ECE 375 Lab 1

Introduction to AVR Development Tools

Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

TA Signature

1 Introduction

This is the first Lab in the ECE 375 series and it covers the setup and compilation of an AVR Assembly Program. The student will learn how how to use the sample Basic Bump Bot assembly file and send the binaries to the AVR Microcontroller board. For the second part of the lab the student will be expected to download and compile the included C sample program and from it learn how to configure the I/O ports of the ATmega32U4 Microcontroller. The student will then write their own C program and upload it to the Microcontroller to verify that it runs as expected. The provided programs have been attached in the source code section of this report.

2 Design

As for part 1 of this lab assignment, no design needs to be done as the program is supplied. For part 2 of this lab assignment the C program was created to mimic the operaitons of the bump bot assembly file. Firstly the student must understand how the Bump Bot code must operate and they gain this information from the slides provided as they must program the right LED's to illuminate. For our program we decided that we wanted everything to be as readable as possible, thus we created constants for each of the LED directional cues.

3 Assembly Overview

As for the Assembly program an overview can be seen below

3.1 Internal Register Definitions and Constants

Four different registers have been setup, those being the multipurpose register (mpr), the wait counter register (waitcnt), and two loop counters, for counting the cycles of the delay function. In addition to these, there are several different constants. WTime defines the time in milliseconds to wait inside the wait loop. The rest of the defined constants are either input bits, engine enable bits, or engine direction bits.

3.2 Initialization Routine

The initialization routine sets up several important ports and pointers that allow the rest of the assembly to work. Firstly the stack pointer is initialized at the end of RAM so that when the program pushes and pops items into and out of it, the stack does not interfere with any other data. Port B is then initialized for output, and Port D is initialized for input. The move forward command is also in this phase, to give a default movement type.

3.3 Main Routine

The main program constantly checks for if either of the whisker buttons have been hit, by reading the input of the PIND. When one of the whiskers is hit, the correct subroutine is called. As long as no button is hit the bump bot will continue in a straight line.

3.4 Subroutines

3.4.1 Hit Right

The HitRight subroutine describes what happens when the right whisker bit is triggered. The robot will move backwards for a second, then turn left for a second, then it will continue forward.

3.4.2 Hit Left

The HitLeft subroutine describes what happens when the left whisker bit is triggered. First the bump bot will move backwards for a second, then it will turn right for a second, then it will continue forward.

3.4.3 Wait

The Wait subroutine controls the wait intervals while the bump bot is preforming an action. Due to each clock cycle taking a measurable amount of time, we can calculate how many times we need to loop for. This function used the olcnt and ilcnt to have two nested loops, running the dec command until they equal zero, thus waiting the requested amount of time.

4 C Program Overview

Each of the methods determined to operate the bump bot can be seen in the code section at the end of this report, their descriptions are here.

4.1 Definitions and Constants

Several different constant integer values are prescribed on lines 29 - 33

4.2 Main Method

Text goes here

4.3 Functions

4.3.1 Hit Right

This is just an example.

4.3.2 Hit Left

Replace with your owns.

5 Testing

Text and Figures go here.

