ECE 375 Lab 1

Introduction to AVR Tools

Lab Time: Friday 2 PM - 3:50 PM

Student 1: Winnie Woo
Student 2: Joseph Borisch

TA Signature

1 Additional Questions

1. Define pre-compiler directive. What is the difference between the .def and .equ directives?

   According to the AVR starter guide, pre-compiler directives are special instructions that are executed before the code is compiled and directs the compiler.

   .def directive is defining a symbolic name on a which can be redefined late in the program and .equ directive is setting a symbol equal to an expression and is a constant.

2. Determine the 8-bit binary value that each of the following expressions evaluates to.

   (a) (1 << 5) = 00100000

   (b) (4 << 4) = 01000000

   (c) (8 >> 1) = 00000100

   (d) (5 << 0) = 00000101

   (e) (8 >> 2|1 << 6) = (00001010|01000000) = (01001010)

3. Describe the instructions listed below. ADIW, BCLR, BRCC, BRGE, COM, EOR, LSL, LSR, NEG, OR, ORI, ROL, ROR, SBC, SBIW, and SUB

   (a) **arithmetic and logic instructions: addition**

   (b) **set and clear respectively any bit within the SREG register**

   (c) **modify the PC if the corresponding condition is met**

   (d) **modify the PC if the corresponding condition is met**

   (e) **arithmetic and logic instructions: compliments**

   (f) **arithmetic and logic instructions: logic**

   (g) **logical shift left**

   (h) **logical shift right**

   (i) **arithmetic and logic instructions: compliments**

   (j) **arithmetic and logic instructions: logic**

   (k) **arithmetic and logic instructions: logic**

   (l) **rotate left through carry**

   (m) **rotate right through carry**

   (n) **arithmetic and logic instructions: subtraction**

   (o) **arithmetic and logic instructions: subtraction**

   (p) **arithmetic and logic instructions: subtraction**

2 Source Code

```
{
;***********************************************************
;*
;* BasicBumpBot.asm -V3.0
;*
;* This program contains the necessary 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: Winnie Woo and Joseph Borisch
;* Date: October 4th, 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 BWTime = 200 ; Time to back up

.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 their 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, BWTime ; 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, 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, BWTime ; 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, 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
}
```