ECE 375 Lab 4

Data Manipulation and the LCD Display

Lab Time: Thursday 10:00 - 11:50

Jeremy Fischer

TA Signature

1 Introduction

The purpose of this lab was to get introduced with loading data from program memory, how to use the LCD display and driver, and how to right a full assembly program. In this lab you define two strings in program memory, load them into SRAM where the LCD driver goes to look for them, and then print them each on their own line on the LCD.

2 Internal Register Definitions and Constants

I didn't define any directives/typedefs for registers myself, however I used plenty of them from m128def.inc and the driver code. For instance, LCDLn1Aaddr is the address where the driver looks for the string to be written to line one of the LCD. At the bottom of the program I defined JEREMY_BEG as a .DB in program memory that contains my name, Jeremy, and directly following it JEREMY_END. JEREMY_END doesn't contain anything. Instead, it is used as the stopping address for loading the string pointed to be JEREMY_BEG since JEREMY_END will come right after JEREMY_BEG in memory. HWORLD_BEG is also a .DB in program memory which contains the string "Hello World." Just like JEREMY_BEG, HWORLD_BEG is also followed by a stopper: HWORLD_END.

3 Init()

The Init function first takes care of initializing the stack pointer. It loads the low byte of RAMEND into SPL and the high byte of RAMEND into SPH. Init then calls the LCD driver's init function, LCDInit. LCDClr is then called to blank out anything written on the display. Line one is set up first by loading the strings address into Z, the destination address where the driver looks for the string into Y, and the stopping address into R18 and R19. Then, loadString() is called. The same process is repeated for the string that goes on line two. Finally, the program jumps to main().

4 Main()

The main program simply calls the LCD driver's function to write the first line, calls the LCD driver's function to write the second line, then jumps back to main. The precondition to the main function is the strings which are desired to be output to the LCD display are loaded with line one starting at 0x0100 and line two starting at 0x0110.

5 loadString()

LOADSTRING loads the string pointed to by Z into the location pointed to by Y. A precondition to the function is that Z is properly pointing to the source, Y is properly pointing to the destination, R18 has the low byte stopping address and R19 has the high byte stopping address. The stopping address is the address which directly follows the string we want to load. For instance JEREMY_BEG is the string we want to load and JEREMY_END is the stopping address. When Z points to JEREMY_END we know the function has finished loading the string. LOADSTING runs a loop that continuesly moves the data pointed to by Z to the location pointed to by Y and post increments both Z and Y. After each iteration Z's low address is compared with R18. If it matches, then Z's high address is compared with R19. If they both match, then the stopping address has been met and the function returns.

6 Additional Questions

1. *In this lab, you were required to move data between two memory types: program memory and data memory. Explain the intended uses and key differences of these two memory types*

   Program memory will still be there if the power goes out, where data memory will disappear. The intended use of program memory is to put data there that should not be lost when there is a power outage. Data memory is intended to be the workspace that the program is using while it is running. Also, program memory is 16-bits wide in the Atmega128, where data memory is only 8-bits wide in the Atmega128.

2. *You also learned how to make function calls. Explain how making a function call works (including its connection to the stack), and explain why a RET instruction must be used to return from a function.*

   A function call pushes the address of the next instruction, PC + 1, onto the stack and places the function's address into PC so that the next instruction jumps into it. At the end of the function when RET is called, the program pops the return address off of the stack that was pushed when the function was called and places it into PC so that the next instruction jumps back to the instruction directly following the function call.

3. *To help you understand why the stack pointer is important, comment out the stack pointer initialization at the beginning of your program, and then try running the program on your mega128 board and also in the simulator. What behavior do you observe when the stack pointer is never initialized? In detail, explain what happens (or no longer happens) and why it happens.*

   Function calls no longer work as they should. When a function call is made the stack pointer pushes the address of PC + 1, however, it pushes it to the location where SP is pointing to. Without initializing SP to point to RAMEND, SP could end up overwriting data.

7 Difficulties

This lab went well. I ran into an issue where the strings weren't being printed on the LCD. I opened up the simulator and set a few breakpoints and verified that my string was in fact in the right location before main(). However, right before the call to main() I made a call to LCDClr which I thought would just wipe the LCD display to blank spaces. It did that, but it did so by clearing my strings out of memory. Once I figured that out and removed it all went well.

8 Conclusion

Overall this was a good lab. It was an interesting experience to have to direct where every piece of memory goes and dictate how the program should operate at such a low level.

