Lab 8

Namtalay Laorattanavech – Lab time: Tue 5pm

Samuel Lee – Lab time: Thurs 10am

INTRODUCTION

The purpose of this lab is to make us get used to using Timer and Counter in assembly by making a program that can control the speed of the Tekbot. This can be done by implement the the timer inside the program.

The purpose of this lab is to make us get used to using USART to send signals between ATmega128 microcontrollers. Also, this lab requires us to implement all the labs and what we have learned so far in the class to our programs.

PROGRAM OVERVIEW

There are two ATmega128 microcontrollers, each of them do the different job. The first board will act like a remote control and the second board will act like a robot. The remote control can make the robot move forward, backward, left, and right. Moreover, the remote control can also config the speed of the robot, similar to lab 7. All outputs will be shown using LEDs on the board. The Input will be indicated by buttons.

Remote control: Initialization Routine

In this routine, we initialize stack pointers, interrupts, I/O ports B and D, and the USART settings. The baud rate is set to 2400bps with the value of 416.

Remote control: MAIN Routine

The main function simply check the input for the certain value, if the certain value is received from the input (the button is pressed) it jumps to the transmit action functions. If the 7^{th} bit of PIND is zero, it jumps the transmit turn right function. If the 6^{th} bit of PIND is zero, it jumps the transmit turn left function. If the 5^{th} bit of PIND is zero, it jumps the transmit moving forward function. If the 4^{th} bit of PIND is zero, it jumps the transmit moving backward function. If the 3^{rd} bit of PIND is zero, it jumps the transmit speed max function. If the 2^{nd} bit of PIND is zero, it jumps the transmit speed min function. If the 1^{st} bit of PIND is zero, it jumps the transmit speed up function. If the 0^{th} bit of PIND is zero, it jumps the transmit speed down function.

REMOTE CONTROL: TRANSMIT_R ROUTINE

This is the function that transmits the turn right command. There are two mini functions in TRANSMIT_R: LOOP1, and LOOP2. These two mini functions need to be called before we transmit any data via UDR1. TRANSMIT_R function will first send the bot address, wait a bit, and then send out the action command using USART.

REMOTE CONTROL: TRANSMIT_L ROUTINE

This is the function that transmits the turn left command. There are two mini functions in TRANSMIT_L: LOOP1, and LOOP2. These two mini functions need to be called before we transmit any data via UDR1. TRANSMIT_L function will first send the bot address, wait a bit, and then send out the action command using USART.

REMOTE CONTROL: TRANSMIT_FWD ROUTINE

This is the function that transmits the move forward command. There are two mini functions in TRANSMIT_FWD: LOOP1, and LOOP2. These two mini functions need to be called before we transmit any data via UDR1. TRANSMIT_FWD function will first send the bot address, wait a bit, and then send out the action command using USART.

REMOTE CONTROL: TRANSMIT_BCK ROUTINE

This is the function that transmits the move backward command. There are two mini functions in TRANSMIT_BCK: LOOP1, and LOOP2. These two mini functions need to be called before we transmit any data via UDR1. TRANSMIT_BCK function will first send the bot address, wait a bit, and then send out the action command using USART.

REMOTE CONTROL: TRANSMIT_SPD_UP ROUTINE

This is the function that transmits the speed up command. There are two mini functions in TRANSMIT_SPD_UP: LOOP1, and LOOP2. These two mini functions need to be called before we transmit any data via UDR1. TRANSMIT_SPD_UP function will first send the bot address, wait a bit, and then send out the action command using USART.

REMOTE CONTROL: TRANSMIT_SPD_DOWN ROUTINE

This is the function that transmits the speed down command. There are two mini functions in TRANSMIT_SPD_DOWN: LOOP1, and LOOP2. These two mini functions need to be called before we transmit any data via UDR1. TRANSMIT_SPD_DOWN function will first send the bot address, wait a bit, and then send out the action command using USART.

