ; Right Whisker Input Bit
.equ   WskrL = 1                        ; Left Whisker Input Bit
.equ   EngEnR = 4                       ; Right Engine Enable Bit
.equ   EngEnL = 7                       ; Left Engine Enable Bit
.equ   EngDirR = 5                      ; Right Engine Direction Bit
.equ   EngDirL = 6                      ; Left Engine Direction Bit

;/////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
;/////////////////////////////////////////////////////////////

.equ   MovFwd = (1<<EngDirR|1<<EngDirL)   ; Move Forward Command
.equ   MovBck = $00                       ; Move Backward Command
.equ   TurnR = (1<<EngDirL)               ; Turn Right Command
.equ   TurnL = (1<<EngDirR)               ; Turn Left Command
.equ   Halt = (1<<EngEnR|1<<EngEnL)             ; Halt Command

;***********************************************************
;* Beginning of code segment
;***********************************************************
.cseg

;-----------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------
.org   $0000                      ; Reset and Power On Interrupt
              rjmp    INIT        ; Jump to program initialization

.org   $0002 ;{IRQ0 => pin0, PORTD}
              rcall HitRight            ; Calls the hit right function on interrupt
              reti

.org   $0004 ;{IRQ1 => pin1, PORTD}
              rcall HitLeft       ; Calls the hit left function on interrupt
              reti

.org   $0006 ;{IRQ2 => pin2, PORTD}
              rcall RightClear    ; calls the clear funciton on interrupt
              reti

.org   $0008 ;{IRQ3 => pin3, PORTD}
              rcall LeftClear           ; calls the right clear function in interrupt
              reti

.org   $0046                      ; End of Interrupt Vectors
;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:
    ; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
              ldi           mpr, low(RAMEND) ;low end of stack pointer initialized
```

```
		out		SPL, mpr		; Load SPL with low byte of RAMEND
		ldi		mpr, high(RAMEND) ;high end of stack pointer initialized
		out		SPH, mpr		; Load SPH with high byte of RAMEND

	; Initialize Port B for output
		ldi		mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL) ; Set
Port B Data Direction Register
		out		DDRB, mpr		; for output
		ldi		mpr, $00		; Initialize Port B Data Register
		out		PORTB, mpr		; so all Port B outputs are low

	; Initialize Port D for input
		ldi		mpr, (0<<WskrL)|(0<<WskrR)	; Set Port D Data Direction
Register
		out		DDRD, mpr		; for input
		ldi		mpr, (1<<WskrL)|(1<<WskrR)	; Initialize Port D Data
Register
		out		PORTD, mpr		; so all Port D inputs are Tri-State
	;Initialize external interrupts
		;Set the Interupts Sense control to falling edge
	ldi mpr, (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10) ; setting these values allows
for the falling edge to trigger
	sts EICRA, mpr ;binary value is loaded into external interrupt control register

	;Set the External Interrupt Mask
	ldi mpr, (1<<INT0)|(1<<INT1)|(1<<INT2)|(1<<INT3) ; last four digits in value are
set to 1
	out EIMSK, mpr ; setting the external interrupt mask register allows for signal to
go through on these interrupts

	;intitialise the counters
	clr leftcnt ; counter set to zero
	clr rightcnt ; right counter set to zero

	;Inititalize LCD
	rcall LCDInit ; LCD display is initialized using provided driver


	sei ; global interrupt enable

;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:
	;Move Robot Forwards
	ldi mpr, MovFwd ; both engine bits enabled and forwards, value loaded into mpr
	out PORTB, mpr ; output to LED's

	rjmp MAIN ; loop forever

;***********************************************************
;* Subroutines and Functions
;***********************************************************

;-----------------------------------------------------------
; Sub: HitRight
; Desc:		Handles functionality of the TekBot when the right whisker
;		is triggered.
```

