ECE 375 Lab 1

Introduction to AVR Tools

Lab Time: Friday 4-6pm

Aaron Vaughan

TA Signature

1 Introduction

Text goes here

2 Internal Register Definitions and Constants

Text goes here

3 Interrupt Vectors

Text goes here

4 Program Initialization

Text goes here

5 Main Program

Text goes here

6 A Subroutine

Text goes here

7 Stored Program Data

Text goes here

8 Additional Questions

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.

   (a) My source code will be any legible font that is single spaced and must have a minimum font size of 8-pt. I will then examine the print out to ensure readability.

2. Go to the lab webpage and read the 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 your partner?

   (a) From the lab website: "First name_Last name_Lab4_sourcecode.asm"
   "First name_Last name_and_First name_Last name_Lab4_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) the pre-compiler directives occur before compiling the code. .def, .equ and other expressions direct the compiler.
   .equ can be used to set constants within the program while .def gives the programmer the ability to assign a specific register to any given symbol. Both .equ and .def make the code more readable to a programmer.

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) $(1 \ll 3)$
   b(00001000)

   (b) $(2 \ll 2)$
   b(00001000)

   (c) $(8 \gg 1)$
   b(00000100)

   (d) $(1 \ll 0)$
   b(00000001)

   (e) $(6 \gg 1|1 \ll 6)$
   b(00000011)|b(01000000) = b(01000011)

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.

   - ADIW
   ADIW is a 16-bit addition operation involving a value that is passed in as the second part of the argument.

- BCLR
  BCLR will clear any bit within the SREG
- BRCC
  BRCC means branch if carry cleared. It checks the carry flag 'C' and will branch on the condition C=0.
- BRGE
  BRGE means branch if greater or equal. It checks the signed flag 'S' after a subtraction operation. This will branch if S =0 or S $>$ 0.
- COM
  COM will perform the one's compliment operation on rd.
- EOR
  EOR is an exclusive or operation performed on Rd and Rr.
- LSL
  LSL means logical shift left. It effectively multiplies Rd by two.
- LSR
  LSR means logical shift right, and divides Rd by two.
- NEG
  NEG performs the two's compliment of the content of Rd.
- OR
  OR is logical (bitwise) OR between Rd and Rr.
- ORI
  ORI is the logical OR discussed above performed between Rd and a constant that is put into the expression as the second argument.
- ROL ROL Shifts all bits in Rd one place to the left. The C flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C flag. This operation, combined with LSL, effectively multiplies multi-byte signed and unsigned values by two. As described on microchip.com (https://www.microchip.com/webdoc/avrassembler/avrassembler.wb_ROL.html)
- ROR ROR is similar to the way that ROL operates with the exception that the bits are shifted to the right.
- SBC SBC subtracts with carry. This uses the C bit in SREG and stores the result in Rd.
- SBIW SBIW Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers. As described on microchip.com (https://www.microchip.com/webdoc/avrassembler/avrassembler.wb_ROL.html)
- SUB SUB subtracts two registers and puts the difference into Rd

9 Difficulties

Finding all of the acronym definitions was not even remotely possible to do from the avr starter guide provided on the lab website. Some of the required definitions were embedded into tables with zero explanation as to their intended use or operation. I got all of the definitions from microchip.com (https://www.microchip.com/webdoc/avrassembler/avrassembler.wb_ROL.html)

