ECE 375 LAB 1

Introduction to AVR Development Tools

Lab Time: Tuesday 12-1:50

Joseph Noonan

Matthew Levis

INTRODUCTION

The lab write-up should be done in the style of a professional report/white paper. Proper headers need to be used and written in a clean, professional style. Proof read the report to eliminate both grammatical errors and spelling. The introduction should be a short 1-2 paragraph section discussing what the purpose of this lab is. This is not merely a copy from the lab handout, but rather your own personal opinion about what the object of the lab is and why you are doing it. Basically, consider the objectives for the lab and what you learned and then briefly summarize them. For example, a good introduction to lab 1 may be as follows.

The purpose of this first lab is to provide an introduction on how to use AVRStudio4 software for this course along with connecting the AVR board to the TekBot base. A simple pre-made "BumpBot" program was provided to practice creating a project in AVRStudio4, building the project, and then using the Universal Programmer to download the program onto the AVR board.

PROGRAM OVERVIEW

This section provides an overview of how the assembly program works. Take the time to write this section in a clear and concise manner. You do not have to go into so much detail that you are simply repeating the comments that are within your program, but simply provide an overview of all the major components within your program along with how each of the components work. Discuss each of your functions and subroutines, interesting program features such as data structures, program flows, and variables, and try to avoid nitty-gritty details. For example, simple state that you "First initialized the stack pointer," rather than explaining that you wrote such and such data values to each register. These types of details should be easily found within your source code. Also, do not hesitate to include figures when needed. As they say, a picture is worth a thousand words, and in technical writing, this couldn't be truer. You may spend 2 pages explaining a function which could have been better explained through a simple program-flow chart. As an example, the remainder of this section will provide an overview for the basic BumpBot behavior.

The BumpBot program provides the basic behavior that allows the TekBot to react to whisker input. The TekBot has two forward facing buttons, or whiskers, a left and a right whisker. By default the TekBot will be moving forward until one of the whiskers are triggered. If the left whisker is hit, then the TekBot will backup and then turn right for a bit, while a right whisker hit will backup and turn left. After the either whisker routine completes, the TekBot resumes its forward motion.

Besides the standard INIT and MAIN routines within the program, three additional routines were created and used. The HitRight and HitLeft routines provide the basic functionality for handling either a Right or Left whisker hit, respectively. Additionally a Wait routine was created to provide an extremely accurate busy wait, allowing time for the TekBot to backup and turn.

INITIALIZATION ROUTINE

The initialization routine provides a one-time initialization of key registers that allow the BumpBot program to execute correctly. First the Stack Pointer is initialized, allowing the proper use of function and subroutine calls. Port B was 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.

MAIN ROUTINE

The Main routine executes a simple polling loop that checks to see if a whisker was hit. This is accomplished by first reading 8-bits of data from PINE and masking the data for just the left and right whisker bits. This data is checked to see if the right whisker is hit and if so, then it calls the HitRight routine. The Main routine then checks to see if the left whisker is hit and if so, then it calls the HitLeft routine. Finally a jump command is called to move the program back to the beginning of the Main Routine to repeat the process.

HITRIGHT ROUTINE

The HitRight routine first 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. Finally, the HitRight Routine sends a Move Forward command to PORTB to get the TekBot moving forward and then returns from the routine.

HITLEFT ROUTINE

The HitLeft routine is identical to the HitRight routine, except that a Turn Right command is sent to PORTB instead. This then fills the requirement for the basic BumpBot behavior.

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.

ADDITIONAL QUESTIONS

Almost all of the labs will have additional questions. Use this section to both restate and then answer the questions. Failure to provide this section when there are additional questions will result in no points for the questions. Note that if there are no Additional Questions, this section can be eliminated. Since the original lab does not have any questions, I will make some up to illustrate the proper formatting.

1) What specific font is used for source code, and at what size?

   The source code is written in a font called, “Courier New,” and is size 8 font.

2) What is the naming convention for source code (asm)? What is the naming convention for source code files if you are working with your partner?

   The naming convention for the source code for the lab is “First name_Last name_Lab#_sourcecode.asm”. If it was the challenge code for the lab then I would add that the code was a challenge code for the lab. Example of naming convention: Joseph_Noonan_Lab_1_sourcecode.asm. If I was working with my lab partner the naming convention is “First name_Last name_and_First name_Last name_Lab#_sourcecode.asm”. Example: Joseph_Noonan_and_Matthew_Levis_Lab1_sourcecode.asm

3) Define pre-compiler directive. What is the difference between the .def and .equ directives?

