ECE 375 Lab 6

External Interrupts

Lab Time: Friday 4-6

Aaron Vaughan

Bradley Heenk

TA Signature

1 Introduction

The purpose of this lab is to familiarize ourselves with the use of interrupts. We do this by implementing the basic bumpbot code from lab 1 and 2 but instead of using polling to know when to enter a subroutine, we use interrupts. The functionality of the program remains the same with the added complexity of driving the LCD screen to display the number of times that the code executed an interrupt handler function. We use the LCD screen so we need to include the drivers.

2 Program Overview

This program implements the normal bumpbot behavior with interrupts to deal with I/O. We had to research the use of interrupts for this lab. Mainly, we needed to know how to control the input hardware on our board and how to ignore all the inputs we didnt need. The buttons on our avr board are active low. For this reason, we initialize our interrupt sense control to activate at the falling edge. To ignore the unneeded inputs we use the EIMSK to mask their values. The rest is the same as our basic bumpbot behavior. The bumpbot moves forward indefinitely. If a whisker is hit, we back up and turn away from that direction and resume moving forward. There is a counter display on the LCD screen this lab. Each time an interrupt is triggered the counter on the LCD will increment. We do this to keep track of the number of times each whisker is hit. There is a function to clear the left and right whisker counter on the LCD as well.

3 Internal Register Definitions and Constants

The LCD screen uses R17 to R22 so we are limited in our use of multipurpose registers. We set up R16 as mpr R23 as waitcnt, R24 and R25 are our loop counters. We designate the right and left whisker as constant values that will be used to select the corresponding input button.

4 Interrupt Vectors

Of course we have the reset interrupt at $0000. We also use four other interrupt vectors to handle the subroutine execution: $0002 is HitLeft, $0004 is HitRight, $0006 is HitLeftClr which clears the HitLeft counter, and $0008 is the HitRightClr vector. These vectors just call the subroutine and return back to the location of main before the interrupt was initiated.

5 Program Initialization

For this lab we need the stack pointer initialized. We use it in the LCD driver file and for the subroutines to push and pop our registers to save the state of the processor. We need PORTB as an output bus for use with the LEDs and PORTD as an input bus for the push buttons. As described above, the buttons are acitive low. We set up the interrupt control registers EICRA and EICRB to activate on a falling edge. Next we set up the EIMSK to mask the pins that we do not want to monitor. We then initialize the LCD screen by simply calling the function provided by the LCDDriver.asm file. We then call a reset function to set the lcd output counters to zero. Lastly, we set the global interrupt bit.

6 Main Program

This part is simple. Just move the TekBot forward indefinitely. The interrupts account for all of the subroutine calls.

7 Subroutines

Subroutine Name: HitLeft
Description: Handles functionality of the TekBot when the right whisker is triggered.

Subroutine Name: HitRight
Description: Handles functionality of the TekBot when the right whisker

is triggered.

Function Name: Hit Right Clear
Description: Prepares our update fucntion to be set to a "0" then calls our fucntion to update the screen this is used for when we want to clear a specific button that is pressed

Function Name: Hit Left Clear
Description: Prepares our update fucntion to be set to a "0" then calls our fucntion to update the screen this is used for when we want to clear a specific button that is pressed.

Subroutine Name: Wait
Description: 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

Function Name: Queue Fix Function
Description: This fucntions counts to around 600 micro seconds this is used to help avoid queue delays since we know our Atmega128 chip runs at 16 mhz we know each clock cycle will be 1 / 16 mhz and convert it to microseconds. Now we can take 600 value and divide by our result which was used to determine how many loops of 255 clock cycles we would need.

Subroutine Name: Corner Case
Description: This function checks alternating left and right whisker pushes on our board if it alternates left then right for total of 5 times it will turn 180 degrees and move forward instead

Function Name: Im Stuck
Description: This function is called if the TekBot is stuck in a corner and turns 180 degrees to get unstuck from the corner.

