ECE 375 LAB 1

Introduction to AVR Development Tools

Lab Time: Thursday 1000-1200

Eric Prather – prathere@oregonstate.edu - *932580666*

IMPORTANT:

As per instructions, I have left all of the text in this lab report as-is (as provided on the template). However, in order to complete the requirements of this week's lab, I have made the following changes:

1. I updated the description of the wait subroutine to reflect how it was changed in the challenge code.
2. I added a new subsection for study questions under the regular, provided section for questions in this lab report template. They are answered in full.
3. I added my challenge code as a separate appendix.

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.

For the challenge version of the code, the wait routine was modified. In the modified version, the total time waited is doubled. This is achieved by nesting the main “busylooping” behavior of the subroutine in another loop that executes a total of two times. The defined constants within the “busyloop” were untouched as there was no description of why they were assigned their rather arbitrary values of “224” and “237”. Instead of incrementing one of those numbers by one, because I didn’t understand why those were assigned the way they were, I put a wrapping loop to be safe. It would have been more intuitive if the assigned values were close to each other, instead of “13” apart.

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.

IMPORTANT: Lab 1 ***does*** *have questions. I have written them, as well as their answers, in the following subsection labeled "Study Questions"*

1) Should your lab write-up discuss a narrative of how you accomplished the lab?

No! Remember that this is a professional report and a narrative comment such as "*First we downloaded the lab handout and then the skeleton code. We then followed the TAs instructions…*" should not be used within this report. Simply describe how your program behaves and answer the questions will suffice.

2) What is the purpose of creating a professional Lab report?

Until this class, most students have only been exposed to Technical Writing during Technical Writing course. Since this is a Junior level course, this means that you are close to graduating an entering into the work force or doing an internship. During this time, when your boss requests a report, he/she is expecting a professionally written report. Remember that as engineers, we are expected to be and act professional. So, by requiring you to write these lab reports, you are gain valuable experience need to write professionally.

STUDY QUESTIONS

Study questions are added to this lab report as required by the file "ece-375-lab1.pdf"

1) Go to the lab webpage and download the template write-up. Read it thoroughly and get familiar with the expected format. What specific font is used for source code, and at what size? From here on, when you include your source code in your lab write-up, you must adhere to the specified font type and size.

The specific font used for source code is Courrier New and its size is eight.

2) Go to the lab webpage and read Syllabus carefully. Expected format and naming convention are very important for submission. If you do not follow naming conventions and formats, you will lose some points. What is the naming convention for source code (asm)? What is the naming convention for source code files if you are working with a partner?

The naming convention for source code (asm) is "Lastname_Firstname_Lab#_sourcecode.asm". The naming convention for source code files that are worked on with a partner is "Firstname_Lastname_and_Firstname_Last name_Lab#_sourcecode.asm".

3) Take a look at the code you downloaded for today's lab. Notice the lines that begin with .def and .equ followed by some type of expression. These are known as pre-compiler directives. Define pre-compiler directive. What is the difference between the .def and .equ directives? (HINT: see Section 5.1 of the AVR Starter Guide).

A pre-complier directive is a statement in source code interpreted and executed by the complier on the compilation environment and resources prior to any other interpretation or translation. The two types of precompiler directives in the source code, .def and .equ, are register assignments and macro definitions respectively.

4) Take another look at the code you downloaded for today's lab. Read the comment that describes the macro definitions. From that explanation, determine the 8-bit binary value that each of the following expressions evaluates to. Note: the numbers below are decimal values.

(a) (1 << 3)
b00000001 << 3 = b00001000 = **8**
(b) (2 << 2)
b00000010 << 2 = b00001000 = **8**
(c) (8 >> 1)
b00001000 >> 1 = b00000100 = **4**
(d) (1 << 0)
b00000001 << 0 = b00000001 = **1**
(e) (6 >> 1|1 << 6)
b00000110 >> 1 | b00000001 << 6
= b00000011 | b01000000
= b01000011 = 64 + 2 + 1 = **67**

5) Go to the lab webpage and read the AVR Instruction Set Manual. Based on this manual, describe the instructions listed below. ADIW, BCLR, BRCC, BRGE, COM, EOR, LSL, LSR, NEG, OR, ORI, ROL, ROR, SBC, SBIW, and SUB.