REMOTE CONTROL: TRANSMIT_SPD_MAX ROUTINE

This is the function that transmits the speed up command. There are two mini functions in TRANSMIT_SPD_UP: LOOP1, and LOOP2. These two mini functions need to be called before we transmit any data via UDR1. TRANSMIT_SPD_UP function will first send the bot address, wait a bit, and then send out the action command using USART.

REMOTE CONTROL: TRANSMIT_SPD_MAX ROUTINE

This is the function that transmits the speed up command. There are two mini functions in TRANSMIT_SPD_UP: LOOP1, and LOOP2. These two mini functions need to be called before we transmit any data via UDR1. TRANSMIT_SPD_UP function will first send the bot address, wait a bit, and then send out the action command using USART.

ROBOT: Initialization Routine

In this routine, we initialize stack pointers, interrupts, I/O ports B and D, and the USART settings. The baud rate is set to 2400bps with the value of 416. This is basically the same as the remote control except that we also need to initialize and set the timer. The timers R0 and R1 are set to fast PWM mode.

ROBOT: MAIN Routine

There is nothing inside the main function since we use interrupts to detect the input.

ROBOT: USART_Receive

This function first receives the data from UDR1. The first data received is the bot address. The function will check if that the bot address is matched or not. If not, the function will just return back to the main. Else the function continues. Then, the function will receive another data from UDR1 which is the action command. If the command is forward, backward, left, or right, it will just execute that command and send the result to the LEDs. If the command is speed up, down, min, or max, the function will branch to specific functions that handle those command (INPUT0-3).

ROBOT: INPUT0

This is the function for minimum speed. It will set the speed and speed level of the Tekbot to zero, load the speed level to the timers. Then upload that value to PORT B, which will end up on the LEDs.

ROBOT: INPUT1

This is the function for max speed. It will set the speed and speed level of the Tekbot to max (15), load the speed level to the timers. Then upload that value to PORT B, which will end up on the LEDs.

ROBOT: INPUT2

This is the function for decrease the speed. It will decrement the speed and speed level of the Tekbot by one, load the speed level to the timers. Then upload that value to PORT B, which will end up on the LEDs.

ROBOT: INPUT3

This is the function for increase the speed. It will increment the speed and speed level of the Tekbot by one, load the speed level to the timers. Then upload that value to PORT B, which will end up on the LEDs.

ROBOT: HitRight

This is the function was from previous lab. It basically using interrupts to check if the right whisker is triggered or not (in this case a button). If it is triggered, the bot will move back a bit then turn left. The queue is also cleared before exit the function to prevent the unintentional repeat inputs.

ROBOT: HitLeft

This is the function was from previous lab. It basically using interrupts to check if the left whisker is triggered or not (in this case a button). If it is triggered, the bot will move back a bit then turn right. The queue is also cleared before exit the function to prevent the unintentional repeat inputs.

Wait Routine

This is a function that was given in lab1. It is used to create a wait time in between receiving input signal in order to prevent receiving unintentional multiple inputs.

