# ECE 375 Lab 4

Interrupts

Lab Time: Tuesday 10-12

Miles Van de Wetering
Mathew Popowski

TA Signature

## 1 Introduction

In this lab, we learned to work with the interrupt vectors. We created an interrupt detecting a rising edge on the left and right whiskers which triggered turns.

## 2 Internal Register Definitions and Constants

WskrR, WskrL, EngEnR, EngEnL, EngDirR, and EngDirL were constants used to address specific bits in order to more easily control the motors. WTime and BWTime were constants used to calculate wait loop time, and waitcnt, ilcnt, and olcnt were loop counters. mpr was a general use variable.

## 3 Program Initialization

To initialize our program we directed the stack pointer to the end of the program memory section, allowing us to push and pop from the stack as we change scopes (we did very little of this). We then set up Port B and D to listen for input, and started the tekbot moving forward. Lastly, we set up interrupts for the rising edges to turn left and right.

## 4 Main Program

There was effectively no main (one line infinite loop).

## 5 HitRight

This was the function called when the right whisker was hit. It first clears interrupts to prevent nested interrupts, then backs up for one second, turns the bot left for one second, then starts moving forward again. Lastly, it sets the interrupts back up.

## 6 HitLeft

This was the function called when the left whisker was hit. It first clears interrupts to prevent nested interrupts, then backs up for one second, turns the bot right for one second, then starts moving forward again. Lastly, it sets the interrupts back up.

## 7 Interrupt Vectors

As described in the initialization process, we created two interrupt vectors. The first and second external interrupt vectors were set up, one for hit right and one for hit left.

## 8 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 primary advantage of using assembly over C is that we have much more fine-grained control over the program that is produced. Furthermore, the compiled assembly code from C generally is much larger than the regular assembly that we would write manually, leading to a minor inefficiency. The advantage of course of using C is access to higher level abstractions unavailable in the assembly programming language, and the fact that many programmers are simply more used to programming in C.

   The advantage of using polling over interrupts is that they are somewhat easier to set up (in our opinion) and they can be faster in some circumstances. Most of the time however, they cause unnecessary burden on the processor because they are constantly checking for 'hits' which occur only a small fraction of the time. Polling is a fine choice when hits are frequent and regular. Interrupts, on the other hand, don't create this same inefficiency because work is only done when an interrupt occurs. They are best used when hits are irregular or infrequent.

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 seems possible to use interrupts to perform this sort of timing operation, made much more complicated by the fact that this would be a nested interrupt. We'd probably need to set the mask at the beginning of every interrupt function, or make careful use of interrupt priority in order to ensure that the timing doesn't get interrupted by an inappropriate signal.

## 9 Difficulties

We didn't have any major difficulties with this lab.

## 10 Conclusion

This was a pretty interesting lab, a little harder to comprehend this code than usual since thinking about bits while writing vectors in hexadecimal is sometimes hard. Interrupts certainly seem better for this application than polling, and are useful in general.

## 11 Source Code

Base Code:

```
;***********************************************************
;*
;* Lab 6
;*
;* Interrupts
;*
;***********************************************************
;*
;*  Author: Mathew & Miles
;*    Date: 11/7/2016
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multipurpose register

.def waitcnt = r17 ; Wait Loop Counter
.def ilcnt = r18 ; Inner Loop Counter
.def olcnt = r19 ; Outer Loop Counter

.equ WTime = 100 ; Time to wait in wait loop
.equ BWTime = 200 ; Time to wait in reverse

.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
```

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

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

; Set up interrupt vectors for any interrupts being used
.org $0002
rcall HitRight
reti

.org $0004
rcall HitLeft
reti
; This is just an example:
;.org $002E ; Analog Comparator IV
; rcall HandleAC ; Call function to handle interrupt
; reti ; Return from interrupt

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT: ; The initialization routine
; 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 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 TekBot Forward Movement
ldi mpr, MovFwd ; Load Move Forward Command
out PORTB, mpr ; Send command to motors
; Initialize external interrupts

; Set the Interrupt Sense Control to falling edge
ldi mpr, $0A
sts EICRA, mpr
; Configure the External Interrupt Mask
ldi mpr, $03
out EIMSK, mpr
; Turn on interrupts
; NOTE: This must be the last thing to do in the INIT function
sei

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


rjmp MAIN ; Create an infinite while loop to signify the
; end of the program.

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

;-----------------------------------------------------------
; You will probably want several functions, one to handle the
; left whisker interrupt, one to handle the right whisker
; interrupt, and maybe a wait function
;------------------------------------------------------------

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
; beginning of your functions
;-----------------------------------------------------------
HitRight:
cli
push mpr ; Save mpr register
push waitcnt ; Save wait register
```

```
in mpr, SREG ; Save program state
push mpr ;

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

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

pop mpr ; Restore program state
out SREG, mpr ;
pop waitcnt ; Restore wait register
ldi mpr, $ff
out EIFR, mpr
pop mpr ; Restore mpr
sei
ret ; Return from subroutine

;-----------------------------------------------------------------
; Sub: HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
; is triggered.
;-----------------------------------------------------------------
HitLeft:
cli
push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;

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

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

pop mpr ; Restore program state
out SREG, mpr ;
pop waitcnt ; Restore wait register
ldi mpr, $ff
out EIFR, mpr
pop mpr ; Restore
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
;----------------------------------------------------------------
Wait:
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 ; End a function with RET
```