   Pre-complier directives is machine code that runs before the source code is compiled. The difference between the .def and .equ directives is that .def will define a symbolic name on a register and .equ will assign a value to a symbol.

4) Determine the 8-bit binary value that each of the following expressions evaluates to:
   a. 00001000
   b. 00001000
   c. 00000100
   d. 00000001
   e. 01000011

5) Describe the instructions listed below:
   a. ADIW: Adds an immediate value to a register pair and places the result in the register pair.
   b. BCLR: Clears a single flag in SREG
   c. BRCC: Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is cleared.
   d. BRGE: Conditional relative branch. Tests the Signed flag (S) and branches relatively to PC if S is cleared.
   e. COM: This instruction performs a one's complement of register Rd.
   f. EOR: Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.
   g. LSL: Shifts all bits in Rd one place to the left. Bit 0 is cleared.
   h. LSR: Shifts all bits in Rd one place to the right. Bit 7 is cleared.
   i. NEG: Replaces the contents of register Rd with its two's complement; the value $80 is left unchanged.
   j. OR: Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register Rd.
   k. ORI: Performs the logical OR between the contents of register Rd and a constant and places the result in the destination register Rd.
   l. ROL: Shifts all bits in Rd one place to the left.
   m. ROR: Shifts all bits in Rd one place to the right.
   n. SBC: Subtracts two registers and subtracts with the C flag and places the result in the destination register Rd.
   o. SBIW: Subtracts an immediate value (0-63) from a register pair and places the result in the register pair.
   p. SUB: Subtracts two registers and places the result in the destination register Rd.

DIFFICULTIES

This section is entirely optional. Your grade does not depend on it. But it is recommended that, if you had difficulties of some sort, list them here and how you solved them. By documenting your "bugs" and "bug fixes", you can then quickly go back to these sections in the event that the same bug occurs again, allowing you to quickly fix the problem. An example difficulty may be:

Upon loading the program into the TekBot, the TekBot was turning left instead of forward. The problem was a wiring issue with the left motor as the left direction and enable wires were crossed. By swapping the wires, the Left Motor began moving forward and the problem was fixed.

CONCLUSION

The conclusion should sum up the report along with maybe a personal though on the lab. For example, in this lab, we were simply required to set up an AVRStudio4 project with an example program, compile this project and then download it onto our TekBot bases. The result of this program allowed the TekBot to behave in a BumpBot fashion. The lab was great and allowed us the time to build the TekBot with the AVR board and learn the software for this lab.

SOURCE CODE

Provide a copy of the source code. Here you should use a mono-spaced font and can go down to 8-pt in order to make it fit. Sometimes the conversion from standard ASCII to a word document may mess up the formatting.

```
;***********************************************************
;*
;*      BasicBumpBot.asm       -       V2.1
;*
;*      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: Joseph Noonan
;*         Date: January 9, 2020
;*      Company: TekBots(TM), Oregon State University - EECS
;*      Version: 2.1
;*
;***********************************************************
;*      Rev     Date    Name           Description
;*---------------------------------------------------------
;*      -       3/29/02 Zier           Initial Creation of Version 1.0
;*      -       1/08/09 Sinky          Version 2.0 modifictions
;*      -       1/09/20 Noonan         Added extra wait time for reversal
;*
;***********************************************************

.include "m128def.inc"                      ; Include definition file
```

```
;***********************************************************
;* Variable and Constant Declarations
;***********************************************************
.def    mpr = r16                               ; Multi-Purpose 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    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    $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 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

;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:
            in          mpr, PIND           ; Get whisker input from Port D
            andi    mpr, (1<<WskrR|1<<WskrL)
            cpi         mpr, (1<<WskrL)     ; Check for Right Whisker input (Recall
Active Low)
            brne    NEXT                ; Continue with next check
            rcall   HitRight            ; Call the subroutine HitRight
            rjmp    MAIN                ; Continue with program
NEXT:   cpi         mpr, (1<<WskrR)     ; Check for Left Whisker input (Recall Active)
            brne    MAIN                ; No Whisker input, continue program
            rcall   HitLeft             ; Call subroutine HitLeft
            rjmp    MAIN                ; Continue through main

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

;-----------------------------------------------------------
; Sub:  HitRight
; Desc: Handles functionality of the TekBot when the right whisker
;       is triggered.
;-----------------------------------------------------------
HitRight:
            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, (WTime*2)  ; 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
            pop         mpr         ; Restore mpr
            ret                     ; Return from subroutine

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

```
; Sub:  HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
;               is triggered.
;-----------------------------------------------------------------
HitLeft:
                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, (WTime*2)     ; 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
                pop     mpr             ; Restore 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
;-----------------------------------------------------------------
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
```