ECE 375 LAB 6

External Interrupts

Lab Time: Wednesday 12-2

Bradley Martin

INTRODUCTION

The Purpose of this lab is to utilize Interrupt Vectors to take input for the BumpBot. The operation of this lab is similar to lab 1 and lab 2 in addition to using the LCD screen. A pre-made skeleton code is provided to get started.

PROGRAM OVERVIEW

The Program provides the basic behavior for how the TekBot will react to whisker input and keep track of the input to be displayed on an LCD screen. There are two buttons for the left and right whisker that will trigger an interrupt. If the right whisker is hit then the TekBot will increase the counter for the right whisker, backup and turn left, then continue forward. If the left whisker is hit, then the TekBot will increase the counter for left whisker, backup and turn right, then continue forward.

Other than the INIT and MAIN routines in the program, 6 additional routines were created and used. The HitRight and HitLeft provide the basic functionality for handling input from the right and left whiskers. The ClrRight and ClrLeft routines are used to reset variables and flags. The wait routine was used to give the TekBot time to backup and turn. Last is the UpdateLCD which is used to print the variables of the counters to the LCD screen.

INITIALIZATION ROUTINE

The initialization routine provides a one-time initialization of key registers that allow the program to execute correctly. First the Stack Pointer is initialized, allowing the proper use of function and subroutine calls. The right and left counters are set to 0 and the LCD is Initialized. Port B is initialized to all outputs and will be used to direct the motors. Port D was initialized to inputs and will receive the whisker input. Finally, the Move Forward command was sent to Port B to get the TekBot moving forward. Last, EICRA is set to trigger on the falling edge and EIMSK is set for the first 4 interrupts.

MAIN ROUTINE

The main routine simply tells the TekBot to move forward. The Tekbot will continue to move forward until and interrupt is called.

HITRIGHT ROUTINE

The HitRight routine first increases the count on the RightCount variable. Then moves the TekBot backwards for roughly 1 second by first sending the Move Backwards command to PORTB followed by a call to the Wait routine. Upon returning from the Wait routine, the Turn Left command is sent to PORTB to get the TekBot to turn left and then another call to the Wait routine to have the TekBot turn left for roughly another second. The HitRight Routine sends a Move Forward command to PORTB to get the TekBot moving forward. The routine clears EIFR and then returns from the routine.

HITLEFT ROUTINE

The HitLeft routine first increases the count on the LeftCount variable. Then moves the TekBot backwards for roughly 1 second by first sending the Move Backwards command to PORTB followed by a call to the Wait routine. Upon returning from the Wait routine, the Turn right command is sent to PORTB to get the TekBot to turn right

and then another call to the Wait routine to have the TekBot turn right for roughly another second. The HitLeft Routine sends a Move Forward command to PORTB to get the TekBot moving forward. The routine clears EIFR and then returns from the routine.

CLRRIGHT ROUTINE

The ClrRight routine clears the right counter variable and resets EIFR.

CLRLEFT ROUTINE

The ClrLeft routine clears the left counter variable and resets EIFR.

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.

UPDATELCD ROUTINE

The UpdateLCD routine first clears the screen. Then it initializes the X register and the right counter to enter the Bin2ASCII function. The now converted counter string is copied over to the Y register which holds the address for the first line of the LCD screen. The same process is done except for the left counter and the second line of the LCD screen. The last function writes all the values to the LCD screen.

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 polling method is only useful if the only thing you want the microcontroller to do is check the status of its ports. Interrupt is much more efficient in that respect because it will only check the status if an interrupt has been called. The rest of the time the CPU is free to run other programs. Polling can also take quite a bit of time and resources from the CPU because it must check every single port regardless if it has changed or not. However, is terms of understandability and ease of implementation, polling wins that one. If a program is not commented well it would take away to figure out what an interrupt program is performing.

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.

It is technically possible to use a timer/counter to perform the one-second delay. It really depends on what kind of percent error you want with your counters. Interrupt priority can influence when the timer/counter increments, but I was thinking more along the lines of the clock speed fluctuating due to noise/heat. Adding an extra layer like something that counts every so many clock cycles could give an overall more accurate time. This would be a combination of the two. You would lose some precision however you would gain reliability and for most programmers that is what is more important.