SOURCE CODE: REMOTE CONTROL

```
;***********************************************************
;*
;*     Lab 8 TX - REMOTE
;*
;*
;***********************************************************
;*
;*      Author: Samuel Jia Khai Lee & Namtalay Laorattanavech
;*        Date: 3/13/2017
;*
;***********************************************************

.include "m128def.inc"                  ; Include definition file

;***********************************************************
;*     Internal Register Definitions and Constants
;***********************************************************
.def   mpr = r16                        ; Multi-Purpose Register
.def   mpr2 = r17
.def   waitcnt = r18              ; Wait Loop Counter
.def   ilcnt = r19                      ; Inner Loop Counter
.def   olcnt = r20                      ; Outer Loop Counter

.equ   WTime = 100                      ; Time to wait in wait loop
.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

; Use these action codes between the remote and robot
; MSB = 1 thus:
.equ   BotAddress = 0b11111111
; control signals are shifted right by one and ORed with 0b10000000 = $80
.equ   MovFwd =  ($80|1<<(EngDirR-1)|1<<(EngDirL-1))    ;0b10110000 Move Forward Action
Code
.equ   MovBck =  ($80|$00)
       ;0b10000000 Move Backward Action Code
.equ   TurnR =   ($80|1<<(EngDirL-1))                                ;0b10100000 Turn
Right Action Code
.equ   TurnL =   ($80|1<<(EngDirR-1))                                ;0b10010000 Turn
Left Action Code
.equ   SpeedUp = ($80|1<<(EngDirL))                                  ;0b11000000 Speed
Up Action Code
.equ   SpeedDown = (1<<(EngDirR-1)|1<<(EngDirL-1))            ;0b00110000 Speed Down
Action Code
.equ   SpeedMax = ($80|1<<(EngDirL-1))                               ;0b10100000 Speed
Max Action Code
.equ   SpeedMin = (1<<(EngDirR-1)|1<<(EngDirL))        ;0b01010000 Speed Min Action
Code

;***********************************************************
;*     Start of Code Segment
;***********************************************************
.cseg                                           ; Beginning of code segment

;***********************************************************
;*     Interrupt Vectors
;***********************************************************
.org   $0000                            ; Beginning of IVs
              rjmp    INIT              ; Reset interrupt
```

```
.org    $0046                           ; End of Interrupt Vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:
        ;Stack Pointer (VERY IMPORTANT!!!!)
        ldi     mpr, high(RAMEND)
        out     SPH, mpr
        ldi     mpr, low(RAMEND)
        out     SPL, mpr
        ;I/O Ports
        ldi             mpr, $FF            ;Set Port B Data Direction Regisiter
        out             DDRB, mpr
        ldi             mpr, $00            ;Initialize Port B Data Register
        out             PORTB, mpr

        ldi             mpr, $00            ;Set Port D Data Direction Register
        out             DDRD, mpr
        ldi             mpr, $FF            ;Initialize Port D Data Register
        out             PORTD, mpr
        ;USART1
                ldi     mpr, (1<<U2X1)              ;Set double data rate
                sts     UCSR1A, mpr
                ;Set baudrate at 2400bps
                ldi     mpr, high(416)      ; Load high byte of 0x0340
                sts     UBRR1H, mpr   ; UBRR1H in extended I/O space
                ldi     mpr, low(416)       ; Load low byte of 0x0340
                sts     UBRR1L, mpr

                ;Enable receiver and enable receive interrupts
                ldi     mpr, (1<<TXEN1 | 1<<RXEN1)
                sts     UCSR1B, mpr

                ;Set frame format: 8 data bits, 2 stop bits
                ldi     mpr, (1<<UCSZ10)|(1<<UCSZ11)|(1<<USBS1)|(1<<UPM01)
                sts     UCSR1C, mpr         ; UCSR0C in extended I/O space

