ECE 375 Lab 1

Introduction to AVR Development Tools

Lab Time: Tuesday 4-6

Alexander Uong

Introduction

Not Required for Lab 1.

Program Overview

Not Required for Lab 1.

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.

   For source code, the code should be in a mono-spaced font in size 8-pt. In the word document, the source code is in Courier New font at size 8-pt.

2) Go to the lab webpage and read the Syllabus section 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)?

   The naming convention for the source code (asm) is "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 defined as instructions that are executed prior to the code being compiled and directing the compiler. These directives are used to adjust the location of the program in memory, initialize memory, define macros, and more.

The .DEF pre-compiler directive defines a symbolic name on a register as the .EQU pre-compiler directive sets a symbol equal to an expression.

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 << 5)

      b00000001 << 5

      = b00100000

   b) (4 << 4)

      b00000100 << 4

      = b01000000

   c) (8 >> 1)

      b00001000 >> 1

      = b00000100

   d) (5 << 0)

      b00000101 << 0

      = b00000101

   e) (8 >> 2|1 << 6)

      b00001000 >> 2 | b00000001 << 6

b00000010 | b01000000

=b01000010

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 (Add Immediate to Word): This adds an immediate value to a register pair and places the result in the register pair. This operates on the upper four register pairs.

   BCLR (Bit Clear in SREG): Clears a single flag in SREG.

   BRCC (Branch if Carry Cleared): Tests the Carry Flag and branches relatively to PC if C is cleared. It can branch to PC in either direction.

   BRGE (Branch if Greater or Equal (Signed)): A conditional relative branch that tests the Signed Flag and branches relatively to PC if the Signed Flag is cleared. If BRGE is executed after the instructions CP, CPI, SUB, or SUBI, the branch will occur only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr.

   COM (One's Complement): Performs a one's complement of register Rd.

   EOR (Exclusive OR): Performs logical exclusive OR between the contents of register Rr and register Rd, placing the results in the register Rd (destination register).

   LSL (Logical Shift Left): Shifts all bits in Rd a single place to the left. When this instruction is executed, Bit 0 is cleared and Bit 7 is loaded into the C flag of SREG. This leads to signed and unsigned values being multiplied by two.

   LSR (Logical Shift Right): Shifts all bits in Rd a single place to the right. When this instruction is executed, Bit 7 is cleared and Bit 0 is loaded into the C Flag of the SREG. This leads to unsigned values being divided by two.

   NEG (Two's Complement): Replaces the contents of register Rd with its two's complement.

OR (Logical OR): Performs the logical OR between the contents of register Rd and register Rr, placing the result in the register Rd (destination).

ORI (Logical OR with Immediate): Performs the logical OR between the contents of register Rd and a constant, placing the result in the register Rd (destination).

ROL (Rotate Left through Carry) Shifts all bits in Rd a place to the left. The C flag is shifted into Bit 0 of Rd and Bit 7 is shifted into the C Flag. Combined with LSL instruction, ROL multiplies multi-byte signed and unsigned values by two.

ROR (Rotate Right through Carry): Shifts all bits in Rd a place to the right. The C flag is shifted into biy 7 of Rd and Bit 0 is shifted into the C flag. Combined with LSR instruction, ROR divides multi-byte unsigned values by two.

SBC (Subtract with Carry): Subtracts two registers and subtracts with the C flag, placing the result in the register Rd (destination).

SBIW (Subtract Immediate from Word): Subtracts an immediate value from a register pair, placing the result in the register pair. This instruction operates on the upper four register pairs.

SUB (Subtract Without Carry): Subtracts two registers and places the result in the register Rd (destination).

Difficulties

Not Required for Lab 1.

Conclusion

Not Required for Lab 1.

Challenge Code

```
;***********************************************************
;*
;*     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: Alexander Uong
;*        Date: January 11, 2021
;*     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 = 200                          ; Time to wait in wait loop
.equ   RTime = 100                          ;

.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)
Active Low)   cpi           mpr, (1<<WskrL)      ; Check for Right Whisker input (Recall
              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 2 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, RTime ; 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 2 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, RTime ; 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
```