```
;-----------------------------------------------------------------
HitRight:

              inc rightcnt ; add one to right bumper hit count

              mov mpr, rightcnt ; rightcnt is loaded into mpr to prepare for BIN2ASCII
function
              ldi XL, low(LCDLn1Addr) ; output will be placed into low and high of lcd
line 1 Addr
              ldi XH, high(LCDLn1Addr)
              rcall Bin2ASCII ; converts the bin value to to an ascii and stores it into
the lcd line 1 address
              rcall LCDWrLn1 ; writes value in line address to the the the top line of
lcd display

              push   mpr                  ; Save mpr register
              push   waitcnt                    ; Save wait register
              in            mpr, SREG     ; Save program state
              push   mpr                  ;

              ldi mpr, 0b00000000 ; value of zero is loaded into mpr
              out EIFR, mpr ; filling the flag register with zeroes will clear any
requests for interrupts

              ; Move Backwards for a second
              ldi           mpr, MovBck   ; Load Move Backward command
              out           PORTB, mpr    ; Send command to port
              ldi           waitcnt, WTime      ; Rest for 1 second
              rcall  Rest                 ; Call wait function

              ; Turn left for a second
              ldi           mpr, TurnL    ; Load Turn Left Command
              out           PORTB, mpr    ; Send command to port
              ldi           waitcnt, WTime      ; Rest for 1 second
              rcall  Rest                 ; Call wait function

              ldi mpr, 0b11111111 ; ones are loaded into the mpr
              out EIFR, mpr ; flags are then set to logical high

              pop           mpr           ; Restore program state
              out           SREG, mpr     ;
              pop           waitcnt             ; Restore wait register
              pop           mpr           ; Restore mpr
              ret                         ; Return from subroutine

;-----------------------------------------------------------------
; Sub: HitLeft
; Desc:       Handles functionality of the TekBot when the left whisker
;             is triggered.
;-----------------------------------------------------------------
HitLeft:
              inc leftcnt ; left counter is incremented by one

              mov mpr, leftcnt ; leftcnt loaded into mpr for bin2ascii
              ldi XL, low(LCDLn2Addr) ; address of line two is loaded into X
              ldi XH, high(LCDLn2Addr)
              rcall Bin2ASCII ; converts binary value to ascii string
              rcall LCDWrLn2 ; writes the converted string to the string
```

```
                push    mpr                     ; Save mpr register
                push    waitcnt                         ; Save wait register
                in              mpr, SREG       ; Save program state
                push    mpr                     ;

                ldi mpr, 0b00000000 ; zeroes are loaded into mpr
                out EIFR, mpr ; flags are set to zero, preventing more signals for the
moment

                ; Move Backwards for a second
                ldi             mpr, MovBck     ; Load Move Backward command
                out             PORTB, mpr      ; Send command to port
                ldi             waitcnt, WTime          ; Rest for 1 second
                rcall   Rest                    ; Call wait function

                ; Turn right for a second
                ldi             mpr, TurnR      ; Load Turn Left Command
                out             PORTB, mpr      ; Send command to port
                ldi             waitcnt, WTime          ; Rest for 1 second
                rcall   Rest                    ; Call wait function

                ldi mpr, 0b11111111 ; ones are loaded into the mpr
                out EIFR, mpr ; flags are then set to logical high

                pop             mpr             ; Restore program state
                out             SREG, mpr       ;
                pop             waitcnt                 ; Restore wait register
                pop             mpr             ; Restore mpr
                ret                             ; Return from subroutine

;-----------------------------------------------------------------
; Sub: Rest
; Desc:         A wait loop that is 16 + 159975*waitcnt cycles or roughly
;               waitcnt*10ms.  Just initialize wait for the specific amount
;               of time in 10ms intervals. Here is the general eqaution
;               for the number of clock cycles in the wait loop:
;                       ((3 * ilcnt + 3) * olcnt + 3) * waitcnt + 13 + call
;-----------------------------------------------------------------
Rest:
                push    waitcnt                         ; Save wait register
                push    ilcnt                   ; Save ilcnt register
                push    olcnt                   ; Save olcnt register