        ;External Interrupts
                ;Set the External Interrupt Mask
                ldi             mpr, (1<<INT0) | (1<<INT1)
                out             EIMSK, mpr

        clr mpr
        out PORTB, mpr


;***********************************************************
;*      Main Program
;***********************************************************
MAIN:
                in      mpr, PIND

                sbrs    mpr, 7                  ;Check if each button/bit is cleared
                rjmp    TRANSMIT_R              ;Then go to respective functions to
transmit Bot Address and Action Code
                sbrs    mpr, 6
                rjmp    TRANSMIT_L
                sbrs    mpr, 5
                rjmp    TRANSMIT_FWD
                sbrs    mpr, 4
```

```
                rjmp    TRANSMIT_BCK
                sbrs    mpr, 3
                rjmp    TRANSMIT_SPD_MAX
                sbrs    mpr, 2
                rjmp    TRANSMIT_SPD_MIN
                sbrs    mpr, 1
                rjmp    TRANSMIT_SPD_UP
                sbrs    mpr, 0
                rjmp    TRANSMIT_SPD_DOWN

                ldi             mpr, (1<<INT0 | 1<<INT1)    ;Clean Queue
                out             EIFR, mpr

                rjmp    MAIN

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

TRANSMIT_R:                                             ;Transmit Bot Address using
UDR1 to be checked for Transmit Right Function
                ldi             mpr2, (1<<7)        ;if transmitter and receiver have the
same address
                out     PORTB, mpr2
                ldi     mpr, BotAddress
                sts     UDR1, mpr
                ldi     waitcnt, 55
                rcall   Wait

TRANSMIT_R_LOOP1:                               ;Transmit Bot address and check if
USART Data Register Empty is cleared
                lds     mpr, UCSR1A
                sbrs    mpr, UDRE1
                rcall   TRANSMIT_R_LOOP1

                ldi     mpr, TurnR
                sts     UDR1, mpr
                ldi             waitcnt, 250
                rcall   Wait

TRANSMIT_R_LOOP2:
                lds     mpr, UCSR1A
                sbrs    mpr, UDRE1
                rjmp    TRANSMIT_R_LOOP2

                rjmp    MAIN

;***********************************************************

TRANSMIT_L:                                             ;Transmit Bot Address using
UDR1 to be checked for Transmit Left Function
                ldi             mpr2, (1<<6)        ;if transmitter and receiver have the
same address
                out     PORTB, mpr2
                ldi     mpr, BotAddress
                sts     UDR1, mpr
                ldi     waitcnt, 55
                rcall   Wait

TRANSMIT_L_LOOP1:
                lds     mpr, UCSR1A
                sbrs    mpr, UDRE1
```

```
              rjmp    TRANSMIT_L_LOOP1

              ldi     mpr, TurnL
              sts     UDR1, mpr
              ldi             waitcnt, 250
              rcall   Wait
TRANSMIT_L_LOOP2:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
              rjmp    TRANSMIT_L_LOOP2

              rjmp    MAIN

;***********************************************************

TRANSMIT_FWD:                                 ;Transmit Bot Address using UDR1 to be
checked for Transmit Forward Function
              ldi             mpr2, (1<<5)    ;if transmitter and receiver have the
same address
              out     PORTB, mpr2
              ldi     mpr, BotAddress
              sts     UDR1, mpr
              ldi     waitcnt, 55
              rcall   Wait

TRANSMIT_FWD_LOOP1:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
              rjmp    TRANSMIT_FWD_LOOP1

              ldi     mpr, MovFwd
              sts     UDR1, mpr
              ldi             waitcnt, 250
              rcall   Wait
TRANSMIT_FWD_LOOP2:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