CONCLUSION

In this lab we were required create a program that allowed the TekBot to handle input using Interrupts. The number of times a whisker was hit was also required to be display on the LCD screen. This lab showed there are many ways to tackle a problem since this is the third time, we are coding a program that has the same result. It was hard at times to understand this new way of programming with interrupts, but overall I think it's a useful tool to have when we need it.

SOURCE CODE

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

;*

;*       Author: Bradley Martin

;*         Date: 11/10/2020

;*

;***********************************************************


.include "m128def.inc"                      ; Include definition file


;***********************************************************

;* Variable and Constant Declarations

;***********************************************************

.def    mpr = r16                           ; Multi-Purpose Register

.def    waitcnt = r23                       ; Wait Loop Counter

.def    ilcnt = r24                         ; Inner Loop Counter

.def    olcnt = r25                         ; Outer Loop Counter

.def    RightCount = r3                     ; Right whisker counter

.def    LeftCount = r4                ; Left whisker counter

.def    temp = r2                     ; Holds the number of characters for Bin2ASCII
```

```
.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

.equ    Con1 = $0120                ; Beginning address of counter string
.equ    Con2 = $0130                ; Beginning address of counter string

;/////////////////////////////////////////////////////////////
;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
                rcall HitRight          ; Run HitRight function
                reti

.org    $0004
                rcall HitLeft           ; Run HitLeft Function
                reti

.org    $0006
                rcall ClrRight          ; Run ClrRight Function
                reti

.org    $0008
                rcall ClrLeft           ; Run ClrLeft Function
                reti

.org    $0046                           ; End of Interrupt Vectors
;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:
    ; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
                ldi             mpr, low(RAMEND)
                out             SPL, mpr                ; Load SPL with low byte of RAMEND
                ldi             mpr, high(RAMEND)
                out             SPH, mpr                ; Load SPH with high byte of RAMEND

        ; Initialize counters for right and left whisker
                clr             LeftCount
                clr             RightCount
```

```
        ; Initialize LCD Display
                rcall LCDInit
                rcall UpdateLCD

    ; Initialize Port B for output
                ldi             mpr, $FF              ; 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, $00              ; Set Port D Data Direction Register
                out             DDRD, mpr             ; for input
                ldi             mpr, $FF              ; Initialize Port D Data Register
                out             PORTD, mpr            ; so all Port D inputs are Tri-State

        ; Initialize external interrupts:

        ; Set the Interrupt Sense Control to falling edge
                ldi             mpr, (1<<ISC01) | (0<<ISC00) | (1<<ISC11)| (0<<ISC10)
                sts             EICRA, mpr
        ; Set the External Interrupt Mask
                ldi             mpr, (1<<INT0) | (1<<INT1) | (1<<INT2) | (1<<INT3)
                out             EIMSK, mpr
        ; Turn on interrupts

        sei
;-----------------------------------------------------------------
; Main Program
;-----------------------------------------------------------------
MAIN:
                ; Move Robot Forward
```

```
                ldi             mpr, MovFwd                     ; Load FWD command
                out             PORTB, mpr                      ; Send to motors

                rjmp    MAIN                            ; Infinite loop. End of the program

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

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

                inc             RightCount                      ; Increment by 1
                rcall   UpdateLCD                       ; Update display

                ldi             mpr, 0b11111111                 ; Set mpr to all high
                out             EIFR, mpr                       ; Reset EIFR with all high to clear
interrupts

                ; Move Backwards for a second
                ldi             mpr, MovBck                     ; Load Move Backward command
                out             PORTB, mpr                      ; Send command to port
                ldi             waitcnt, WTime          ; Wait for 1 second
                rcall   Wait1                           ; 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          ; Wait for 1 second
                rcall   Wait1                           ; Call wait function

                ; Move Forward again
                ldi             mpr, MovFwd                     ; Load Move Forward command
                out             PORTB, mpr                      ; Send command to port

                ldi             mpr, 0b11111111                 ; Set mpr to all high
                out             EIFR, mpr                       ; Reset EIFR with all high to clear
interrupts

                sei
                ret                             ; Return from subroutine

;-----------------------------------------------------------------
; Sub:  HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
;               is triggered.
;-----------------------------------------------------------------
HitLeft:

                inc             LeftCount                       ; Increment by 1
                rcall   UpdateLCD                       ; Update display

                ldi             mpr, 0b11111111                 ; Set mpr to all high
                out             EIFR, mpr                       ; Reset EIFR with all high to clear
interrupts