Loop:   ldi             olcnt, 224              ; load olcnt register
OLoop:  ldi             ilcnt, 237              ; load ilcnt register
ILoop:  dec             ilcnt                   ; decrement ilcnt
                brne    ILoop                   ; Continue Inner Loop
                dec             olcnt           ; decrement olcnt
                brne    OLoop                   ; Continue Outer Loop
                dec             waitcnt                 ; Decrement wait
                brne    Loop                    ; Continue Rest loop

                pop             olcnt           ; Restore olcnt register
                pop             ilcnt           ; Restore ilcnt register
                pop             waitcnt                 ; Restore wait register
                ret                             ; Return from subroutine
```

```
;-----------------------------------------------------------------
; Sub: RightClear
; Desc:       Clears the counter on the LCD display top line for the amount
;             of times that the right whisker has been triggered by the
;             interrupt
;-----------------------------------------------------------------
RightClear:

       ldi mpr, 0b00000000 ; repeats same process as above to prevent other interrupts
from occurring at same time
       out EIFR, mpr

       clr rightcnt ; clears the right count, resets to zero
       rcall LCDClrLn1 ; clears the first line of the lcd, doesn't really work and clears
whole screen

       ldi mpr, 0b11111111
       out EIFR, mpr

;-----------------------------------------------------------------
; Sub: LeftClear
; Desc:       Clears the counter on the LCD display bottom line for the amount
;             of times that the left whisker has been triggered by the
;             interrupt
;-----------------------------------------------------------------
LeftClear:

       ldi mpr, 0b00000000 ; repeats same process as above to prevent other interrupts
from occurring at same time
       out EIFR, mpr

       clr leftcnt ; clears the right count, resets to zero
       rcall LCDClrLn2 ; clears the first line of the lcd, doesn't really work and clears
whole screen

       ldi mpr, 0b11111111
       out EIFR, mpr

;***********************************************************
;*     Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"          ; Include the LCD Driver
```

```
;***********************************************************
;*
;*     LCDDriver.asm -      V2.0
;*
;*     Contains the neccessary functions to display text to a
;*     2 x 16 character LCD Display.  Additional functions
;*     include a conversion routine from an unsigned 8-bit
;*     binary number to and ASCII text string.
;*
;*     Version 2.0 - Added support for accessing the LCD
;*             Display via the serial port. See version 1.0 for
;*             accessing a memory mapped LCD display.
```

```
;*
;***********************************************************
;*
;*      Author: David Zier
;*        Date: March 17, 2003
;*     Company: TekBots(TM), Oregon State University - EECS
;*     Version: 2.0
;*
;***********************************************************
;*     Rev    Date    Name          Description
;*---------------------------------------------------------
;*     -      8/20/02       Zier          Initial Creation of Version 1.0
;*     A      3/7/03 Zier          V2.0 - Updated for USART LCD
;*
;*
;***********************************************************

;***********************************************************
;*     Internal Register Definitions and Constants
;*            NOTE: A register MUST be named 'mpr' in the Main Code
;*                  It is recomended to use register r16.
;*            WARNING: Register r17-r22 are reserved and cannot be
;*                  renamed outside of the LCD Driver functions. Doing
;*                  so will damage the functionality of the LCD Driver
;***********************************************************
.def   wait = r17                          ; Wait Loop Register
.def   count = r18                         ; Character Counter
.def   line = r19                          ; Line Select Register
.def   type = r20                          ; LCD data type: Command or Text
.def   q = r21                                    ; Quotient for div10
.def   r = r22                                    ; Remander for div10

.equ   LCDLine1 = $80                      ; LCD Line 1 select command
.equ   LCDLine2 = $c0                      ; LCD Line 2 select command
.equ   LCDClear = $01                      ; LCD Clear Command
.equ   LCDHome = $02                ; LCD Set Cursor Home Command
.equ   LCDPulse = $08                      ; LCD Pulse signal, used to simulate
                                                  ; write signal

.equ   LCDCmd = $00                ; Constant used to write a command
.equ   LCDTxt = $01                ; Constant used to write a text character

