ECE 375 LAB 1

Introduction to AVR Development Tools

Lab Time: Wednesday 10-12

Tu Lam

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 type of specific font used for the section "Source Code" rely on using the font mono-spaced. On top of that, the size of the font must be in size 8-pt to fit the code in the section.

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

The naming convention for source code (asm) should follow your first and last name, follow by the lab number of that lab, and the title label it as source code. An example convention would be like:

Firstname_Lastname_Lab#_sourcecode.asm , but also include your name and date in the .asm file when writing the code.

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

Pre-complier directive can be defined as in the AVR Stater Guide as "special instructions that are executed before the code is compiled and directs the compiler". In there, the difference between ***.def*** and ***.equ*** directives are that ***.def*** is referring to "Define a symbolic name on a register" while the ***.equ*** is refer to "Set a symbol equal to an expression".

4) Determine the 8-bit binary value that each of the following expressions evaluates to. Note: the numbers below are decimal values.

a) (1 << 3) = 00001000

b) (2 << 2) = 00001000

c) (8 >> 1) = 00000100

d) (1 << 0) = 00000001

e) (6 >> 1 | 1 << 6) = 01000011

5) 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: Addition arithmetic with 16 bits with intermediate value

BCLR: Clear any bit within the SREG register.

BRCC: Conditionally Branch if Carry is Clear

BRGE: Conditionally branch is greater than or equal

COM: This is a complement or the opposite of the binary

EOR: A logic meaning “exclusive or” same thing with a XOR

LSL: “Logical shift left” meaning shift to the left 0 to 31 places

LSR: “Logical shift right” meaning shift to the right 0 to 32 places

NEG: Same thing as complement and reverse binary

OR: The logic of OR depend on the input

ORI: OR operation between one register, with an immediate value

ROL: Rotate left through carry bits

ROR: Rotate right through carry bits

SBC: Subtraction involve a carry bit

SBIW: Subtraction with intermediate value with 16 bits

SUB: Arithmetic doing the subtraction

CONCLUSION

The first ever lab for ECE 375, we learn about setting up the program that will be use in the class for the rest of the term. We also learn how to navigate around the program and learn to use the mega128 board and how it works. Also we review on how to submit the .asm file with the format that is required for the class and follow the template for the lab write-up and learn about the template of the pre-lab too.

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 -       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: Tu Lam
;*         Date: September 30, 2020
;*      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    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
;-----------------------------------------------------------------
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
```