Function Name: SameWhisker

Description: This function is called when the same whisker is hit repeatedly. It extends the turning and backing up routine by one second to get more clearance from an obstacle that is repeatedly in our way.

Function Name: Update Left Function
Description: This uses the value in olcnt and loads that char into the LCDscreen and updates that specific char

Function Name: Update Right Function
Description: This uses the value in ilcnt and loads that char into the LCDscreen and updates that specific char.

Function Name: Base Text Function
Description: This uses the value in ilcnt and loads that char into the LCDscreen and updates that specific char

8 Stored Program Data

We allocate some space in program memory to display a label on the LCD screen to signify which counter is being incremented. They aer named "LW: " and "RW: " for left and right whisker respectively.

9 Additional Program Includes

The LCD driver file must be included in this lab to utilize the functionality of the LCD screen. "LCDDriver.asm"

10 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.

The main benefit of the use of interrupts is the understandability of the code. The main program is much simpler. The use of interrupts limits the lines of code in main by handling the calling and returning of each of the subroutine cases. The use of the LCD screen complicated the code considerably but before this section was added, the code is very streamlined. Above all, the C-language programming is the most understandable because it is performed in a high-level programming language. The high-level tactic comes with its own set of complications though. If we are concerned with the speed and precision of each operation the C-Language may hinder our ability to control *exactly* what the processor is doing. For this reason, the risk of unforeseen bugs using a high-level coding language offers some area of concern. When considering the time to program, the polling and the interrupt coding methods were the about equal. The C-Language was fastest but I think that is because of my familiarity to that language.

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.

   Since using the timer/counterx in ctc mode would not interfere with the interrupt pins, we would be able to use it to perform the wait sequence.

11 Difficulties

Since the TA's forgot to tell us to remove the jumper J11 or J10 we got stuck on the clear function for HOURS!! We commented out the code and the errors still existed. This made no sense. We emailed one of the TAs and they told us to remove the jumper and this completely fixed our problem.

12 Conclusion

Seeing how interrupts work was interesting to us. The code was much more streamlined using this technique rather than pin polling. The polling technique was much more confusing to read and understand. We spent hours on the clear function before one of the TAs told us to remove a jumper. The interrupt INT3 and INT2 pin must have some connectivity to J10 and J11 on the board. This is an interesting fact that I will not soon forget. Implementing the basic bumpbot functionality was extremely simple. We coded, compiled, uploaded the hexfile to the avr board and it worked flawlessly. We spent the next three days trying to make the interrupt counter on the LCD work. After coding we compiled and uploaded it to the board and tested. The clear function was clearing both interrupt counters. After removing the jumper, and retesting, the code did what it was supposed to do.

We then took care of the challenge code. I sure wish we had some more registers to work with. This part was extremely challenging. Providing some real world solutions to the TekBot getting stuck was rewarding.