              rjmp    TRANSMIT_FWD_LOOP2

              rjmp    MAIN

;***********************************************************

TRANSMIT_BCK:                                 ;Transmit Bot Address using UDR1 to be
checked for Transmit Backward Function
              ldi             mpr2, (1<<4)    ;if transmitter and receiver have the
same address
              out     PORTB, mpr2
              ldi     mpr, BotAddress
              sts     UDR1, mpr
              ldi     waitcnt, 55
              rcall   Wait

TRANSMIT_BCK_LOOP1:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
              rjmp    TRANSMIT_BCK_LOOP1

              ldi     mpr, MovBck
              sts     UDR1, mpr
              ldi             waitcnt, 250
              rcall   Wait
```

```
TRANSMIT_BCK_LOOP2:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
              rjmp    TRANSMIT_BCK_LOOP2

              rjmp    MAIN

;***********************************************************

TRANSMIT_SPD_UP:                              ;Transmit Bot Address using UDR1 to be
checked for Transmit Speed Up Function
              ldi           mpr2, (1<<1)      ;if transmitter and receiver have the
same address
              out     PORTB, mpr2
              ldi     mpr, BotAddress
              sts     UDR1, mpr
              ldi     waitcnt, 55
              rcall   Wait

TRANSMIT_SPD_UP_LOOP1:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
              rcall   TRANSMIT_SPD_UP_LOOP1

              ldi     mpr, SpeedUp
              sts     UDR1, mpr

              ldi           waitcnt, 250
              rcall   Wait

TRANSMIT_SPD_UP_LOOP2:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
              rjmp    TRANSMIT_SPD_UP_LOOP2

              rjmp    MAIN

;***********************************************************

TRANSMIT_SPD_DOWN:                            ;Transmit Bot Address using UDR1 to be
checked for Transmit Speed Down Function
              ldi           mpr2, (1<<0)      ;if transmitter and receiver have the
same address
              out     PORTB, mpr2
              ldi     mpr, BotAddress
              sts     UDR1, mpr
              ldi     waitcnt, 55
              rcall   Wait

TRANSMIT_SPD_DOWN_LOOP1:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
              rcall   TRANSMIT_SPD_DOWN_LOOP1

              ldi     mpr, SpeedDown
              sts     UDR1, mpr
              ldi           waitcnt, 250
              rcall   Wait

TRANSMIT_SPD_DOWN_LOOP2:
              lds     mpr, UCSR1A
              sbrs    mpr, UDRE1
```

```
		rjmp	TRANSMIT_SPD_DOWN_LOOP2

		rjmp	MAIN

;***********************************************************

TRANSMIT_SPD_MAX:				;Transmit Bot Address using UDR1 to be
checked for Transmit Speed Max Function
		ldi	mpr, BotAddress		;if transmitter and receiver have the
same address
		sts	UDR1, mpr
		ldi	waitcnt, 55
		rcall	Wait

TRANSMIT_SPD_MAX_LOOP1:
		lds	mpr, UCSR1A
		sbrs	mpr, UDRE1
		rcall	TRANSMIT_SPD_MAX_LOOP1

		ldi	mpr, SpeedMax
		sts	UDR1, mpr
		ldi		mpr2, (1<<3)
		out	PORTB, mpr2
		ldi		waitcnt, 250
		rcall	Wait

TRANSMIT_SPD_MAX_LOOP2:
		lds	mpr, UCSR1A
		sbrs	mpr, UDRE1
		rjmp	TRANSMIT_SPD_MAX_LOOP2

		rjmp	MAIN

;***********************************************************

TRANSMIT_SPD_MIN:				;Transmit Bot Address using UDR1 to be
checked for Transmit Speed Min Function
		ldi		mpr2, (1<<2)	;if transmitter and receiver have the
same address
		out	PORTB, mpr2
		ldi	mpr, BotAddress
		sts	UDR1, mpr
		ldi	waitcnt, 55
		rcall	Wait

TRANSMIT_SPD_MIN_LOOP1:
		lds	mpr, UCSR1A
		sbrs	mpr, UDRE1
		rcall	TRANSMIT_SPD_MIN_LOOP1

		ldi	mpr, SpeedMin
		sts	UDR1, mpr
		ldi		waitcnt, 250
		rcall	Wait

TRANSMIT_SPD_MIN_LOOP2:
		lds	mpr, UCSR1A
		sbrs	mpr, UDRE1
		rjmp	TRANSMIT_SPD_MIN_LOOP2

		rjmp	MAIN
;***********************************************************
```

