ECE 375 LAB 1

Introduction to AVR Development Tools

Lab Time: Wednesday 4-5:50 pm

Frankie Herbert

INTRODUCTION

The purpose of the first lab was to download and familiarize ourselves with AVRStudio4 software. Using a pre-made program called “BasicBumpBot” to create a new project in AVRStudio4, this was then built and compiled onto the AVR board. Once the program and the AVR board were tested to ensure that LEDs on the mega32 board would indicate the performance of the “BasicBumpBot” routine was running.

PROGRAM OVERVIEW

The TekBot as whiskers alongside the AVR board in which directs the TekBot to move right, left given certain behaviors within the “BasicBumpBot” program. For this “BasicBumpBot” is the TekBot is needing to trigger those whiskers to reverse for twice as long before then turning away to resume the forward motion. This programs focus was to double the WTime orginally set to 100 milliseconds. In doing so in both HitRight and HitLeft for the Move Backwards WTime, this provided an accumulated wait time of 200 milliseconds or 2 seconds.

INITIALIZATION ROUTINE

“BasicBumpBot” first initializes the Stack Pointer in which will call functions and subroutines in an orderly manner. Port B is then initialized to handle the output and Port E is initialized to handle the inputs alongside the inputs from the whiskers. Lastly, the program initializes the TekBot forward movement which is being outputted to PortB and then sent to the TekBot indicating forward movement.

MAIN ROUTINE

The main routine performs checks on both Right and Left Whisker to see if either were hit by reading a det of data bits from the PINE and then masks the data for either hit of left or right whisker. The program is checking to see if right whisker or left whisker was hit, so that then calls can be made to initiate the HitRight or HitLeft routine of the TekBot. Lastly, the program provides a jump command in which resets the program back to the top of the main routine to reperform all tasks.

SUBROUTINES

1. HitRight Routine

The HitRight Routine first loads the move backwards commands to the TekBot by sending an output to PortB and waiting for 100 milliseconds using the wait routine. Once the wait routine has finished, the program loads the turn left command by sending an output to PortB and then waiting another 100 milliseconds using another wait routine. Upon these being accomplished, the command for moving forward is then outputted to PortB followed by calling the wait routine for another 100 milliseconds. Upon completion of the forward command the TekBot then returns normal from the routine.

2. HitLeft Routine

HitLeft Routine behaves the exact same way as HitRight routine, except there is no left turn routine.

3. Wait Routine

The Wait Routine is set for 100 milliseconds which is used within all movement routines of the TekBot. Since the TekBot is needing to be reversed however, the backwards routine was doubled for the wait routine by another 100 milliseconds in order to accomplish reversing the TekBot.

ADDITIONAL QUESTIONS

1) How can we become more familiar with the assembly code and C language faster to understand what is being given for future labs needing to be student written?

2) Are we to place any Study Questions given within the lab handouts into our lab reports though it is not specified as to where they should go?

DIFFICULTIES

Starting off this lab it was a bit overwhelming with seeing such assembly language and not having a full understanding of what was going on. After reading the fully commented code, I was still a bit confused as to the logistics of everything though had a somewhat understanding of what was to happen and go on. The Lab Document was a little hard to understand and as well provide enough knowledge into the lab and what was needing done. However, I still found starting off this to be rather confusing as well. All new material and just needing a lot more office hours this term to have it pat down.

CONCLUSION

In conclusion of Lab 1, I was able to properly download all required software needed for the AVR board. Programs were then able to be fully uploaded to the AVR board at ease and was a very fun exploration for me.

STUDY QUESTIONS

1. **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 5.4 of the AVR Assembler User Guide).**

   - The difference between .def and .equ is that one is a variable and the other is a constant. Meaning that .def is a defined symbol used throughout the program and is called the variable and can be redefined later in the program. Whereas .equ is a symbol equal to an expression and remains a constant and cannot be changed withing the program.

