ECE 375 Lab 1

Introduction to AVR Tools

Lab Time: Wednesday 10a-12n

Robert Detjens

David Headrick

TA Signature

Introduction

The lab write-up should be done in the style of a professional report/white paper. Proper headers need to be used and written in a clean, professional style. Proof read the report to eliminate both grammatical errors and spelling. The introduction should be a short 1-2 paragraph section discussing what the purpose of this lab is. This is not merely a copy from the lab handout, but rather your own personal opinion about what the object of the lab is and why you are doing it. Basically, consider the objectives for the lab and what you learned and then briefly summarize them. For example, a good introduction to lab 1 may be as follows.

The purpose of this first lab is to provide an introduction on how to use AVRStudio4 software for this course along with connecting the AVR board to the TekBot base. A simple pre-made "BumpBot" program was provided to practice creating a project in AVRStudio4, building the project, and then using the Universal Programmer to download the program onto the AVR board.

Program Overview

This section provides an overview of how the assembly program works. Take the time to write this section in a clear and concise manner. You do not have to go into so much detail that you are simply repeating the comments that are within your program, but simply provide an overview of all the major components within your program along with how each of the components work. Discuss each of your functions and subroutines, interesting program features such as data structures, program flows, and variables, and try to avoid nitty-gritty details. For example, simple state that you "First initialized the stack pointer," rather than explaining that you wrote such and such data values to each register. These types of details should be easily found within your source code. Also, do not hesitate to include figures when needed. As they say, a picture is worth a thousand words, and in technical writing, this couldn't be truer. You may spend 2 pages explaining a function which could have been better explained through a simple program-flow chart. As an example, the remainder of this section will provide an overview for the basic BumpBot behavior.

The BumpBot program provides the basic behavior that allows the TekBot to react to whisker input. The TekBot has two forward facing buttons, or whiskers, a left and a right whisker. By default the TekBot will be moving forward until one of the whiskers are triggered. If the left whisker is hit, then the TekBot will backup and then turn right for a bit, while a right whisker hit will backup and turn left. After the either whisker routine completes, the TekBot resumes its forward motion.

Besides the standard INIT and MAIN routines within the program, three additional routines were created and used. The HitRight and HitLeft routines provide the basic functionality for handling either a Right or Left whisker hit, respectively. Additionally a Wait routine was created to provide an extremely accurate busy wait, allowing time for the TekBot to backup and turn.

Additional Questions

- What font is used for the source code portion of the report?

  Monospaced font at down to 8pt size.

- What is the naming format for source code submissions?

  `$FIRST_$LAST_and_$FIRST_$LAST_$LAB_sourcecode.asm`

- What are pre-compiler directives?

  These are special instructions that the assembler reads to do stuff unrelated to the actual opcodes, such as setting the memory location of things, or setup memory.

  `.def` vs `.equ`?

  `.def` adds a symbolic name for a register, allowing for descriptive names much like a variable. `.equ` does the same but for an expression, somewhat like `#DEFINE` in C.

- Determine the binary values for the following:
  - `(1 <<5)`: `00100000`
  - `(4 <<4)`: `01000000`
  - `(8 >>1)`: `00000100`
  - `(5 <<0)`: `00000101`
  - `(8 >>2|1 <<6)` `01000010`

- Describe the following instructions:
  - ADIW: add an immediate 16-bit value to a register pair
  - BCLR: clears a flag in the status register
  - BRCC: Jump if the carry is not set
  - BRGE: Jump if the sign flag is set
  - COM: Performs one's compliment on target register
  - EOR: XOR between two registers
  - LSL: Shift register left one bit, evicted bit set to carry flag
  - LSR: Shift register right one bit, evicted bit set to carry flag
  - NEG: Performs two's complement on target register
  - OR: ORs two registers
  - ORI: ORs register with immediate value
  - ROL: Shift register left, new bit from carry flag
  - ROR: Shift register right, new bit from carry flag
  - SBC: Subtract two registers and the carry flag
  - SBIW: Subtracts 16-bit constant from two registers
  - SUB: Subtracts two registers (without carry)

Difficulties

I originally installed the GNU GCC AVR toolchain, but the code from this class requires the Atmel AVR assembler instead. Tracking down a Linux version of that proved to be somewhat tricky, but I did eventually find it.

