ECE 375 Lab 1

Introduction to AVR Development Tools

**Lab Time: Monday 10-12**

Zachary DeVita

Fernando Robertson

# Introduction

The purpose of this lab is to become familiar with the primary software which will be used to edit assembly code for the duration of this course, as well as, learn to upload the code to the ATmega128 controller. This includes an introduction to both the AVRStudio and the Tekbots Universal Programmer applications. A simple introductory program, ‘BasicBumpBot.asm’, has been provided; the objective is to convert this assembly language to its .hex equivalent, then upload the program to the controller.

# Study Questions

1) What type of font is used? What size is the font?

The type of font used is ‘Calibri’, and the font size is ‘10’. Headings use a size of ‘16’, and both the fonts and the font sizes are different for the cover page.

2) What are pre-compiler directives? What is the difference between the .def and .equ directives?

A pre-compiler directive is a language construct that specifies how a compiler should process its input. These are not part of the language proper and do not perform any action in the language itself, but only make a change to the behavior of the compiler itself. These are executed before the code is compiled.

The directive ‘.def’ defines a symbolic name on a register, while ‘equ’ sets a symbol equal to an expression.

3) What is the 8-bit binary value that each of the following expressions evaluate to?

a) ( 1 << 2 ) = 4

b) ( 2 << 1 ) = 4

c) ( 4 >> 1 ) = 2

d) ( 1 << 4 ) = 16

e) ( 6 >> 1|1 << 6 ) = 67

# Challenge

For the challenge, the initial period of time that the wait loop was set to was 1 second, or, in assembly language, 100 tens-of-a-thousandths of a second. The objective was to double the period of time that the controller moved in the reverse direction prior to a left or right turn. Since the pre-compiler directive which held the period of time which the controller moved in reverse was additionally used for other operations for the controller, it was necessary to create a new pre-compiler directive. For the new directive, I simply multiplied the value held by the initial directive by 2.

.equ WTimeRevTurn = 2\*WTime ; Time in wait loop while reversing before turn

I then change the directive being used in left and right turn functions, but only for the wait times governing the “backward” movements.

# Source Code

/\*

\* Lab1.asm

\*

\* Created: 9/27/2016 9:43:52 AM

\* Author: devitaz

\*/

;\* 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: January 8, 2009

;\* 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 WTimeRevTurn = 2\*WTime ; Time in wait loop while reversing before turn

.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, WTimeRevTurn ; 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, WTimeRevTurn ; 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