13 Source Code & Challenge Code

```
;***********************************************************
;*
;* Bradley_Heenk_and_Aaron_Vaughan_Lab6_challenegecode.asm
;*
;* This program uses the process or interrupts intstead of
;* polling for out BumpBots it also displays on the screen
```

```
;* how many times each of the left or right whiskers are hit
;* this displays up to a maximum of 19 which was how it was
;* designed which is more than enough for this lab. This
;* program also detects when we're int a corner or hit
;* the same whisker twice.
;*
;* This is the skeleton file for Lab 6 of ECE 375
;*
;**********************************************************
;*
;*  Author: Bradley Heenk and Aaron Vaughan
;*    Date: 11/13/2019
;*
;**********************************************************

.include "m128def.inc" ; Include definition file

;**********************************************************
;* Internal Register Definitions and Constants
;**********************************************************
.def mpr = r16 ; Multipurpose register
.def waitcnt = r23
.def ilcnt = r24
.def olcnt = r25
.def memory = r5
.def lastwsk = r2
.def leftcnt = r3
.def rightcnt = r4
.def currentwsk = r6
.def checkbit = r7

.equ WskrR = 0 ; Right Whisker Input Bit
.equ WskrL = 1 ; Left Whisker Input Bit

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

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

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

.org $0002 ; Beginning of IVs
rcall  HitRight ; Reset interrupt for HitLeft
reti

.org $0004
rcall  HitLeft ; Reset interrupt for HitRight
reti

.org $0006
rcall  HitRightClr ; Reset interrupt for HitLeftClr
reti

.org $0008
rcall  HitLeftClr ; Reset interrupt for HitRightClr
reti

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT: ; The initialization routine
; Initialize Stack Pointer
ldi mpr, low(RAMEND)
out SPL, mpr
ldi mpr, high(RAMEND)
out SPH, mpr

; Initialize Port B for out output
ldi mpr, $00
out PORTB, mpr
ldi mpr, $FF
out DDRB, mpr

; Initialize Port D for our inputs
```

```
ldi mpr, $FF
out PORTD, mpr
ldi mpr, $00
out DDRD, mpr

; Initialize external interrupts
ldi mpr, 0b10101010 ; Set the Interrupt Sense Control to falling edge
sts EICRA, mpr

ldi mpr, 0b00000000 ; Set the Interrupt Sense Control to falling edge
out EICRB, mpr

; Configure the External Interrupt Mask
ldi mpr, 0b00001111 ; Set value to what we want to hide
out EIMSK, mpr

rcall LCDInit ; Initialize LCD Display

; Get our screen intialized to zero to start with
rcall HitLeftClr
rcall HitRightClr

; Turn on interrupts
sei
; NOTE: This must be the last thing to do in the INIT function

;***********************************************************
;* Main Program
;***********************************************************
MAIN: ; The Main program

; Turns off the motors
ldi mpr,0xF0 ; Load 0xF0 into mpr to indicate stopped
out PORTB,mpr ; Store mpr into the I/O register of PORTB

; Move the bumpbot forward
ldi mpr,0x60 ; Load 0x60 into mpr to move forward
out PORTB,mpr ; Store mpr into the I/O register of PORTB

rjmp MAIN ; Create an infinite while loop to signify the
```

```
; end of the program.

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

;-----------------------------------------------------------------
; Sub: HitLeft
; Desc: Handles functionality of the TekBot when the right whisker
; is triggered.
;-----------------------------------------------------------------
HitLeft: ; Begin a function with a label

; Save variable by pushing them to the stack
push mpr
push waitcnt
in   mpr, SREG
push mpr

ldi mpr, $01 ; Load 1 into mpr
mov currentwsk, mpr ; Load mpr into the current whisker