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

SOURCE CODE: ROBOT

```
;***********************************************************
;*
;*     Lab 8 RX - ROBOT
;*
;*
;***********************************************************
;*
;*      Author: Samuel Jia Khai Lee & Namtalay Laorattanavech
;*        Date: 3/13/2017
;*
;***********************************************************

.include "m128def.inc"                    ; Include definition file

;***********************************************************
;*     Internal Register Definitions and Constants
;***********************************************************
.def   mpr = r16                          ; Multi-Purpose Register
.def   mpr2 = r17
.def   waitcnt = r18               ; Wait Loop Counter
.def   ilcnt = r19                        ; Inner Loop Counter
.def   olcnt = r20                        ; Outer Loop Counter
.def   speed = r21
.def   speed_level = r22

.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

.equ   BotAddress = 0b11111111

;/////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
;/////////////////////////////////////////////////////////////
.equ   MovFwd =  (1<<EngDirR|1<<EngDirL)  ;0b01100000 Move Forward Action Code
.equ   MovBck =  $00                                ;0b00000000 Move Backward
Action Code
.equ   TurnR =   (1<<EngDirL)                       ;0b01000000 Turn Right Action
Code
.equ   TurnL =   (1<<EngDirR)                       ;0b00100000 Turn Left Action
Code
.equ   Halt =    (1<<EngEnR|1<<EngEnL)          ;0b10010000 Halt Action Code

;***********************************************************
;*     Start of Code Segment
;***********************************************************
.cseg                                           ; Beginning of code segment

;***********************************************************
;*     Interrupt Vectors
;***********************************************************
.org   $0000                            ; Beginning of IVs
              rjmp   INIT               ; Reset interrupt

;Should have Interrupt vectors for:
;- Left whisker
;- Right whisker
;- USART receive

.org   $0002
              rcall HitRight
              reti

.org   $0004
              rcall HitLeft
              reti

.org   $0003C
              rcall USART_Receive
              reti

.org   $0046                            ; End of Interrupt Vectors

;***********************************************************
;*     Program Initialization
;***********************************************************
INIT:
       ;Stack Pointer (VERY IMPORTANT!!!!)
       ldi    mpr, high(RAMEND)
       out    SPH, mpr
       ldi    mpr, low(RAMEND)
       out    SPL, mpr
       ;I/O Ports
       ldi           mpr, $FF             ;Set Port B Data Direction Regisiter
       out           DDRB, mpr
       ldi           mpr, $00             ;Initialize Port B Data Register
       out           PORTB, mpr
```

```
        ldi             mpr, $00            ;Set Port D Data Direction Register
        out             DDRD, mpr
        ldi             mpr, $FF            ;Initialize Port D Data Register
        out             PORTD, mpr
        ;USART1
                ldi     mpr, (1<<U2X1)
                sts     UCSR1A, mpr
                ;Set baudrate at 2400bps
                ldi     mpr, high(416)      ; Load high byte of 0x0340
                sts     UBRR1H, mpr   ; UBRR0H in extended I/O space
                ldi     mpr, low(416)       ; Load low byte of 0x0340
                sts     UBRR1L, mpr

                ;Enable receiver and enable receive interrupts
                ldi     mpr, (1<<RXEN1 | 1<<TXEN1 | 1<<RXCIE1)
                sts     UCSR1B, mpr

                ;Set frame format: 8 data bits, 2 stop bits
                ldi     mpr, (0<<UMSEL1 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
                sts     UCSR1C, mpr         ; UCSR0C in extended I/O space


        ;External Interrupts
                ;Set the External Interrupt Mask
                ldi             mpr, (1<<INT0) | (1<<INT1)
                out             EIMSK, mpr
                ;Set the Interrupt Sense Control to falling edge detection
                ldi             mpr, (1<<ISC01) | (0<<ISC00) | (1<<ISC11) | (0<<ISC10)
                sts             EICRA, mpr          ;Use sts, EICRA in extended I/O space

        ; Configure 8-bit Timer/Counters
                ldi             mpr, 0b01010000     ; Fast PWM w/ toggle
                out             TCCR0, mpr          ;
                ldi             mpr, 0b01010000
                out             TCCR2, mpr          ;

                ;default state
                ldi             speed_level, 0                  ;Initialize Speed level
and clock
                out             OCR0, speed_level
