ECE 375 LAB 6

External Interrupts

Lab Time: Friday 16:00 ~ 17:50

Hyunjae Kim

INTRODUCTION

The purpose of this lab is to implement the Tekbot to behave same as the lab#1 by using external interrupts. The provided “BumpBot” program in the first lab is allowed to use in this lab, but the program has to use interrupts instead of polling as provided “BumpBot” program did. The updated “BumpBot” program will be tested in AVR board by using Universal Programmer and Atmel Studio.

PROGRAM OVERVIEW

The BumpBot keeps move forward if there is no external interrupt. If there is a interrupt, the program goes to subroutine. After completing subroutine, the program returns to the main routine. Moreover, if there are multiple interrupts, the program completes the multiple interrupts by following the priority of interrupts, and then returns to the main routine.

INITIALIZATION ROUTINE

The program first initializes the Stack Pointer to point the end of the memory to allow the return instructions. Then the program initializes the external interrupts, INT3:0, and mask out the rest of the external interrupts. After initializing the interrupts, the program initializes Port B as outputs connected to the motors, and Port D as inputs to receive the whisker inputs. Then, the program sends Move Forward command to Port B, and then initializes the right and left interrupt counter, and LCD display.

MAIN ROUTINE

The program keeps sends Move Forward Command to the Port B.

(When there is an external interrupt, the program jumps to subroutine.)

SUBROUTINES

1. HitRight Routine

The TekBot backwards for around 1 seconds by sending Move Backwards command to PORT B and wait for 1 seconds, and then turns left for around 1 second by sending Turn Left command to PORT B and wait for 1 second. Then the TekBot moves forward by sending Move Forward command to PORT B, and increment the counter.

2. HitLeft Routine

This routine is almost same as HitRight Routine. The difference is that this routine sends Turn Right command to PORT B.

3. Wait Routine

This subroutine enables other subroutines to wait for around 1 second. The subroutine uses triple-nested loops to roughly calculate 1 second by using 16MHz clock in AVR Board. First, the program does the calculation 16+159975*wait count to roughly equal to 1 second. After completing the loops, the subroutine will roughly match 1 second for other subroutines.

STUDY 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.

Polling:

The benefit of polling is that the programmer has everything under control. The programmer can understand every event in the program, and much easier to write a code so that the time consumption for writing code will be decrease.

Interrupts:

The benefit of interruption is that the program is fast and efficient. Since the program does not use any busy-waiting, the amount of program will be small, and the program runs faster than the polling because the program immediately jumps to the subroutine whenever the unexpected event happens.

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 not possible to use a timer/counter interrupt to perform with external interrupts. The reason is that the interrupt priority of external interrupts is much higher than the interrupt priority of timer/counter interrupts. Thus, because of interrupt priority, there is no way to interrupt 'external interrupts' by using 'timer/count interrupts'.

DIFFICULTIES

In clearing left and right interrupt counters, there was a challenge. When the counters count the number over 10, then writing 0 to the memory does not effectively clear all of two digits in LCD display. The challenge was connected and grew bigger. For instance, how the numbers have to be cleared if the counter counts over 100, 1000, or more?

By changing the view of clearing counters, the challenge was solved. The solution was that clear the LCD display and writing 0 on the display. Through this method, the challenge was resolved, and the program worked same as the expectation.

CONCLUSION

By using external interrupts for BumpBot, the code of the program is much more efficient than before. From learning the interrupts, the aspect of designing robot is much more developed and grown. Thus, through this lab activity, the viewpoint of dealing events is expanded than before of this lab activity.

SOURCE CODE

```
;***********************************************************
;*
;*      BasicBumpBot.asm -       V2.0
;*
;*      This program contains the neccessary code to enable the
;*      the TekBot to behave in the traditional BumpBot fashion.
;*     It is written to work with the latest TekBots platform.
;*      If you have an earlier version you may need to modify
;*      your code appropriately.
;*
;*      The behavior is very simple.  Get the TekBot moving
;*      forward and poll for whisker inputs.  If the right
;*      whisker is activated, the TekBot backs up for a second,
;*      turns left for a second, and then moves forward again.
;*      If the left whisker is activated, the TekBot backs up
;*      for a second, turns right for a second, and then
;*      continues forward.
;*
;***********************************************************
;*
;*       Author: David Zier and Mohammed Sinky (modification Jan 8, 2009)
;*         Date: January 8, 2009
;*      Company: TekBots(TM), Oregon State University - EECS
;*      Version: 2.0
;*
;***********************************************************
;*      Rev     Date    Name            Description
;*---------------------------------------------------------
;*      -       3/29/02 Zier            Initial Creation of Version 1.0
;*      -       1/08/09 Sinky           Version 2.0 modifictions
;*
;***********************************************************

.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    rcnt = r14              ;Right Whisker hit counter
.def    lcnt = r15              ; Left whisker hit counter
.def    Status = r12    ; Status register (Not SREG in AVR!!)
.def    altcnt = r11    ; Alternative whisker hit counter

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

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

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

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