; Setup our UpdateRight function for 0
push olcnt
mov olcnt, leftcnt ; Move our current left whisker counter to olcnt
rcall UpdateLeft ; Call our update counter function
pop olcnt

rcall CornerCase ; Call the cornercase function to check if we are stuck
sbrs checkbit,0 ; Check if the checkbit is set and if so skip rjmp LEFTSKIP
rjmp LEFTSKIP ; Relative jump to LEFTSKIP

; Execute the function here
; Preform reverse command wait 100 ms
ldi mpr,0x0
out PORTB,mpr
ldi waitcnt, 100
rcall WAITFUNC

; Preform left command wait 100 ms
```

```
ldi mpr,0x20
out PORTB,mpr
ldi waitcnt, 100
rcall WAITFUNC

; Preform forward command
ldi mpr,0x60
out PORTB,mpr

LEFTSKIP:

rcall QueueFix ; Call the QueueFix function for 600 us delay
ldi mpr, $03 ; Load $03 into mpr and have a
out EIFR, mpr

ldi mpr, $01 ; Load $01 into mpr
mov lastwsk, mpr ; Load mpr into the lastwsk

ldi mpr, $01 ; Load $01 into mpr
mov checkbit, mpr ; Load mpr into the checkbit

; Restore variable by popping them from the stack in reverse order
pop  mpr
out  SREG, mpr
pop  waitcnt
pop  mpr

ret ; End a function with RET

;----------------------------------------------------------------
; Sub: HitRight
; Desc: Handles functionality of the TekBot when the right whisker
; is triggered.
;----------------------------------------------------------------
HitRight: ; Begin a function with a label

; Save variable by pushing them to the stack
push mpr
push waitcnt
in   mpr, SREG
```

```
push mpr

ldi mpr, $02
mov currentwsk, mpr

; Setup our UpdateRight function for 0
push ilcnt
mov ilcnt, rightcnt
rcall UpdateRight ; Call the update right function
pop ilcnt

rcall CornerCase ; Call our corner case function
sbrs checkbit,0 ; Skip the next step if the bit if checkbit is set
rjmp RIGHTSKIP ; Jump to RIGTHSKIP

; Execute the function here
; Preform reverse command wait 100 ms
ldi mpr,0x0
out PORTB,mpr
ldi waitcnt, 100
rcall WAITFUNC

; Preform right command wait 100 ms
ldi mpr,0x40
out PORTB,mpr
ldi waitcnt, 100
rcall WAITFUNC

; Preform forward command
ldi mpr,0x60
out PORTB,mpr

RIGHTSKIP:

rcall QueueFix ; Call the queue function for 600 us delay
ldi mpr, $03 ; Store $03 into mpr
out EIFR, mpr ; Load mpr into the I/O of EIFR

ldi mpr, $02 ; $02 into mpr
mov lastwsk, mpr ; Load mpr nto lastwsk
```

```
ldi mpr, $01 ; Load $01 into mpr
mov checkbit, mpr ; Load 1 into checkbit

; Restore variable by popping them from the stack in reverse order
pop  mpr
out  SREG, mpr
pop  waitcnt
pop  mpr

ret ; End a function with RET

;-----------------------------------------------------------
; Func: Hit Right Clear Function
; Desc: Prepares our update fucntion to be set to a "0"
; then calls our fucntion to update the screen
; this is used for when we want to clear a specific
; button that is pressed
;-----------------------------------------------------------
HitRightClr: ; Begin a function with a label

push mpr ; Push registers onto the stack

ldi mpr, $30 ; Load the value of $30 into mpr
mov rightcnt, mpr ; Load mpr into rightcnt
ldi mpr, $00 ; Load 0 into mpr
mov lastwsk, mpr ; Set the last wsk to 0
mov currentwsk, mpr ; Set the current wsk to 0
mov memory, mpr ; Set the memory to 0

; Setup our UpdateLeft function
push ilcnt
mov ilcnt, rightcnt ; Load $30 ("0") into olcnt
rcall UpdateRight ; Call UpdateRight
pop ilcnt

rcall QueueFix ; Call the QueueFix function
ldi mpr, $03 ; Load value of 3 into mpr
out EIFR, mpr ; Store to the I/O of EIFR with mpr
```

```
pop mpr ; Pop registers onto the stack

ret ; End a function with RET

;-----------------------------------------------------------
; Func: Hit Clear Left
; Desc: Prepares our update fucntion to be set to a "0"