2. **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. The answer must be written in binary.**

   (a) **(2 << 6)** = 0b01000000

   (b**) (3 << 3)** = 0b00011000

(c) **(8 >> 1)** = 0b01000000

(d) **(255 << 0 & 15 << 4)** = 0b11110000

(e) **(8 >> 3 | 7 << 4)** = 0b01110000

3. **Go to the lab webpage and read the AVR Instruction Set Manual. Based on this manual, describe the instructions listed below**.

- **ADIW** = (Add Immediate Word) this instruction immediately adds to a register pair and places the results within the register. (avr-instruction pg. 33)

- **BCLR** = (Bit Clear in SREG) to clear single Flag in the status register. (avr-instruction pg. 38)

- **BRCC** = (Branch if Carry Cleared) to test the Carry Flag and then branches depending on the PC if the Carry Flag is cleared. (avr-instruction pg. 42)

- **BRGE** = (Branch if Greater or Equal (Signed)) to test the Signed Flag and then branches depending on the PC if the Signed Flag is cleared. (avr-instruction pg. 46)

- **COM** = (One’s Complement) performs an One’s Complement of register Rd. (avr-instruction pg. 76)

- **EOR** = (Exclusive OR) performs EOR between register Rd and register Rr and then places the results in Rd. (avr-instruction pg.91)

- **LSL** = (Logical Shift Left) shifts every bit in Rd one place to the left. (avr-instruction pg. 120)

- **LSR** = (Logical Shift Right) shifts every bit in Rd one place to the right. (avr-instruction pg. 122)

- **NEG** = (Two’s Complement) replaces the contents of register Rd with its two’s complement. (avr-instruction pg. 129)

- **OR** = (Logical OR) performs OR between register Rd and register Rr, then places the result in Rd. (avr-instruction pg. 132)

- **ORI** = (Logical OR with Immediate) performs OR between register Rd and the constant, then places the result in Rd. (avr-instruction pg. 133)

- **ROL** = (Rotate Left through Carry) shifts every bit in register Rd one place to the left. (avr-instruction pg.143)

- **ROR** = (Rotate Right through Carry) shifts every bit in register Rd one place to the right. (avr-instruction pg. 145)

- **SBC** = (Subtract with Carry) subtracts two registers and subtracts with C Flag, then places the result in Rd. (avr-instruction pg. 147)

- **SBIW** = (Subtract Immediate from Word) subtracts an immediate value from a register pair and places the result in the register pair. (avr-instruction pg. 154)

- **SUB** = (Subtract without Carry) subtracts two registers and places the result in register Rd. (avr-instruction pg. 181)

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   -        V3.0
;*
;*       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: Frankie Hebrert (modification October 5, 2022)
;*          Date: October 5, 2022
;*       Company: TekBots(TM), Oregon State University - EECS
;*       Version: 3.0
;*
;***********************************************************
;*       Rev      Date     Name              Description
;*----------------------------------------------------------
;*       -        3/29/02  Zier              Initial Creation of Version 1.0
;*       -        1/08/09 Sinky              Version 2.0 modifications
;*  -   8/10/22 Dongjun               The chip transition from Atmega128 to Atmega32U4
;***********************************************************

.include "m32U4def.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 = 5                                  ; Right Engine Enable Bit
.equ     EngEnL = 6                                  ; Left Engine Enable Bit
.equ     EngDirR = 4                                 ; Right Engine Direction Bit
.equ     EngDirL = 7                                 ; 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<<4|1<<7)
;        Step 3.            calculate shifts
;                            .equ     MovFwd = (b00010000|b10000000)
;        Step 4.            calculate logical or
;                            .equ     MovFwd = b10010000
; Thus MovFwd has a constant value of b10010000 or $90 and any
; instance of MovFwd within the code will be replaced with $90
; before the code is compiled.  So why did I do it this way
; instead of explicitly specifying MovFwd = $90?  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      $0056                                 ; 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 + 100         ; Wait for 1 second, added 100
milliseconds to WTime to double WTime to get 2 seconds
                   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 + 100         ; Wait for 1 second, added 100
milliseconds to WTime to double WTime to get 2 seconds

                   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)-1+4)*olcnt)-1+4)*waitcnt)-1+16
;----------------------------------------------------------------
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
```