```
.equ     Halt = (1<<EngEnR|1<<EngEnL)              ; Halt Command

;***********************************************************
; NOTE: Let me explain what the macros above are doing.
; Every macro is executing in the pre-compiler stage before
; the rest of the code is compiled.  The macros used are
; left shift bits (<<) and logical or (|).  Here is how it
; works:
;       Step 1.  .equ    MovFwd = (1<<EngDirR|1<<EngDirL)
;       Step 2.          substitute constants
;                         .equ    MovFwd = (1<<5|1<<6)
;       Step 3.          calculate shifts
;                         .equ    MovFwd = (b00100000|b01000000)
;       Step 4.          calculate logical or
;                         .equ    MovFwd = b01100000
; Thus MovFwd has a constant value of b01100000 or $60 and any
; instance of MovFwd within the code will be replaced with $60
; before the code is compiled.  So why did I do it this way
; instead of explicitly specifying MovFwd = $60?  Because, if
; I wanted to put the Left and Right Direction Bits on different
; pin allocations, all I have to do is change thier individual
; constants, instead of recalculating the new command and
; everything else just falls in place.
;***********************************************************

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

;-----------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------
.org     $0000                              ; Reset and Power On Interrupt
                rjmp     INIT               ; Jump to program initialization
.org     $0002
                rcall HitRight
                reti
.org     $0004
                rcall HitLeft
                reti
.org     $0006
                rcall ClearRctr
                reti
.org     $0008
                rcall ClearLctr
                ;rcall AltHit
                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
        rcall   LCDInit

        ; Set interrupts
                ldi              mpr, $AA ;int 0 int 1, int2, int3 in falling edge --> 1010  1010
                sts              EICRA, mpr
                ldi              mpr, $0f   ; 0000 1111
                out              EIMSK, mpr

   ; 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

                ldi             mpr, $00
                out             DDRE, mpr
                ldi             mpr, $ff
                out             PORTE, mpr

                clr             Status
                clr             altcnt
                ; Initialize TekBot Forward Movement
                ldi             mpr, MovFwd             ; Load Move Forward Command
                out             PORTB, mpr              ; Send command to motors
                ;Initialize Right/Left Counter
                ldi             XL, $00
                ldi             XH, $01
                ldi             mpr, 0
                mov             rcnt, mpr
                st              X, rcnt
                rcall   Bin2ASCII

                ldi             XL, $10
                ldi             XH, $01
                ldi             mpr, 0
                mov             lcnt, mpr
                st              X, lcnt
                rcall   Bin2ASCII
                sei
;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:
                ldi             mpr, MovFwd             ; Load Move Forward Command
                out             PORTB, mpr              ; Send command to motors
                rcall   LCDWrite
                rjmp    MAIN                    ; Continue through main

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

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



                ; 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   Wait2                   ; 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   Wait2                   ; Call wait function

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

```
                mov             mpr, rcnt  ;Use mpr to increment rcnt because rcnt is lower 16
regs in GPR.
                inc             mpr
                mov             rcnt, mpr
                ldi             XL, $00
                ldi             XH, $01
                st              X, rcnt
                rcall   Bin2ASCII ; Write in ASCII on LCD display

                ldi             mpr, 1
                add             Status, mpr ;Update Status reg that right whisker is once hit

                ;ldi            mpr, $0f
                ;sts            EIFR, mpr


Exit3:
                ldi mpr, $0f
                out EIFR, mpr
                ret                             ; Return from subroutine

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


                ; 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   Wait2                   ; 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   Wait2                   ; Call wait function

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



                mov             mpr, lcnt       ; use mpr becuase lcnt is lower 16 regs in GPR
                inc             mpr
                mov             lcnt, mpr
                ldi             XL, $10
                ldi             XH, $01
                st              X, lcnt
                rcall   Bin2ASCII ; Write it in ASCII on LCD display



Exit2:
                ldi             mpr, $0f
                out             EIFR, mpr
                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
;-----------------------------------------------------------------
Wait2:
                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
ClearRctr:
                rcall   LCDClrLn1
                ldi             XL, $00
                ldi             XH, $01
                ldi             mpr, 0
                mov             rcnt, mpr
                st              X, rcnt
                rcall   Bin2ASCII

                ldi             mpr, $0f
                sts             EIFR, mpr
                ret
ClearLctr:
                rcall   LCDClrLn2
                ldi             XL, $10
                ldi             XH, $01
                ldi             mpr, 0
                mov             lcnt, mpr
                st              X, lcnt
                rcall   Bin2ASCII
                ldi             mpr, $0f
                sts             EIFR, mpr
                ret



.include "LCDDriver.asm"
```