; then calls our fucntion to update the screen
; this is used for when we want to clear a specific
; button that is pressed
;-----------------------------------------------------------
HitLeftClr: ; Begin a function with a label

push mpr ; Push registers onto the stack

ldi mpr, $30 ; Load the value $30 into mpr
mov leftcnt, mpr ; Load leftcnt with mpr
ldi mpr, $00 ; Load the value of 0 into mpr
mov lastwsk, mpr ; Set the last wsk to 0
mov currentwsk, mpr ; Set the current wsk to 0
mov memory, mpr ; Set the memory to 0

; Setup our UpdateLeft function
push olcnt
mov olcnt, leftcnt ; Load $30 ("0") into olcnt
rcall UpdateLeft ; Call UpdateLeft
pop olcnt

rcall QueueFix ; Call the QueueFix function
ldi mpr, $03 ; Load value of 3 into mpr
out EIFR, mpr ; Store to the I/O of EIFR with mpr

pop mpr ; Pop registers off the stack

ret ; End a function with RET

;----------------------------------------------------------------
; Sub: Wait
; 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
;-----------------------------------------------------------
WAITFUNC:
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 Wait loop

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

;-----------------------------------------------------------
; Func: Queue Fix Function
; Desc: This fucntions counts to around 600 micro seconds
; this is used to help avoid queue delays since we
; know our Atmega128 chip runs at 16 mhz we know each
; clock cycle will be 1 / 16 mhz and convert it to
; microseconds. Now we can take 600 value and divide
; by out result which was used to determine how many
; loops of 255 clock cycles we would need to stack
; hence the inner loop and outer loops
;-----------------------------------------------------------
QueueFix:
push ilcnt ; Push registers onto the stack
push olcnt
ldi ilcnt, 255 ; Load 255 into ilcnt
ldi olcnt, 1 ; Load 30 into olcnt
ILOOPQUEUE:
```

```
dec ilcnt ; Decrement ilcnt
brne ILOOPQUEUE ; Branch if not equal to zero to ILOOPQUEUE
OLOOPQUEUE:
ldi ilcnt, 255 ; Load 255 into ilcnt
dec olcnt ; Decrement olcnt
brne ILOOPQUEUE ; Branch if not equal to zero to ILOOPQUEUE

pop olcnt ; Pop registers off the stack
pop ilcnt
ret

;----------------------------------------------------------------
; Sub: Corner Case
; Desc: This function checks alternating left and right whisker
; pushes on our board if it alternates left then right for
; a total of 5 times it will turn 180 degrees and move
; forward instaed
;----------------------------------------------------------------
CornerCase:

push mpr

mov mpr, lastwsk ; Move laskwsk into mpr

cpi mpr, 0 ; Compare mpr to 0
breq CORNEREXIT ; If true brance to CORNEREXIT
mov mpr, currentwsk ; move currentwsk into mpr
cpi mpr, 1 ; Compare mpr to 1
breq LEFTWHISKER ; If true branch to LEFTWHISKER
cpi mpr, 2 ; Compare mpr to 2
breq RIGHTWHISKER ; If true branch to RIGHTWHISKER
rjmp CORNEREXIT ; Sanity check in case mpr is garbage


LEFTWHISKER: ; If left whisker is hit this is executed
cp currentwsk, lastwsk ; Compare current whisker to last hisker
breq SAME ; If true we hit the same whisker and branch to SAME

mov mpr, memory ; Load our memory for our cases into mpr
andi mpr, $F0 ; And the 4 most significant bits and store into mpr
```

```
cpi mpr, 0b01000000 ; Check if the left most bits = 4
breq STUCK ; If true execute our STUCK corner case function

push olcnt ; Push olcnt onto the stack
ldi olcnt, $10 ; Load $01 into olcnt
add mpr, olcnt ; add olcnt to mpr and store into mpr

mov olcnt, memory ; Move our memory into olcnt
andi olcnt, $0F ; And olcnt with 0F getting us the right most significant bits
add mpr, olcnt ; Add olcnt with mpr and store intro