                ; Move Backwards for a second
                ldi             mpr, MovBck                     ; Load Move Backward command
                out             PORTB, mpr                      ; Send command to port
                ldi             waitcnt, WTime          ; Wait for 1 second
                rcall   Wait1                           ; 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          ; Wait for 1 second
                rcall   Wait1                           ; Call wait function

                ; Move Forward again
                ldi             mpr, MovFwd                     ; Load Move Forward command
                out             PORTB, mpr                      ; Send command to port

                ldi             mpr, 0b11111111                 ; Set mpr to all high
                out             EIFR, mpr                       ; Reset EIFR with all high to clear
interrupts

                sei
                ret                             ; Return from subroutine

;-----------------------------------------------------------------
; 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
;-----------------------------------------------------------------
Wait1:
                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 subrou

;-----------------------------------------------------------
; Func: Clrright
; Desc: Clears the right counter and reprints the LCD
;-----------------------------------------------------------
ClrRight:
                clr             RightCount                      ; Set RightCount to 0

                rcall   UpdateLCD                       ; Update the LCD

                ldi             mpr, 0b11111111                 ; Set mpr to all high
                out             EIFR, mpr                       ; Reset EIFR with all high to clear
interrupts

                sei
                ret

;-----------------------------------------------------------
; Func: Clrleft
; Desc: Clears the left counter and reprints the LCD
```

```
;-----------------------------------------------------------

ClrLeft:

                clr             LeftCount                       ; Set LeftCount to 0


                rcall   UpdateLCD                       ; Update the LCD


                ldi             mpr, 0b11111111                 ; Set mpr to all high

                out             EIFR, mpr                       ; Reset EIFR with all high to clear
interrupts


                sei

                ret

;-----------------------------------------------------------

; Func: UpdateLCD

; Desc: Loads string 1 to the top line of the LCD display and

;               loads string 2 to the bottom line of the LCD display.

;-----------------------------------------------------------


UpdateLCD:


                rcall LCDClr                                            ; Clear LCD


                ldi XL,         low(Con1)                               ; Initialize the X pointer to
hold the address of the converted left counter

                ldi XH,         high(Con1)


                mov mpr,        RightCount                              ; Move the value of the Right
Counter to mpr

                rcall Bin2ASCII                                                 ; Convert the count to
a string


                mov mpr, r18                                            ;  Get  how  many  characters
long the string is

                mov temp, mpr                                           ; Move that number to temp
```

```
                ldi YL,         low(LCDLn1Addr)                 ; Initialize the Y pointer to
hold our string on line 1

                ldi YH,         high(LCDLn1Addr)


L1:             ld mpr, X+                                      ;  Load  the  strings
from X registers

                st Y+, mpr                                      ;  store  the  contents
of the string to the Y register shift by one

                dec     temp                                    ; Decrease the counter
for the loop to tell how many bits are left

                brne L1                                         ;  If  L1  has  not
decremented the counter to 0 then keep looping


                ldi XL,         low(Con2)                       ; Initialize the X pointer to
hold the address of the converted right counter

                ldi XH,         high(con2)


                mov mpr,        LeftCount                       ; Move the value of LeftCount
to mpr

                rcall Bin2ASCII                                 ; Convert the count to
a string


                mov mpr, r18                                    ;  Get  how  many  characters
long the string is

                mov temp, mpr                                   ; Move that number to temp




                ldi YL,         low(LCDLn2Addr)                 ; Initialize the Y pointer to
hold our string on line 2

                ldi YH,         high(LCDLn2Addr);


L2:             ld mpr, X+                                      ;  Load  the  strings
from X registers

                st Y+, mpr                                      ;  store  the  contents
of the string to the Y register shift by one

                dec temp                                        ; Decrease the counter
for the loop to tell how many bits are left
```

```
                brne L2                                                 ;   If   L1   has   not
decremented the counter to 0 then keep looping


                rcall LCDWrite                                      ; Update the LCD display


                ret                                             ; End a function with RET


;***********************************************************
;*      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
;-----------------------------------------------------------



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