.equ   LCDMaxCnt = 16                      ; Maximum number of characters per line
.equ   LCDLn1Addr = $0100          ; Beginning address for Line 1 data
.equ   LCDLn2Addr = $0110          ; Beginning address for Line 2 data

;-----------------------------------------------------------
;***********************************************************
;*     Public LCD Driver Suboutines and Functions
;*            These functions and subroutines can be called safely
;*            from within any program
;***********************************************************
;-----------------------------------------------------------


;*********************************************************
;* SubRt:     LCDInit
;* Desc:      Initialize the Serial Port and the Hitachi
```

```
;*                  Display 8 Bit inc DD-RAM
;*                  Pointer with no features
;*                  - 2 LInes with 16 characters
;*****************************************************
LCDInit:
              push   mpr                          ; Save the state of machine
              in            mpr, SREG             ; Save the SREG
              push   mpr                          ;
              push   wait                 ; Save wait

              ; Setup the Communication Ports
              ; Port B: Output
              ; Port D: Input w/ internal pullup resistors
              ; Port F: Output on Pin 3
              ldi           mpr, $00              ; Initialize Port B for outputs
              out           PORTB, mpr            ; Port B outputs high
              ldi           mpr, $ff              ; except for any overrides
              out           DDRB, mpr             ;
              ldi           mpr, $00              ; Initialize Port D for inputs
              out           PORTD, mpr            ; with Tri-State
              ldi           mpr, $00              ; except for any overrides
              out           DDRD, mpr             ;
              ldi           mpr, $00              ; Initialize Port F Pin 3 to
              sts           PORTF, mpr            ; output inorder to twiddle the
              ldi           mpr, (1<<DDF3)        ; LCD interface
              sts           DDRF, mpr             ; Must NOT override this port

              ; Setup the Serial Functionality
              ; SPI Type: Master
              ; SPI Clock Rate: 2*1000.000 kHz
              ; SPI Clock Phase: Cycle Half
              ; SPI Clock Polarity: Low
              ; SPI Data Order: MSB First
              ldi           mpr, (1<<SPE|1<<MSTR)
              out           SPCR, mpr             ; Set Serial Port Control Register
              ldi           mpr, (1<<SPI2X)
              out           SPSR, mpr             ; Set Serial Port Status Register

              ; Setup External SRAM configuration
              ; $0460 - $7FFF / $8000 - $FFFF
              ; Lower page wait state(s): None
              ; Uppoer page wait state(s): 2r/w
              ldi           mpr, (1<<SRE) ;
              out           MCUCR, mpr            ; Initialize MCUCR
              ldi           mpr, (1<<SRL2|1<<SRW11)
              sts           XMCRA, mpr            ; Initialize XMCRA
              ldi           mpr, (1<<XMBK)        ;
              sts           XMCRB, mpr            ; Initialize XMCRB

              ; Initialize USART0
              ; Communication Parameter: 8 bit, 1 stop, No Parity
              ; USART0 Rx: On
              ; USART0 Tx: On
              ; USART0 Mode: Asynchronous
              ; USART0 Baudrate: 9600
              ldi           mpr, $00              ;
              out           UCSR0A, mpr           ; Init UCSR0A
              ldi           mpr, (1<<RXEN0|1<<TXEN0)
```

```
                out             UCSR0B, mpr             ; Init UCSR0B
                ldi             mpr, (1<<UCSZ01|1<<UCSZ00)
                sts             UCSR0C, mpr             ; Init UCSR0C
                ldi             mpr, $00                ;
                sts             UBRR0H, mpr             ; Init UBRR0H
                ldi             mpr, $67                ;
                out             UBRR0L, mpr             ; Init UBRR0L