pop olcnt  ; Pop olcnt off the stack

mov memory, mpr ; move mpr into memory and replace it

rjmp CORNEREXIT ; Go to the end of our function

RIGHTWHISKER: ; If right whisker is hit this is executed
cp currentwsk, lastwsk ; Compare last whisker to current whisker
breq SAME ; If true branch to the same meaning the same whisker is hit

mov mpr, memory ; Load our memory into mpr
andi mpr, $0F ; And the 4 least significant bits and store into mpr

cpi mpr, 0b00000100 ; Compare the least signicant 4 bits
breq STUCK ; if true branch to STUCK statement

push olcnt ; Push olcnt onto the stack
ldi olcnt, $01 ; Load $01 into olcnt
add mpr, olcnt ; Add mpr with olcnt and store into mpr

mov olcnt, memory ; Move our memory into olcnt
andi olcnt, $F0 ; And the 4 most signficant bits and store into olcnt
add mpr, olcnt ; Add olcnt with mpr and store into mpr
pop olcnt  ; Pop olcnt off the stack

mov memory, mpr ; Move our new mpr value into memory

rjmp CORNEREXIT ; Go to the end of our function
```

```
SAME:
rcall SameWhisker ; We ended up in this area call SameWhisker
rjmp CORNEREXIT ; Jump to the end of our function

STUCK:
rcall ImStuck ; We ended up in this area call ImStuck
rjmp CORNEREXIT ; Jump to the end of our function

CORNEREXIT:

pop mpr ; Pop mpr off the stack

ret ; Return from subroutine

;-------------------------------------------------------------
; Func: Im Stuck
; Desc: This function preforms the flip and turn 180 degrees
; to turn around and get unstuck from the corner
;-------------------------------------------------------------
ImStuck:
ldi mpr, $00 ; Load $00 into mpr
mov memory, mpr ; Move mpr into memory to reset our memory

; Execute the function here
; Preform reverse command wait 100 ms
ldi mpr,0x0
out PORTB,mpr
ldi waitcnt, 100
rcall WAITFUNC

; Preform right command wait 100 ms
ldi mpr,0x40
out PORTB,mpr

; Wait for 4 seconds
ldi waitcnt, 200
rcall WAITFUNC
ldi waitcnt, 200
rcall WAITFUNC
```

```
; Preform forward command
ldi mpr,0x60
out PORTB,mpr

ldi mpr, $00 ; Load $00 into mpr
mov checkbit, mpr ; Move mpr into the checkbit

ret

;-----------------------------------------------------------
; Func: Same Whisker
; Desc: This fucntion calls the same whisker fucntion
; when the same whisker is hit to avoid hitting
; the same object over and over again.
;-----------------------------------------------------------
SameWhisker:
ldi mpr, $00 ; Load $00 into mpr
mov memory, mpr ; Move mpr into memory to reset our memory

; Execute the function here
; Preform reverse command wait 100 ms
ldi mpr,0x0
out PORTB,mpr
ldi waitcnt, 200
rcall WAITFUNC

; Preform right command wait 100 ms
ldi mpr,0x40
out PORTB,mpr

; Wait for 2 seconds
ldi waitcnt, 200
rcall WAITFUNC
dec ilcnt

; Preform forward command
ldi mpr,0x60
out PORTB,mpr

ldi mpr, $00 ; Load $00 into mpr
```

```
mov checkbit, mpr ; Move mpr into the checkbit

ret
;-------------------------------------------------------------
; Func: Update Left Function
; Desc: This uses the value in olcnt and loads that char
; into the LCDscreen and updates that specific char
;-------------------------------------------------------------
UpdateLeft: ; Begin a function with a label

push mpr

rcall BaseText

ldi YL, low(LCDLn1Addr) ; Load the low byte of LCDLn1Addr into YL
ldi YH, high(LCDLn1Addr) ; Load the high byte of LCDLn1Addr into YH


ldi mpr, 4 ; Load 14 into mpr this used for where on the screen we want to be
add YL, mpr ; Now add this value to YL

mov mpr, olcnt ; Copy and store olcnt into mpr
cpi mpr, $3A ; Compare mpr to $3A which is greater then 9
brge TENSLEFT ; This indicated if were above 9

st Y+, mpr ; Store mpr into Y