Conclusion

Text goes here

Source Code

Standard source 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: David Zier and Mohammed Sinky (modification Jan 8, 2009)
;*         Date: September 29, 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 = 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 programinitialization

.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 ofRAMEND
  ldi         mpr,high(RAMEND)
  out         SPH, mpr          ; Load SPH with high byte ofRAMEND

  ; Initialize Port B for output
  ldi         mpr, $FF          ; Set Port B Data DirectionRegister
  out         DDRB, mpr         ; foroutput
  ldi         mpr, $00          ; Initialize Port B DataRegister
  out         PORTB, mpr        ; so all Port B outputs arelow

  ; Initialize Port D for input
  ldi         mpr, $00          ; Set Port D Data DirectionRegister
  out         DDRD, mpr         ; forinput
  ldi         mpr, $FF          ; Initialize Port D DataRegister
  out         PORTD, mpr        ; so all Port D inputs areTri-State

  ; Initialize TekBot Forward Movement
```

```
  ldi         mpr, MovFwd       ; Load Move ForwardCommand
  out         PORTB, mpr        ; Send command tomotors

;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:
  in          mpr, PIND         ; Get whisker input from PortD
  andi        mpr,(1<<WskrR|1<<WskrL)
  cpi         mpr, (1<<WskrL)   ; Check for Right Whisker input (Recall ActiveLow)
  brne        NEXT              ; Continue with nextcheck
  rcall       HitRight          ; Call the subroutineHitRight
  rjmp        MAIN              ; Continue withprogram
NEXT:
  cpi         mpr, (1<<WskrR)   ; Check for Left Whisker input (RecallActive)
  brne        MAIN              ; No Whisker input, continueprogram
  rcall       HitLeft           ; Call subroutineHitLeft
  rjmp        MAIN              ; Continue throughmain

;***********************************************************
;* Subroutines and Functions
;***********************************************************

;-----------------------------------------------------------
; Sub:  HitRight
; Desc: Handles functionality of the TekBot when the right whisker
;       is triggered.
;-----------------------------------------------------------
HitRight:
  push        mpr               ; Save mprregister
  push        waitcnt           ; Save waitregister
  in          mpr, SREG         ; Save programstate
  push        mpr               ;

  ; Move Backwards for a second
  ldi         mpr, MovBck       ; Load Move Backwardcommand
  out         PORTB, mpr        ; Send command toport
  ldi         waitcnt, WTime    ; Wait for 1 second
  rcall       Wait              ; Call waitfunction

  ; Turn left for a second
  ldi         mpr, TurnL        ; Load Turn LeftCommand
  out         PORTB, mpr        ; Send command toport
  ldi         waitcnt, WTime    ; Wait for 1second
  rcall       Wait              ; Call waitfunction
```

```
  ; Move Forward again
  ldi         mpr, MovFwd       ; Load Move Forwardcommand
  out         PORTB, mpr        ; Send command toport

  pop         mpr               ; Restore programstate
  out         SREG, mpr         ;
  pop         waitcnt           ; Restore waitregister
  pop         mpr               ; Restorempr
  ret                           ; Return fromsubroutine

;-----------------------------------------------------------------
; Sub:  HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
;       is triggered.
;-----------------------------------------------------------------
HitLeft:
  push        mpr               ; Save mprregister
  push        waitcnt           ; Save waitregister
  in          mpr, SREG         ; Save programstate
  push        mpr               ;

  ; Move Backwards for a second
  ldi         mpr, MovBck       ; Load Move Backwardcommand
  out         PORTB, mpr        ; Send command toport
  ldi         waitcnt, WTime    ; Wait for 1 second
  rcall       Wait              ; Call waitfunction

  ; Turn right for a second
  ldi         mpr, TurnR        ; Load Turn LeftCommand
  out         PORTB, mpr        ; Send command toport
  ldi         waitcnt, WTime    ; Wait for 1second
  rcall       Wait              ; Call waitfunction

  ; Move Forward again
  ldi         mpr, MovFwd       ; Load Move Forwardcommand
  out         PORTB, mpr        ; Send command toport

  pop         mpr               ; Restore programstate
  out         SREG, mpr         ;
  pop         waitcnt           ; Restore waitregister
  pop         mpr               ; Restorempr
  ret                           ; Return fromsubroutine

;-----------------------------------------------------------------
```

```
; 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 waitregister
  push         ilcnt            ; Save ilcntregister
  push         olcnt            ; Save olcntregister

Loop:   ldi   olcnt, 224        ; load olcnt register
OLoop:  ldi   ilcnt, 237        ; load ilcnt register
ILoop:  dec   ilcnt             ; decrement ilcnt
  brne        ILoop             ; Continue InnerLoop
  dec         olcnt             ; decrementolcnt
  brne        OLoop             ; Continue OuterLoop
  dec         waitcnt           ; Decrementwait
  brne        Loop              ; Continue Waitloop

  pop         olcnt             ; Restore olcntregister
  pop         ilcnt             ; Restore ilcntregister
  pop         waitcnt           ; Restore waitregister
  ret                           ; Return fromsubroutine
```