9 Source Code

Jeremy_Fischer_Lab4_Sourcecode.asm

```
;***********************************************************
;*
;* Jeremy_Fischer_Lab4_sourcecode.asm
;*
;* This program interacts with the LCD to display characters
;*
;* This is the skeleton file for Lab 4 of ECE 375
;*
;***********************************************************
;*
;*  Author: Jeremy Fischer
;*    Date: 01/31/2018
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def MPR = r16 ; Multipurpose register is
; required for LCD Driver

;***********************************************************
;* Start of Code Segment
;***********************************************************
.cseg ; Beginning of code segment

;***********************************************************
;* Interrupt Vectors
;***********************************************************
.org $0000 ; Beginning of IVs
rjmp INIT ; Reset interrupt

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT: ; The initialization routine
; Initialize Stack Pointer
ldi R16, low(RAMEND)  ; Low Byte of End SRAM Address
out SPL, R16  ; Write byte to SPL
```

```
ldi R16, high(RAMEND)  ; High Byte of End SRAM Address
out SPH, R16  ; Write byte to SPH

; Initialize LCD Display
call LCDInit ;Call LCDInit to init the driver
rcall LCDClr ;Clear the LCD display

;NOW LOAD STRING 1
ldi ZL, low(JEREMY_BEG << 1)  ;Load the low byte of JEREMY_BEG
ldi ZH, high(JEREMY_BEG << 1) ;Load the high byte of JEREMY_BEG

;Load LCDLn1Addr into Y becuase this is where LCDDriver reads from
;when looking for line 1.
ldi YL, low(LCDLn1Addr)   ;Load the low byte of LCDLn1Addr
ldi YH, high(LCDLn1Addr)  ;Load the low byte of LCDLn1Addr

ldi R18, low(JEREMY_END << 1)
ldi R19, high(JEREMY_END << 1)
rcall LOADSTRING

;NOW LOAD STRING 2
ldi ZL, low(HWORLD_BEG << 1)  ;Load the low byte of HWORLD_BEG
ldi ZH, high(HWORLD_BEG << 1) ;Load the high byte of HWORLD_BEG

;Load LCDLn2Addr into Y becuase this is where LCDDriver reads from
;when looking for line 2.
ldi YL, low(LCDLn2Addr)   ;Load the low byte of LCDLn2Addr
ldi YH, high(LCDLn2Addr)  ;Load the low byte of LCDLn2Addr

ldi R18, low(HWORLD_END << 1)
ldi R19, high(HWORLD_END << 1)
rcall LOADSTRING



; NOTE that there is no RET or RJMP from INIT, this
; is because the next instruction executed is the
; first instruction of the main program

;************************************************************
;* Main Program
;************************************************************
MAIN: ; The Main program
; Display the strings on the LCD Display
rcall LCDWrLn1 ;Write the first line
rcall LCDWrLn2 ;Write the second line
```

```
rjmp MAIN ;jump back to main and create an infinite
;while loop.  Generally, every main program is an
;infinite while loop, never let the main program
;just run off

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

;-----------------------------------------------------------
; Func: LOADSTRING
; Desc: Given the 'read' address in Z and the 'write' address
; in Y, and the stopping address (low in R18 and high in R19),
; this function will copy bytes from Z address to Y address
; until the stop address is met by Z
;-----------------------------------------------------------
LOADSTRING: ; Begin a function with a label
; Save variables by pushing them to the stack
push MPR
push ZL
push ZH
push YL
push YH

;Y and Z are already set up for read/write before function

WRLINE:
LPM MPR, Z+ ;Load byte from Z (my string) and then point to next byte
ST Y+, MPR ;Store byte in SRAM then point to next slot
cp ZL, R18 ;Compare the current ZL with the ending address
breq COMPHIGH ;Low bytes match, check high bytes
rjmp WRLINE ;Low bytes don't match, not at end of Line,
;continue reading line
COMPHIGH:
cp ZH, R19 ;Compare the current ZH with the ending address
breq EXIT
rjmp WRLINE ;High bytes don't match, not at end of Line,
;continue reading line
EXIT:
pop YH
pop YL
pop ZH
pop ZL
pop MPR
ret ;return from function. String is loaded in SRAM
```

```
;***********************************************************
;* Stored Program Data
;***********************************************************

;-----------------------------------------------------------
; An example of storing a string. Note the labels before and
; after the .DB directive; these can help to access the data
;-----------------------------------------------------------
STRING_BEG: .DB "My Test String" ; Declaring data in ProgMem
STRING_END:

JEREMY_BEG: .DB  "Jeremy" ;Declaring my name in ProgMem
JEREMY_END:

HWORLD_BEG: .DB  "Hello World!" ;Declaring 'Hello World' in ProgMem
HWORLD_END:

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

LCDDriver.asm

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