                out             OCR2, speed_level

                ldi             speed, 0            ;Initialize Speed
                andi    mpr, $F0            ;
                or              mpr, speed          ;
                out             PORTB, mpr          ;

        ; Initialize Fwd Movement
                ldi     mpr, MovFwd
                out     PORTB, mpr

                sei
        ;Other

;***********************************************************
;*      Main Program
;***********************************************************
MAIN:
                rjmp MAIN


;***********************************************************
```

```
;*     Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------
; Sub: USART_Receive
; Desc:       Receive USART Command from Transmitter
;-----------------------------------------------------------
USART_Receive:
              lds    mpr, UDR1                  ;Read Bot Address and compare
              ldi           mpr2, BotAddress     ;if Bot Address is incorrect, then
return interrupt
              cpse          mpr, mpr2                   ;aka do nothing, else continue
              ret
              ldi           waitcnt, 55
              rcall  Wait


              lds           mpr, UDR1                   ;Read Action Code and Print it
              out    PORTB, mpr                 ;Print Action Code

              ldi    waitcnt, WTime
              rcall  Wait

              cpi           mpr, (1<<6|1<<4)                  ;Check Speed Min, Speed
Max, Speed Down, Speed Up bits
              breq   INPUT0
              cpi           mpr, ($80|1<<(EngDirL-1))
              breq   INPUT1
              cpi           mpr, (1<<(EngDirR-1)|1<<(EngDirL-1))
              breq   INPUT2
              cpi           mpr,($80|1<<(EngDirL))
              breq   INPUT3

              rcall UPLOAD                                    ;Update Leds


              ldi           mpr, (1<<INT0 | 1<<INT1)    ;Clean Queue
              out           EIFR, mpr

              ldi           waitcnt, 160
              rcall  Wait

              ret

;SPEED UP, SPEED DOWN, SPEED MIN, SPEED MAX FUNCTIONS FROM LAB 7
INPUT0:;min speed
              ldi           speed_level, 0              ;Min Speed
              ldi           speed, 0

              out           OCR0, speed_level
              out           OCR2, speed_level
              rcall  UPLOAD
              ret

INPUT1:;max speed
              ldi           speed_level, 15             ;Max Speed
              ldi           speed, $F

              out           OCR0, speed_level
              out           OCR2, speed_level
              rcall   UPLOAD
              ret
```

```
INPUT2:;-speed
              cpi           speed_level,0       ;Check if Min speed, else dec speed
and speed level
              breq    INPUT0

              dec           speed
              dec           speed_level

              out           OCR0, speed_level
              out           OCR2, speed_level
              rcall   UPLOAD
              ret

INPUT3:;+speed
              cpi           speed_level,15              ;Check if Max Speed, else inc
speed and speed level
              breq    INPUT1

              ldi           mpr, 1
              inc     speed
              inc           speed_level

              out           OCR0, speed_level
              out           OCR2, speed_level
              rcall   UPLOAD
              ret
;######################
UPLOAD:
              ldi           mpr, Halt                   ;Update Leds
              or            mpr, speed
              out           PORTB, mpr
              ret

;-----------------------------------------------------------------
; 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              ; 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

              ldi     mpr,(1<<INT0 | 1<<INT1)     ; clean the queue
              out     EIFR,mpr


              ; Move Forward again
```

```
            ldi         mpr, MovFwd                         ; Load Move
Forward command
            out         PORTB, mpr                          ; Send command to
port
            ldi         mpr, (1<<INT0 | 1<<INT1)    ;Clean Queue
            out         EIFR, mpr

            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      ; 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

            ldi     mpr,(1<<INT0 | 1<<INT1)     ; clean the queue
            out     EIFR,mpr


            ; Move Forward again
            ldi         mpr, MovFwd                         ; Load Move
Forward command
            out         PORTB, mpr                          ; Send command to
port
            ldi         mpr, (1<<INT0 | 1<<INT1)    ;Clean Queue
            out         EIFR, mpr

            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
```