Case	Expected	Actual meet expected

6 Additional Questions

1. The text of the question

   The text of the answer

2. The text of the question

   (a) Text of the first part of the answer

   (b) Text of the second part of the answer

7 Difficulties

Text goes here

8 Conclusion

Text goes here

9 Source Code

Listing 1: Assembely Bump Bot Script

```
;
; Lab1_Sourcecode.asm
;
; Created: 1/13/2023 12:15:20 PM
; Author : Astrid Delestine and Lucas Plaisted!
;

;***************************************************
;*
;*   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: David Zier, Mohammed Sinky, and Dongjun Lee
;*                          (modification August 10, 2022)
;*      Date: August 10, 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 modifictions
;*   -   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.
;/////////////////////////////////////////////////////////////
```

```
64
65  .equ    MovFwd = (1<<EngDirR|1<<EngDirL)    ; Move Forward Command
66  .equ    MovBck = $00                        ; Move Backward Command
67  .equ    TurnR = (1<<EngDirL)                ; Turn Right Command
68  .equ    TurnL = (1<<EngDirR)                ; Turn Left Command
69  .equ    Halt = (1<<EngEnR|1<<EngEnL)        ; Halt Command
70
71  ;=====================================================================
72  ; NOTE: Let me explain what the macros above are doing.
73  ; Every macro is executing in the pre-compiler stage before
74  ; the rest of the code is compiled.  The macros used are
75  ; left shift bits (<<) and logical or (|).  Here is how it
76  ; works:
77  ;   Step 1.  .equ   MovFwd = (1<<EngDirR|1<<EngDirL)
78  ;   Step 2.     substitute constants
79  ;            .equ   MovFwd = (1<<4|1<<7)
80  ;   Step 3.     calculate shifts
81  ;            .equ   MovFwd = (b00010000|b10000000)
82  ;   Step 4.     calculate logical or
83  ;            .equ   MovFwd = b10010000
84  ; Thus MovFwd has a constant value of b10010000 or $90 and any
85  ; instance of MovFwd within the code will be replaced with $90
86  ; before the code is compiled.  So why did I do it this way
87  ; instead of explicitly specifying MovFwd = $90?  Because, if
88  ; I wanted to put the Left and Right Direction Bits on different
89  ; pin allocations, all I have to do is change thier individual
90  ; constants, instead of recalculating the new command and
91  ; everything else just falls in place.
92  ;=====================================================================
93
94  ;*********************************************************
95  ;* Beginning of code segment
96  ;*********************************************************
97  .cseg
98
99  ;---------------------------------------------------------------------
100 ; Interrupt Vectors
101 ;---------------------------------------------------------------------
102 .org    $0000               ; Reset and Power On Interrupt
103         rjmp    INIT        ; Jump to program initialization
104
105 .org    $0056               ; End of Interrupt Vectors
106 ;---------------------------------------------------------------------
107 ; Program Initialization
108 ;---------------------------------------------------------------------
109 INIT:
```

```
110     ; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
111         ldi     mpr, low(RAMEND)
112         out     SPL, mpr        ; Load SPL with low byte of RAMEND
113         ldi     mpr, high(RAMEND)
114         out     SPH, mpr        ; Load SPH with high byte of RAMEND
115
116     ; Initialize Port B for output
117         ldi     mpr, $FF        ; Set Port B Data Direction Register
118         out     DDRB, mpr       ; for output
119         ldi     mpr, $00        ; Initialize Port B Data Register
120         out     PORTB, mpr      ; so all Port B outputs are low
121
122     ; Initialize Port D for input
123         ldi     mpr, $00        ; Set Port D Data Direction Register
124         out     DDRD, mpr       ; for input
125         ldi     mpr, $FF        ; Initialize Port D Data Register
126         out     PORTD, mpr      ; so all Port D inputs are Tri-State
127
128         ; Initialize TekBot Forward Movement
129         ldi     mpr, MovFwd     ; Load Move Forward Command
130         out     PORTB, mpr      ; Send command to motors
131
132 ;-----------------------------------------------------------
133 ; Main Program
134 ;-----------------------------------------------------------
135 MAIN:
136         in      mpr, PIND       ; Get whisker input from Port D
137         andi    mpr, (1<<WskrR|1<<WskrL)
138         cpi     mpr, (1<<WskrL) ; Check for Right Whisker input
139                                 ;(Recall Active Low)
140         brne    NEXT            ; Continue with next check
141         rcall   HitRight        ; Call the subroutine HitRight
142         rjmp    MAIN            ; Continue with program
143 NEXT:   cpi     mpr, (1<<WskrR) ; Check for Left Whisker input
144                                 ;(Recall Active Low)
145         brne    MAIN            ; No Whisker input, continue program
146         rcall   HitLeft         ; Call subroutine HitLeft
147         rjmp    MAIN            ; Continue through main
148
149 ;*****************************************************************
150 ;* Subroutines and Functions
151 ;*****************************************************************
152
153 ;-----------------------------------------------------------
154 ; Sub:   HitRight
155 ; Desc: Handles functionality of the TekBot when the right whisker
```

```
156 ;       is triggered.
157 ;-----------------------------------------------------------------
158 HitRight:
159         push    mpr         ; Save mpr register
160         push    waitcnt         ; Save wait register
161         in      mpr, SREG   ; Save program state
162         push    mpr         ;
163
164         ; Move Backwards for a second
165         ldi     mpr, MovBck ; Load Move Backward command
166         out     PORTB, mpr  ; Send command to port
167         ldi     waitcnt, (WTime<<1) ; Shifted bit back by 1,
168                                     ; making the wait time two seconds
169         rcall   Wait            ; Call wait function
170
171         ; Turn left for a second
172         ldi     mpr, TurnL  ; Load Turn Left Command
173         out     PORTB, mpr  ; Send command to port
174         ldi     waitcnt, WTime  ; Wait for 1 second
175         rcall   Wait            ; Call wait function
176
177         ; Move Forward again
178         ldi     mpr, MovFwd ; Load Move Forward command
179         out     PORTB, mpr  ; Send command to port
180
181         pop     mpr     ; Restore program state
182         out     SREG, mpr   ;
183         pop     waitcnt     ; Restore wait register
184         pop     mpr     ; Restore mpr
185         ret             ; Return from subroutine
186
187 ;-----------------------------------------------------------------
188 ; Sub:  HitLeft
189 ; Desc: Handles functionality of the TekBot when the left whisker
190 ;       is triggered.
191 ;-----------------------------------------------------------------
192 HitLeft:
193         push    mpr         ; Save mpr register
194         push    waitcnt         ; Save wait register
195         in      mpr, SREG   ; Save program state
196         push    mpr         ;
197
198         ; Move Backwards for a second
199         ldi     mpr, MovBck ; Load Move Backward command
200         out     PORTB, mpr  ; Send command to port
201         ldi     waitcnt, (WTime<<1) ; Wait for 1 second
```

```
202             rcall   Wait            ; Call wait function
203
204             ; Turn right for a second
205             ldi     mpr, TurnR      ; Load Turn Left Command
206             out     PORTB, mpr      ; Send command to port
207             ldi     waitcnt, WTime  ; Wait for 1 second
208             rcall   Wait            ; Call wait function
209
210             ; Move Forward again
211             ldi     mpr, MovFwd     ; Load Move Forward command
212             out     PORTB, mpr      ; Send command to port
213
214             pop     mpr             ; Restore program state
215             out     SREG, mpr       ;
216             pop     waitcnt         ; Restore wait register
217             pop     mpr             ; Restore mpr
218             ret                     ; Return from subroutine
219
220 ;-----------------------------------------------------------------
221 ; Sub:   Wait
222 ; Desc: A wait loop that is 16 + 159975*waitcnt cycles or roughly
223 ;       waitcnt*10ms.  Just initialize wait for the specific amount
224 ;       of time in 10ms intervals. Here is the general eqaution
225 ;       for the number of clock cycles in the wait loop:
226 ;           (((((3*ilcnt)-1+4)*olcnt)-1+4)*waitcnt)-1+16
227 ;-----------------------------------------------------------------
228 Wait:
229             push    waitcnt         ; Save wait register
230             push    ilcnt           ; Save ilcnt register
231             push    olcnt           ; Save olcnt register
232
233 Loop:       ldi     olcnt, 224      ; load olcnt register
234 OLoop:      ldi     ilcnt, 237      ; load ilcnt register
235 ILoop:      dec     ilcnt           ; decrement ilcnt
236             brne    ILoop           ; Continue Inner Loop
237             dec     olcnt       ; decrement olcnt
238             brne    OLoop           ; Continue Outer Loop
239             dec     waitcnt     ; Decrement wait
240             brne    Loop            ; Continue Wait loop
241
242             pop     olcnt       ; Restore olcnt register
243             pop     ilcnt       ; Restore ilcnt register
244             pop     waitcnt     ; Restore wait register
245             ret             ; Return from subroutine
```