Keyword definitions are provided as follows as an unordered list:

- ADIW: Add Immediate to Word: Adds a value from 0 to 63 on a register pair in the upper four pairs, conforming to endianness of the word.
- BCLR: Clears a flag (bit index 0-7) in SREG, a special register used to keep track of status.
- BRCC: Branch if Carry Cleared: Conditional statement which performs a goto (relatively -63 to +64) if the carry bit is set. Equivalent to BRBC 0,k
- BRGE: Branch if Greater or Equal: Conditional statement which performs a goto (relatively -63 to +64) if the value in register Rd is greater than or equal to that in Rr. Rd and Rr are written to by other commands.
- COM: One's compliment of Rd. What it says on the tin.
- EOR: Exclusive OR of Rd and Rr into Rd. What it says on the tin.
- LSL: Logial Shift left: Bitshift left by 1 of Rd. Sets bit 0 to 0 and sets SREG's C flag to bit 7. *Note: this multiplies unsigned and unsigned values by 2*.
- RSR: Logical shift right: Bitshift right by 1 of Rd. Sets bit 7 to 0 and sets SREG's C flag to bit 0. *Note: This operation divides an unsigned value by two, but does NOT work for signed values*.
- NEG: Two's compliment: Two's compliment of Rd

- OR: Logical Or: OR of Rd and Rr to Rd; what it says on the tin
- ORI: Logical Or with Immediate : Same as logical OR, but with a constant instead of Rd.
- ROL: Rotate left trough carry: Bitshift left by one, overflowing bit seven to bit zero. Important part of multi-word multiplication.
- ROR: Rotate right through carry. Bitshift right by one, overflowing bit zero to bit seven. Important part of multi-word division.
- SBC: Subtract with carry: Subtracts Rr from Rd. Then subtracts carry bit from the difference. Stores result in Rd. Does not change carry bit.
- SBIW Subtract Immediate from word: Subtracts a constant from Rd. Only operates on upper four register pairs.
- SUB: Subtract without carry: Subtracts Rr from Rd.

Challenge code is in appendix.

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.

The most pertinent difficulty to this lab were nebulous submission requirements and directions, specifically as they pertained to the content of the lab report with respect to the template and main lab document

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 (ORIGINAL)

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. Make sure to reformate the code so it looks nice and is readable.

IMPORTANT: While this lab report provides V1.0 of the code in this section, the actual compiled version of the code as used during the lab was V2.0. For the sake of adherence to directions,

```
;***********************************************************
;*
;*      BasicBumpBot.asm       -       V1.0
;*
;*      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 v1.03 TekBots plateform.
;*      For v1.02 TekBots, comment and uncomment the appropriate
;*      code in the constant declaration area as noted.
;*
;*      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
;*         Date: March 29, 2003
;*      Company: TekBots(TM), Oregon State University - EECS
;*      Version: 1.0
;*
;***********************************************************
;*      Rev     Date    Name            Description
;*----------------------------------------------------------
;*      -       3/29/02 Zier            Initial Creation of Version 1.0
;*
;*
;***********************************************************

.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 = 4                               ; Right Whisker Input Bit
.equ    WskrL = 5                               ; 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.
;/////////////////////////////////////////////////////////////
                                ; Move Forwards Command
.equ    MovFwd = (1<<EngDirR|1<<EngDirL)
.equ    MovBck = $00            ; Move Backwards Command
.equ    TurnR = (1<<EngDirL)    ; Turn Right Command
.equ    TurnL = (1<<EngDirR)    ; Turn Left Command
                                ; Halt Command
.equ    Halt = (1<<EngEnR|1<<EngEnL)

;-----------------------------------------------------------
; 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:
        ; Initilize 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, $00        ; Initialize Port B for outputs
        out     PORTB, mpr      ; Port B outputs low
        ldi     mpr, $ff        ; Set Port B Directional Register
        out     DDRB, mpr       ; for output

        ; Initialize Port E for inputs
        ldi     mpr, $FF        ; Initialize Port E for inputs
        out     PORTE, mpr      ; with Tri-State
        ldi     mpr, $00        ; Set Port E Directional Register
        out     DDRE, mpr       ; for inputs

        ; Initialize TekBot Foward Movement
        ldi     mpr, MovFwd     ; Load Move Foward Command
        out     PORTB, mpr      ; Send command to motors

;-----------------------------------------------------------------
; Main Program
;-----------------------------------------------------------------
MAIN:
        in      mpr, PINE       ; Get whisker input from Port D
        andi    mpr, (1<<WskrR|1<<WskrL) ; Mask the whiskers
        cpi     mpr, (1<<WskrR); Check for Right Whisker input
        brne    NEXT            ; Continue with next check
        rcall   HitRight        ; Call the subroutine HitRight
        rjmp    MAIN            ; Continue with program
NEXT:   cpi     mpr, (1<<WskrL); Check for Left Whisker input
        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 Backwards command
        out     PORTB, mpr      ; Send command to port
        ldi     waitcnt, WTime  ; 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 Forwards 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 Backwards command
        out     PORTB, mpr      ; Send command to port
        ldi     waitcnt, WTime  ; Wait for 1 second
        rcall   Wait            ; Call wait function

        ; Turn right for a second
        ldi     mpr, TurnR      ; Load Turn Right 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 Forwards 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
```