                ; Initialize the LCD Display
                ldi             mpr, 6                  ;
LCDINIT_L1:
                ldi             wait, 250               ; 15ms of Display
                rcall   LCDWait                         ; Bootup wait
                dec             mpr                             ;
                brne    LCDINIT_L1              ;

                ldi             mpr, $38                ; Display Mode set
                rcall   LCDWriteCmd             ;
                ldi             mpr, $08                ; Display Off
                rcall   LCDWriteCmd             ;
                ldi             mpr, $01                ; Display Clear
                rcall   LCDWriteCmd             ;
                ldi             mpr, $06                ; Entry mode set
                rcall   LCDWriteCmd             ;
                ldi             mpr, $0c                ; Display on
                rcall   LCDWriteCmd             ;
                rcall   LCDClr                  ; Clear display

                pop             wait                    ; Restore wait
                pop             mpr                             ; Restore SREG
                out             SREG, mpr               ;
                pop             mpr                             ; Restore mpr
                ret                                             ; Return from subroutine

;*************************************************************
;* Func:        LCDWrite
;* Desc:        Generic Write Function that writes both lines
;*                      of text out to the LCD
;*                      - Line 1 data is in address space $0100-$010F
;*                      - Line 2 data is in address space $0110-$010F
;*************************************************************
LCDWrite:
                rcall LCDWrLn1                          ; Write Line 1
                rcall LCDWrLn2                          ; Write Line 2
                ret                                     ; Return from function

;*************************************************************
;* Func:        LCDWrLn1
;* Desc:        This function will write the first line of
;*                      data to the first line of the LCD Display
;*************************************************************
LCDWrLn1:
                push    mpr                             ; Save mpr
                push    ZL                              ; Save Z pointer
                push    ZH                              ;
                push    count                   ; Save the count register
                push    line                    ; Save the line register
```

```
		ldi		ZL, low(LCDLn1Addr)
		ldi		ZH, high(LCDLn1Addr)
		ldi		line, LCDLine1		; Set LCD line to Line 1
		rcall	LCDSetLine		; Restart at the beginning of line 1
		rcall	LCDWriteLine	; Write the line of text

		pop		line
		pop		count			; Restore the counter
		pop		ZH				; Restore Z pointer
		pop		ZL				;
		pop	mpr				; Restore mpr
		ret					; Return from function

;*****************************************************
;* Func:	LCDWrLn2
;* Desc:	This function will write the second line of
;*		data to the second line of the LCD Display
;*****************************************************
LCDWrLn2:
		push	mpr			; Save mpr
		push	ZL			; Save Z pointer
		push	ZH			;
		push	count			; Save the count register
		push	line			; Save the line register

		ldi		ZL, low(LCDLn2Addr)
		ldi		ZH, high(LCDLn2Addr)
		ldi		line, LCDLine2		; Set LCD line to Line 2
		rcall	LCDSetLine		; Restart at the beginning of line 2
		rcall	LCDWriteLine	; Write the line of text

		pop		line
		pop		count			; Restore the counter
		pop		ZH				; Restore Z pointer
		pop		ZL				;
		pop	mpr				; Restore mpr
		ret					; Return from function

;*****************************************************
;* Func:	LCDClr
;* Desc:	Generic Clear Subroutine that clears both
;*		lines of the LCD and Data Memory storage area
;*****************************************************
LCDClr:
		rcall	LCDClrLn1		; Clear Line 1
		rcall	LCDClrLn2		; Clear Line 2
		ret					; Return from Subroutine

;*****************************************************
;* Func:	LCDClrLn1
;* Desc:	This subroutine will clear the first line of
;*		the data and the first line of the LCD Display
;*****************************************************
LCDClrLn1:
		push	mpr			; Save mpr
		push	line			; Save line register
		push	count			; Save the count register
		push	ZL			; Save Z pointer
```

```
                push    ZH                          ;

                ldi             line, LCDline1      ; Set Access to Line 1 of LCD
                rcall   LCDSetLine              ; Set Z pointer to address of line 1 data
                ldi             ZL, low(LCDLn1Addr)
                ldi             ZH, high(LCDLn1Addr)
                rcall   LCDClrLine              ; Call the Clear Line function

                pop             ZH                          ; Restore Z pointer
                pop             ZL                          ;
                pop             count               ; Restore the count register
                pop             line                ; Restore line register
                pop             mpr                         ; Restore mpr
                ret                                         ; Return from Subroutine