Listing 2: C Bump Bot Script

```c
/*
 * Lab1C.c
 *
 * Created: 1/14/2023 12:51:47 PM
 * Author : Astrid Delestine and Lucas Plaisted
 */

/*
This code will cause a TekBot connected to the AVR board to
move forward and when it touches an obstacle, it will reverse
and turn away from the obstacle and resume forward motion.

PORT MAP
Port B, Pin 5 -> Output -> Right Motor Enable
Port B, Pin 4 -> Output -> Right Motor Direction
Port B, Pin 6 -> Output -> Left Motor Enable
Port B, Pin 7 -> Output -> Left Motor Direction
Port D, Pin 5 -> Input -> Left Whisker
Port D, Pin 4 -> Input -> Right Whisker
*/

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>

// Led final integer values

const int FORWARD = 0b10010000,
HALT = 0b11110000,
BACKWARD = 0b00000000,
 RIGHT = 0b00010000,
 LEFT = 0b10000000;

void BotActionL();
void BotActionR();
void goBackwards2Sec();

int main(void)
{
    DDRB = 0b11110000; // set 7-4th bits as outputs
    //PORTB = 0b01100000; // turn on LEDs connected to 5-6th bits
    DDRD = 0b00000000; // set 5th and 4th pins on D as inputs
    PORTD = 0b11110000; //enable pull up resistors for port D pins 7-4

```

```
46
47      while (1) // loop forever
48      {
49          // read and extract only 4-5 th bit
50          uint8_t mpr = PIND & 0b00110000;
51          mpr = ~mpr; //flip bits since PINDD is active low
52          if (mpr & 0b00010000) // check if the right whisker is hit
53          {
54              BotActionR(); // call BotAction
55          }
56          else if (mpr & 0b00100000) // check if the left whisker is hit
57          {
58              BotActionL(); // call BotAction
59          }
60          PORTB = FORWARD; //resume forward movement
61          _delay_ms(50); //delay for 50ms to help prevent switch bouncing
62      }
63  }
64
65
66  void BotActionL(){
67      goBackwards2Sec(); //self explanatory
68      //left motor forwards, right motor backwards = turn right
69      PORTB = LEFT;
70      _delay_ms(1000); //wait 1 second
71      return;
72  }
73
74  void BotActionR(){
75      goBackwards2Sec(); //self explanatory :)
76      //right motor forwards, left motor backwards = turn left
77      PORTB = RIGHT;
78      _delay_ms(1000); //wait 1 second
79      return;
80  }
81
82  void goBackwards2Sec(){
83      PORTB = BACKWARD; //turn both motors to reverse
84      _delay_ms(2000); //delay for 2 seconds
85      return;
86  }
```