ECE 375 LAB 1

Introduction to AVR Development Tools

Lab Time: Wednesday 12-2

Bradley Martin

STUDY QUESTIONS

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

Source code font size must be of monotone size but can go down to 8-pt font to get the text to fit.

2) What is the naming convention for source code (asm)?

Source code files follow the naming convention of "First name_Last name_lab#_sourcecode".

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).

Pre-compiler directives are instructions executed before code is compiled. These instructions direct the compiler to adjust location of program memory, define macros, initialize memory, etc. The main difference is that .def variables can be redefined after declaration whereas .equ cannot and must remain constant.

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) = b00001000

b) (2 << 2) = b00001000

c) (8 >> 1) = b00000100

d) (1 << 0) = b00000001

e) (6 >> 1 | 1 << 6) = b01000110

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 - Will immediately add to a register pair and place the result in the register.

BCLR – Clears the bits in the status register.

BRCC- Tests the carry flag and will branch based on program counter if carry flag is cleared.

BRGE- Tests the signed flag and will branch based on program counter if signal flag is cleared. Rd must be greater than or equal to the signed binary of Rr.

COM- This will perform a One's compliment of register Rd.

EOR- Performs an exclusive OR between registers Rd and Rr placing the result in Rd.

LSL- This shifts bits in Rd one place to the left. This multiplies signed and unsigned values by two.

LSR- This shifts bits in Rd one place to the right. This divides an unsigned value by two.

NEG- Replaces register Rd's contents with its two's complement.

OR- Performs the OR operation between register Rd and register Rr. The result is put in Rd

ORI- Performs the OR operation between register Rd and a constant. The result is put in Rd

ROL- This shifts all bits in Rd to the left by one. This multiplies multi-byte signed and unsigned values by two.

ROR- This shifts all bits in Rd to the right by one. These divides multi-byte signed values by two. If combined with LSR it can divide multi-bye unsigned values by two.

SBC- This subtracts two registers and subtracts the C flag. The result is placed in the register Rd.

SBIW- Subtracts an immediate value from a register pair and places the result in the register pair.

SUB- Subtracts two registers and places the result in register Rd.

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

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