ECE 375 LAB 1

Introduction to AVR Development Tools

Lab Time: Tuesday 4-6

Sonia Camacho

Owen Markley

STUDY QUESTIONS

1. The font for the source code must be Courier new and may be as small as 8pt font as well as it must be mono spaced.

   2. **Alone source:**
      "First name_Last name_Lab4_sourcecode (e.g. Youngbin_Jin_Lab4_sourcecode.asm)"
      Partner source:
      "First name_Last name_and_First name_Last name_Lab4_sourcecode.asm (e.g. Youngbin_Jin_and_Arul_Dhamodaran_Lab4_sourcecode.asm)"(Note: Include both of your name in a comment in your .asm file).

3. Before the code compiles the pre-compiler is executed, this pre-compiler is what the AVR guide describes as instructions these are used to tell the computer what is going on and where to direct the compiler. The EQU assigns a value to a label which can then later be used, note that this value will remain constant and can not be redefined or altered.

4.

   00001000

   00001000

   00000100

   00000001

   01000011

5.) .

ADIW: This will add a value of value (0-63) this will be added to a register pair. This operates onthe upper four register pairs.

BCLR: The BCLR “Clears a single Flag in SREG.”

BRCC: This tests the carry flag and branches relitively to the personal computer only if the carry flag is cleared. “This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 0,k.)”

BRGE: this tests the signed flag and branches relatively to the personal computer only if the signed flag is cleared. “If the instruction is executed immediately after any of the instructions CP, CPI, SUB, or SUBI, the branch will occur if and only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 4,k.)”

COM: on the register Rd, the ones complement will be performed.

EOR: This will do the logical EOR for the register Rd and Rr then it will place the result in the Rd register which is the destination register.

LSL: This will shift the bits in the register Rd all one spot to the left. Then the bit 0 will be cleared out and bit 7 is loaded into the c flag on the SREG “This operation effectively multiplies signed and unsigned values by two.”

LSR: This will shift all of the bits in the Rd to the right and then clear the 7th bit and then load the 0 bit into the c flag of the SREG. “This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.”

NEG: Replaces Rd’s contents with the two’s complement, $80 doesn’t change

OR: Does logical OR on Rd and Rr, the evaluation gets placed into register Rd

ORI: Does logical OR on Rd and a constant, the evaluation gets placed into register Rd

ROL: All of Rd’s bits shift left by one. C flag moves to the 0 bit of Rd. but 7 goes into the C flag. If you combine this with LSL, it's the same as multiplying a multi-byte signed and unsigned value by two

ROR: Shifts Rd bits to the right. C flag moves to 7 bit of Rd. 0 bit goes to C flag. Combining this with ASR divides multi-byte signed values by two. LSR can be used for unsigned, and the carry flag can be used to round

SBC: Subtracts two registers with the C flag, the evaluation is placed in Rd

SBIW: Subtracts intermediate value ranging from 0 to 63 from a pair of registers and places the result there. Works on upper four register pairs, and is good for pointer

SUB: two registers are subtracted and the evaluation goes into Rd.

SOURCE CODE

```
;***********************************************************
;*
;*      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
```