;*****************************************************
;* Func:        LCDClrLn2
;* Desc:        This subroutine will clear the second line of
;*                      the data and the second line of the LCD Display
;*****************************************************
LCDClrLn2:
                push    mpr                         ; Save mpr
                push    line                ; Save line register
                push    count               ; Save the count register
                push    ZL                          ; Save Z pointer
                push    ZH                          ;

                ldi             line, LCDline2      ; Set Access to Line 2 of LCD
                rcall   LCDSetLine              ; Set Z pointer to address of line 2 data
                ldi             ZL, low(LCDLn2Addr)
                ldi             ZH, high(LCDLn2Addr)
                rcall   LCDClrLine              ; Call the Clear Line function

                pop             ZH                          ; Restore Z pointer
                pop             ZL                          ;
                pop             count               ; Restore the count register
                pop             line                ; Restore line register
                pop             mpr                         ; Restore mpr
                ret                                         ; Return from Subroutine

;*****************************************************
;* Func:        LCDWriteByte
;* Desc:        This is a complex and low level function that
;*                      allows any program to write any ASCII character
;*                      (Byte) anywhere in the LCD Display.  There
;*                      are several things that need to be initialized
;*                      before this function is called:
;*              count - Holds the index value of the line to where
;*                              the char is written, 0-15(39).  i.e. if
;*                              count has the value of 3, then the char is
;*                              going to be written to the third element of
;*                              the line.
;*              line  - Holds the line number that the char is going
;*                              to be written to, (1 or 2).
;*              mpr      - Contains the value of the ASCII character to
;*                              be written (0-255)
;*****************************************************
LCDWriteByte:
```

```
                push    mpr                             ; Save the mpr
                push    line                    ; Save the line
                push    count                   ; Save the count
                                                        ; Preform sanity checks on count
and line
                cpi             count, 40               ; Make sure count is within range
                brsh    LCDWriteByte_3          ; Do nothing and exit function
                cpi             line, 1                 ; If (line == 1)
                brne    LCDWriteByte_1          ;
                ldi             line, LCDLine1          ; Load line 1 base LCD Address
                rjmp    LCDWriteByte_2          ; Continue on with function
LCDWriteByte_1:
                cpi             line, 2                 ; If (line == 2)
                brne    LCDWriteByte_3          ; Do nothing and exit function
                ldi             line, LCDLine2          ; Load line 2 base LCD Address

LCDWriteByte_2:                                 ; Write char to LCD
                add             line, count             ; Set the correct LCD address
                rcall   LCDSetLine              ; Set the line address to LCD
                rcall   LCDWriteChar  ; Write Char to LCD Display

LCDWriteByte_3:                                 ; Exit Function
                pop             count                   ; Restore the count
                pop             line                    ; Restore the line
                pop             mpr                     ; Restore the mpr
                ret                                     ; Return from function

;***********************************************************
;* Func:      Bin2ASCII
;* Desc:      Converts a binary number into an ASCII
;*                    text string equivalent.
;*                    - The binary number needs to be in the mpr
;*                    - The Start Address of where the text will
;*                          be placed needs to be in the X Register
;*                    - The count of the characters created are
;*                          added to the count register
;***********************************************************
Bin2ASCII:
                push    mpr                             ; save mpr
                push    r                               ; save r
                push    q                               ; save q
                push    XH                              ; save X-pointer
                push    XL                              ;

                ; Determine the range of mpr
                cpi             mpr, 100                ; is mpr >= 100
                brlo    B2A_1                   ; goto next check
                ldi             count, 3                ; Three chars are written
                adiw    XL, 3                   ; Increment X 3 address spaces
                rjmp    B2A_3                   ; Continue with program
B2A_1: cpi              mpr, 10                         ; is mpr >= 10
                brlo    B2A_2                   ; Continue with program
                ldi             count, 2                ; Two chars are written
                adiw    XL, 2                   ; Increment X 2 address spaces
                rjmp    B2A_3                   ; Continue with program
B2A_2: adiw     XL, 1                   ; Increment X 1 address space
                ldi             count, 1                ; One char is written
```