SOURCE CODE (CHALLENGE)

Here is the modified version of the above source code specifically tailored for this week's challenge.

```
; Author: Eric Prather
; Date: January 9, 2020
;*********************************************************
;*
;*      BasicBumpBot.asm (renamed prathereLab1_challenge.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.
;*
;*  This version of the file was modified for the "challenge
;*  code" requirement presented in the Lab 1 document. The
;*  length of the "wait" coroutine is doubled. For more
;*  information, see the lab report.
;*
;*********************************************************
;*
;*       Original Author: David Zier and Mohammed Sinky
;*                       (modification Jan 8, 2009)
;*  New Author: Eric Prather (prathere@oregonstate.edu)
;*            Date: January 8, 2009
;*         Company: Oregon State University Student
```

```
;*         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
;*  -   1/10/20 Prather     Challenge code for lab 1.
;***********************************************************

.include "m128def.inc"                      ; Include definition file

;***********************************************************
;* Variable and Constant Declarations
;***********************************************************

; r## is register number - these are just macros
.def    mpr = r16                           ; Multi-Purpose Register
.def    waitcnt = r17                       ; Wait Loop Counter
.def    ilcnt = r18                         ; Inner Loop Counter
.def    olcnt = r19                         ; Outer Loop Counter
.def    solcnt = r20                    ; Super outer loop counter (custom)

.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 ; 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 ; 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
;         Function modification; I changed it so that the inner loop runs twice as long; this
should make any
;       time the function calls run twice as long too.
;-----------------------------------------------------------------
Wait:
                push    waitcnt                 ; Save wait register
                push    ilcnt                   ; Save ilcnt register
                push    olcnt                   ; Save olcnt register

                ; Loop has two components:
                ; OLoop, or "outer loop" - stores loop amount register
                ; ILoop, or "inner loop: - does the logic that should execute
                ; olcnt: outer loop count
                ; ilcount: inner loop count
                ; waitcnt: wait count
                ; brne: goto?
                ; Pseudocode:
                ; Loop {
                ;     Oloop {
                ;          Iloop {
                ;               dec ilcnt
                ;          dec olcnt
                ;     dec waitcnt


Loop:   ldi             solcnt, 2               ; load solcnt register ; original value N/A
SOLoop: ldi             olcnt, 224              ; load olcnt register ; original value 224
OLoop:  ldi             ilcnt, 237              ; load ilcnt register ; original value 237
ILoop:  dec             ilcnt                   ; decrement ilcnt
                brne    ILoop                   ; Continue Inner Loop
                dec             olcnt           ; decrement olcnt
                brne    OLoop                   ; Continue Outer Loop
                dec             solcnt
                brne    SOLoop
                dec             waitcnt         ; Decrement wait
                brne    Loop                    ; Continue Wait loop
                ; Closing curly brace

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