10 Conclusion

Text goes here

11 Source Code

Source code goes here. It looks best if each line is no more than 60 characters.

```
;***********************************************************
;*
;*       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: Aaron Vaughan
;*          Date: October 7, 2019
;*       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 = 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    WTimeBack = 200                         ; Time to wait in backup 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.
;/////////////////////////////////////////////////////////////
```

```
57
58  .equ    MovFwd = (1<<EngDirR|1<<EngDirL)        ; Move Forward Command
59  .equ    MovBck = $00                            ; Move Backward Command
60  .equ    TurnR = (1<<EngDirL)                    ; Turn Right Command
61  .equ    TurnL = (1<<EngDirR)                    ; Turn Left Command
62  .equ    Halt = (1<<EngEnR|1<<EngEnL)            ; Halt Command
63
64  ;=============================================================
65  ; NOTE: Let me explain what the macros above are doing.
66  ; Every macro is executing in the pre-compiler stage before
67  ; the rest of the code is compiled.  The macros used are
68  ; left shift bits (<<) and logical or (|).  Here is how it
69  ; works:
70  ;       Step 1.  .equ   MovFwd = (1<<EngDirR|1<<EngDirL)
71  ;       Step 2.          substitute constants
72  ;                         .equ   MovFwd = (1<<5|1<<6)
73  ;       Step 3.          calculate shifts
74  ;                         .equ   MovFwd = (b00100000|b01000000)
75  ;       Step 4.          calculate logical or
76  ;                         .equ   MovFwd = b01100000
77  ; Thus MovFwd has a constant value of b01100000 or $60 and any
78  ; instance of MovFwd within the code will be replaced with $60
79  ; before the code is compiled.  So why did I do it this way
80  ; instead of explicitly specifying MovFwd = $60?  Because, if
81  ; I wanted to put the Left and Right Direction Bits on different
82  ; pin allocations, all I have to do is change thier individual
83  ; constants, instead of recalculating the new command and
84  ; everything else just falls in place.
85  ;=============================================================
86
87  ;***********************************************************
88  ;* Beginning of code segment
89  ;***********************************************************
90  .cseg
91
92  ;-----------------------------------------------------------
93  ; Interrupt Vectors
94  ;-----------------------------------------------------------
95  .org    $0000                           ; Reset and Power On Interrupt
96                  rjmp    INIT            ; Jump to program initialization
97
98  .org    $0046                           ; End of Interrupt Vectors
99  ;-----------------------------------------------------------
100 ; Program Initialization
101 ;-----------------------------------------------------------
102 INIT:
103     ; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
104                 ldi             mpr, low(RAMEND)
105                 out             SPL, mpr                ; Load SPL with low byte of RAMEND
106                 ldi             mpr, high(RAMEND)
107                 out             SPH, mpr                ; Load SPH with high byte of RAMEND
108
109     ; Initialize Port B for output
110                 ldi             mpr, $FF                ; Set Port B Data Direction Register
111                 out             DDRB, mpr               ; for output
112                 ldi             mpr, $00                ; Initialize Port B Data Register
113                 out             PORTB, mpr              ; so all Port B outputs are low
114
115         ; Initialize Port D for input
116                 ldi             mpr, $00                ; Set Port D Data Direction Register
117                 out             DDRD, mpr               ; for input
118                 ldi             mpr, $FF                ; Initialize Port D Data Register
119                 out             PORTD, mpr              ; so all Port D inputs are Tri-State
120
121                 ; Initialize TekBot Forward Movement
122                 ldi             mpr, MovFwd             ; Load Move Forward Command
123                 out             PORTB, mpr              ; Send command to motors
124
125 ;-----------------------------------------------------------
126 ; Main Program
127 ;-----------------------------------------------------------
128 MAIN:
129                 in              mpr, PIND               ; Get whisker input from Port D
130                 andi    mpr, (1<<WskrR|1<<WskrL)
131                 cpi             mpr, (1<<WskrL) ; Check for Right Whisker input (Recall Active Low)
132                 brne    NEXT                    ; Continue with next check
133                 rcall   HitRight                ; Call the subroutine HitRight
134                 rjmp    MAIN                    ; Continue with program
135 NEXT:   cpi             mpr, (1<<WskrR) ; Check for Left Whisker input (Recall Active)
136                 brne    MAIN                    ; No Whisker input, continue program
137                 rcall   HitLeft                 ; Call subroutine HitLeft
138                 rjmp    MAIN                    ; Continue through main
139