ldi mpr, $20 ; Load $20 into mpr, $20 => " " character
st Y, mpr ; Store this value into Y

rjmp LEFTDONE ; End the fucntion by calling RIGHTDONE

TENSLEFT:
ldi mpr, $31 ; Load a 1 into our MSB in our display
st Y+, mpr ; Store this value in our display

mov mpr, olcnt ; Copy and store olcnt into mpr
subi mpr, 10
st Y, mpr ; Store mpr into Y
```

```
LEFTDONE:

rcall LCDWrLn1 ; Call the LCDWrite function to update the display

inc leftcnt

pop mpr

ret ; End a function with RET

;------------------------------------------------------------
; Func: Update Right Function
; Desc: This uses the value in ilcnt and loads that char
; into the LCDscreen and updates that specific char
;------------------------------------------------------------
UpdateRight: ; Begin a function with a label

push mpr

rcall BaseText

ldi YL, low(LCDLn1Addr) ; Load the low byte of LCDLn1Addr into YL
ldi YH, high(LCDLn1Addr) ; Load the high byte of LCDLn1Addr into YH

ldi mpr, 14 ; Load 14 into mpr this used for where on the screen we want to be
add YL, mpr ; Now add this value to YL

mov mpr, ilcnt ; Copy and store ilcnt into mpr
cpi mpr, $3A ; Compare mpr to $3A which is greater then 9
brge TENSRIGHT ; This indicated if were above 9

st Y+, mpr ; Store mpr into Y

ldi mpr, $20 ; Load $20 into mpr, $20 => " " character
st Y, mpr ; Store this value into Y

rjmp RIGHTDONE ; End the fucntion by calling RIGHTDONE

TENSRIGHT:
ldi mpr, $31 ; Load a 1 into our MSB in our display
```

```
st Y+, mpr ; Store this value in our display

mov mpr, ilcnt ; Copy and store ilcnt into mpr
subi mpr, 10
st Y, mpr ; Store mpr into Y

RIGHTDONE:

rcall LCDWrLn1 ; Call the LCDWrite function to update the display

inc rightcnt ; Increment rightcnt

pop mpr

ret ; End a function with RET


;-----------------------------------------------------------
; Func: Base Text Function
; Desc: This uses the value in ilcnt and loads that char
; into the LCDscreen and updates that specific char
;-----------------------------------------------------------
BaseText: ; Begin a function with a label

push mpr

ldi YL, low(LCDLn1Addr) ; Load the low byte of LCDLn1Addr into YL
ldi YH, high(LCDLn1Addr) ; Load the high byte of LCDLn1Addr into YH
ldi ZL, low(LW_BEG<<1)
ldi ZH, high(LW_BEG<<1)

lpm mpr, Z+ ; Load program memory from where Z points into mpr
st Y+, mpr ; Store mpr into Y and post inc
lpm mpr, Z+ ; Load program memory from where Z points into mpr
st Y+, mpr ; Store mpr into Y and post inc
lpm mpr, Z+ ; Load program memory from where Z points into mpr
st Y+, mpr ; Store mpr into Y and post inc
lpm mpr, Z+ ; Load program memory from where Z points into mpr
st Y+, mpr ; Store mpr into Y and post inc
```

```
ldi mpr, 6
add YL, mpr

ldi ZL, low(RW_BEG<<1)
ldi ZH, high(RW_BEG<<1)

lpm mpr, Z+ ; Load program memory from where Z points into mpr
st Y+, mpr ; Store mpr into Y and post inc
lpm mpr, Z+ ; Load program memory from where Z points into mpr
st Y+, mpr ; Store mpr into Y and post inc
lpm mpr, Z+ ; Load program memory from where Z points into mpr
st Y+, mpr ; Store mpr into Y and post inc
lpm mpr, Z+ ; Load program memory from where Z points into mpr
st Y+, mpr ; Store mpr into Y and post inc

pop mpr

ret ; End a function with RET

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

LW_BEG:
.DB "LW: " ; Declaring data in ProgMem
LW_END:

RW_BEG:
.DB "RW: " ; Declaring data in ProgMem
RW_END:

; Enter any stored data you might need here

;***********************************************************
;* Additional Program Includes
;***********************************************************
; There are no additional file includes for this program
.include "LCDDriver.asm" ; Include the LCD Driver
```