```
B2A_3: ;Do-While statement that converts Binary to ASCII
              rcall  div10                ; Call the div10 function
              ldi           mpr, '0'              ; Set the base ASCII integer value
              add           mpr, r                ; Create the ASCII integer value
              st            -X, mpr                     ; Load ASCII value to memory
              mov           mpr, q                ; Set mpr to quotiant value
              cpi           mpr, 0                ; does mpr == 0
              brne   B2A_3                ; do while (mpr != 0)

              pop           XL                          ; restore X-pointer
              pop           XH                          ;
              pop    q                            ; restore q
              pop           r                           ; restore r
              pop           mpr                         ; restore mpr
              ret                                       ; return from function

;-----------------------------------------------------------
;***********************************************************
;* Private LCD Driver Functions and Subroutines
;*     NOTE: It is not recommended to call these functions
;*           or subroutines, only call the Public ones.
;***********************************************************
;-----------------------------------------------------------

;***********************************************************
;* Func:      LCDSetLine
;* Desc:      Change line to be written to
;***********************************************************
LCDSetLine:
              push   mpr                          ; Save mpr
              mov           mpr,line              ; Copy Command Data to mpr
              rcall  LCDWriteCmd          ; Write the Command
              pop           mpr                         ; Restore the mpr
              ret                                       ; Return from function

;***********************************************************
;* Func:      LCDClrLine
;* Desc:      Manually clears a single line within an LCD
;*                   Display and Data Memory by writing 16
;*                   consecutive ASCII spaces $20 to both the LCD
;*                   and the memory.  The line to be cleared must
;*                   first be set in the LCD and the Z pointer is
;*                   pointing the first element in Data Memory
;***********************************************************
LCDClrLine:
              ldi           mpr, ' '              ; The space char to be written
              ldi           count, LCDMaxCnt; The character count
LCDClrLine_1:
              st            Z+, mpr                     ; Clear data memory element
              rcall  LCDWriteChar  ; Clear LCD memory element
              dec           count                 ; Decrement the count
              brne   LCDClrLine_1  ; Continue untill all elements are cleared
              ret                                       ; Return from function

;***********************************************************
;* Func:      LCDWriteLine
;* Desc:      Writes a line of text to the LCD Display.
;*                   This routine takes a data element pointed to
```

```
;*                  by the Z-pointer and copies it to the LCD
;*                  Display for the duration of the line.  The
;*                  line the Z-pointer must be set prior to the
;*                  function call.
;*****************************************************
LCDWriteLine:
              ldi           count, LCDMaxCnt; The character count
LCDWriteLine_1:
              ld            mpr, Z+                     ; Get the data element
              rcall  LCDWriteChar  ; Write element to LCD Display
              dec           count               ; Decrement the count
              brne   LCDWriteLine_1       ; Continue untill all elements are written
              ret                                       ; Return from function

;*****************************************************
;* Func:      LCDWriteCmd
;* Desc:      Write command that is in the mpr to LCD
;*****************************************************
LCDWriteCmd:
              push   type                ; Save type register
              push   wait                ; Save wait register
              ldi           type, LCDCmd  ; Set type to Command data
              rcall  LCDWriteData  ; Write data to LCD
              push   mpr                          ; Save mpr register
              ldi           mpr, 2                ; Wait approx. 4.1 ms
LCDWC_L1:
              ldi           wait, 205             ; Wait 2050 us
              rcall  LCDWait                      ;
              dec           mpr                         ; The wait loop cont.
              brne   LCDWC_L1            ;
              pop           mpr                         ; Restore mpr
              pop           wait                ; Restore wait register
              pop           type                ; Restore type register
              ret                                       ; Return from function

;*****************************************************
;* Func:      LCDWriteChar
;* Desc:      Write character data that is in the mpr
;*****************************************************
LCDWriteChar:
              push   type                ; Save type register
              push   wait                ; Save the wait register
              ldi           type, LCDTxt  ; Set type to Text data
              rcall  LCDWriteData  ; Write data to LCD
              ldi           wait, 16              ; Delay 160 us
              rcall  LCDWait                      ;
              pop           wait                ; Restore wait register
              pop           type                ; Restore type register
              ret                                       ; Return from function