140 ;***********************************************************
141 ;* Subroutines and Functions
142 ;***********************************************************
143
```

```
144 ;-----------------------------------------------------------
145 ; Sub:  HitRight
146 ; Desc: Handles functionality of the TekBot when the right whisker
147 ;               is triggered.
148 ;-----------------------------------------------------------
149 HitRight:
150                 push    mpr                     ; Save mpr register
151                 push    waitcnt                 ; Save wait register
152                 in              mpr, SREG       ; Save program state
153                 push    mpr                     ;
154
155                 ; Move Backwards for a second
156                 ldi             mpr, MovBck     ; Load Move Backward command
157                 out             PORTB, mpr      ; Send command to port
158                 ldi             waitcnt, WTimeBack      ; Wait for 2 second
159                 rcall   Wait                    ; Call wait function
160
161                 ; Turn left for a second
162                 ldi             mpr, TurnL      ; Load Turn Left Command
163                 out             PORTB, mpr      ; Send command to port
164                 ldi             waitcnt, WTime  ; Wait for 1 second
165                 rcall   Wait                    ; Call wait function
166
167                 ; Move Forward again
168                 ldi             mpr, MovFwd     ; Load Move Forward command
169                 out             PORTB, mpr      ; Send command to port
170
171                 pop             mpr             ; Restore program state
172                 out             SREG, mpr       ;
173                 pop             waitcnt         ; Restore wait register
174                 pop             mpr             ; Restore mpr
175                 ret                             ; Return from subroutine
176
177 ;-----------------------------------------------------------
178 ; Sub:  HitLeft
179 ; Desc: Handles functionality of the TekBot when the left whisker
180 ;               is triggered.
181 ;-----------------------------------------------------------
182 HitLeft:
183                 push    mpr                     ; Save mpr register
184                 push    waitcnt                 ; Save wait register
185                 in              mpr, SREG       ; Save program state
186                 push    mpr                     ;
187
188                 ; Move Backwards for a second
189                 ldi             mpr, MovBck     ; Load Move Backward command
190                 out             PORTB, mpr      ; Send command to port
191                 ldi             waitcnt, WTimeBack      ; Wait for 2 second
192                 rcall   Wait                    ; Call wait function
193
194                 ; Turn right for a second
195                 ldi             mpr, TurnR      ; Load Turn Left Command
196                 out             PORTB, mpr      ; Send command to port
197                 ldi             waitcnt, WTime  ; Wait for 1 second
198                 rcall   Wait                    ; Call wait function
199
200                 ; Move Forward again
201                 ldi             mpr, MovFwd     ; Load Move Forward command
202                 out             PORTB, mpr      ; Send command to port
203
204                 pop             mpr             ; Restore program state
205                 out             SREG, mpr       ;
206                 pop             waitcnt         ; Restore wait register
207                 pop             mpr             ; Restore mpr
208                 ret                             ; Return from subroutine
209
210 ;-----------------------------------------------------------
211 ; Sub:  Wait
212 ; Desc: A wait loop that is 16 + 159975*waitcnt cycles or roughly
213 ;               waitcnt*10ms.  Just initialize wait for the specific amount
214 ;               of time in 10ms intervals. Here is the general eqaution
215 ;               for the number of clock cycles in the wait loop:
216 ;                       ((3 * ilcnt + 3) * olcnt + 3) * waitcnt + 13 + call
217 ;-----------------------------------------------------------
218 Wait:
219                 push    waitcnt                 ; Save wait register
220                 push    ilcnt                   ; Save ilcnt register
221                 push    olcnt                   ; Save olcnt register
222
223 Loop:   ldi             olcnt, 224              ; load olcnt register
224 OLoop:  ldi             ilcnt, 237              ; load ilcnt register
225 ILoop:  dec             ilcnt                   ; decrement ilcnt
226                 brne    ILoop                   ; Continue Inner Loop
227                 dec             olcnt           ; decrement olcnt
228                 brne    OLoop                   ; Continue Outer Loop
229                 dec             waitcnt         ; Decrement wait
230                 brne    Loop                    ; Continue Wait loop
```

```
231
232                 pop             olcnt           ; Restore olcnt register
233                 pop             ilcnt           ; Restore ilcnt register
234                 pop             waitcnt         ; Restore wait register
235                 ret                             ; Return from subroutine
```