;*****************************************************
;* Func:      LCDWriteData
;* Desc:      Write data or command to LCD
;*****************************************************
LCDWriteData:
              out           SPDR, type          ; Send type to SP
              ldi           wait, 2                     ; Wait 2 us
              rcall  LCDWait                    ; Call Wait function
```

```
            out         SPDR,mpr            ; Send data to serial port
            ldi         wait, 2                     ; Wait 2 us
            rcall  LCDWait                  ; Call Wait function
            ldi         wait, LCDPulse      ; Use wait temporarially to
            sts         PORTF, wait         ; to send write pulse to LCD
            ldi         wait, $00           ;
            sts         PORTF, wait         ;
            ret                                     ; Return from function

;***********************************************************
;* Func:      LCDWait
;* Desc:      A wait loop that is 10 + 159*wait cycles or
;*                  roughly wait*10us.  Just initialize wait
;*                  for the specific amount of time in 10us
;*                  intervals.
;***********************************************************
LCDWait:push  mpr                       ; Save mpr
LCDW_L1:ldi         mpr, $49            ; Load with a 10us value
LCDW_L2:dec         mpr                         ; Inner Wait Loop
            brne   LCDW_L2
            dec         wait                ; Outer Wait Loop
            brne   LCDW_L1
            pop         mpr                         ; Restore mpr
            ret                                     ; Return from Wait Function

;***********************************************************
;*     Bin2ASCII routines that can be used as a psuedo-
;*                  printf function to convert an 8-bit binary
;*                  number into the unigned decimal ASCII text
;***********************************************************

;************************************************************
;* Func:      div10
;* Desc:      Divides the value in the mpr by 10 and
;*                  puts the remander in the 'r' register and
;*                  and the quotiant in the 'q' register.
;*     DO NOT modify this function, trust me, it does
;*     divide by 10 :)  ~DZ
;************************************************************
div10:
            push   r0                       ; Save register

            ; q = mpr / 10 = mpr * 0.000110011001101b
            mov         q, mpr              ; q = mpr * 1.0b
            lsr         q                           ; q >> 2
            lsr         q                           ; q = mpr * 0.01b
            add         q, mpr              ; q = (q + mpr) >> 1
            lsr         q                           ; q = mpr * 0.101b
            add         q, mpr              ; q = (q + mpr) >> 3
            lsr         q
            lsr         q
            lsr         q                           ; q = mpr * 0.001101b
            add         q, mpr              ; q = (q + mpr) >> 1
            lsr         q                           ; q = mpr * 0.1001101b
            add         q, mpr              ; q = (q + mpr) >> 3
            lsr         q
            lsr         q
            lsr         q                           ; q = mpr * 0.0011001101b
```

```
            add         q, mpr              ; q = (q + mpr) >> 1
            lsr         q                       ; q = mpr * 0.10011001101b
            add         q, mpr              ; q = (q + mpr) >> 4
            lsr         q
            lsr         q
            lsr         q
            lsr         q                       ; q = mpr * 0.000110011001101b

            ; compute the remainder as r = i - 10 * q
            ; calculate r = q * 10 = q * 1010b
            mov         r, q                ; r = q * 1
            lsl         r                       ; r << 2
            lsl         r                       ; r = q * 100b
            add         r, q                ; r = (r + q) << 1
            lsl         r                       ; r = q * 1010b
            mov         r0, r               ; r0 = 10 * q
            mov         r, mpr              ; r = mpr
            sub         r, r0               ; r = mpr - 10 * q

            ; Fix any errors that occur
div10_1:cpi         r, 10               ; Compare with 10
            brlo    div10_2                 ; do nothing if r < 10
            inc         q                       ; fix qoutient
            subi    r, 10               ; fix remainder
            rjmp    div10_1                 ; Continue until error is corrected

div10_2:pop         r0                      ; Restore registers
            ret                             ; Return from function
```