ECE 375 Lab 8

Remotely Operated Vehicle

Lab Time: Friday 12-2

Nicholas Kim
Srikar Valluri

TA Signature

1 Introduction

The goal of this lab is to create a TekBot that has a variety of movement capabilities and can respond to signals sent by its paired remote control. The concept of the design is to implement a game of freeze tag between two TekBots. Each TekBot should be controlled via signals from their remote controls, and should be able to send attack signals to each other which cause the opponent's TekBot to halt. After three successful freeze tags, the opponent's TekBot should stop and not respond to any signals.

In this lab, we use the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) module on the TekBot AVR board to communicate between multiple TekBots. The end-product of this lab is to develop a system of two boards, where one is a robot itself, and the other is a remote that sends various different signals to the robot. These are known as the reciever and transmitter respectively, and interactions between them result in a variety of capabilities from moving or halting in different directions, freezing, and subroutine capabilities. The challenge code adds slightly different functionality to the remote control system. This lab also used many concepts from previous labs, such as Timer/Counters, external interrupts, and pulse width modulation.

2 Program Overview

2.1 Transmitter

The transmitter function's main goal is to transmit two bytes of information whenever a specific button is pressed. The first byte of the data is the bot address, which is specific to each pairing of a TekBoard pair, and the second byte is the particular action that should be displayed on the TekBot. In order for transmission to occur, the INIT function must have initialization of the stack pointer, DDRD and PORTD for button inputs and the transmission pin, DDRB and PORTB for LEDs, baud rate, and USART controls/settings. Since we are using polling to gather input from the buttons, we will not be using any interrupts in this program. In the main function, polling occurs to determine what information needs to be sent, and the appropriate 2 bytes are sent to the reciever via UDR1.

2.1.1 Program Flow

The transmitter will use 6 button inputs representing 6 different commands to send to the receiver. The program will repeatedly check for button inputs and send two data signals to the receiver when a button is pressed. The first signal allows the receiver to confirm the Bot ID of the transmitter. The second signal is the command.

2.2 Receiver

The reciever has the capability of performing the HitLeft and HitRight subroutines whenever the left or right whiskers are hit, respectively. It also has the ability to recieve information from the transmitter, and if the bot IDs match, it performs the subsequent action that is sent. It even has the ability to send out a freeze signal to other TekBots, so it also performs transmission. In order for the reciever to occur, the INIT function must have initialization of the stack pointer, DDRD and PORTD for button inputs, transmission, and reciever pin, DDRB and PORTB for LEDs, baud rate, USART controls/settings, and EIMSK and EICRA for the interrupt initializations for the HitRight and HitLeft subroutine calls.

2.2.1 Program Flow

The MAIN part of the program simply loops, as all the functionality is interrupt-based. The same functionality of the Bump Bot using external interrupts on two buttons is implemented.

Whenever info is recieved, the RECIEVE interrupt will trigger, and a set of checks will be performed to determine what the input is. Any time a Freeze signal (from an opponent's attack) is received, the receiver will freeze for 5 seconds. Otherwise, it performs validation on the botID, and ignores the next set of information if it doesn't match. If a valid botID is recieved, the next data to be received will be checked and implemented, whether it is sending out a Freeze signal to other TekBots or changing its own movement.

3 Internal Register Definitions and Constants

3.1 Transmitter

The transmitter file defines two multi-purpose registers for handling data. Labels are created for the engine controls and the action codes to make the code adjustable and readable. The Bot ID is set to $3A.

3.1.1 Challenge

Action code labels for Halt and Freeze Attack are replaced with Speed Up and Speed Down. The Bot ID is set to $1A.

3.2 Receiver

The receiver file defines two multi-purpose registers for handling data. The Freeze Count register counts the number of times this receiver was frozen from an external attack. The Action Check register indicates whether or not the previous received data contained the correct Bot ID so that it can check for an action code. Several registers are defined for loop counting and delay functions. Labels are created for the engine controls and the action codes to make the code adjustable and readable. The Bot ID is set to $3A.

3.2.1 Challenge

The Brightness Multiplier register indicates the factor of 17 (defined by a constant label) used to change the pulse width modulation on the engine enables for the purpose of adjusting between 15 speed levels. The Bot ID is set to $1A.

4 Initialization Routines

4.1 Transmitter

The stack pointer is initialized first. Port D pins are inputs except for pin 3 (TXD1) which is used by the USART1 transmitter. Pins 7:4 and 1:0 are button inputs for the 6 commands and require pull-up resistors. Note that Port D pin 3:2 are used by USART1 and cannot be implemented for the command buttons. Port B pins are LED outputs.

The UBRR1 registers are initialized to a value of 832 with the double data rate enabled in UCSR1A for a baud rate of 2400bps. The transmitter is enabled in UCSR1B. The frame format (8 data bits, 2 stop bits) is configured in UCSR1C.

4.2 Receiver

The stack pointer is initialized first. Port D pins 1:0 are button inputs for the whiskers' external interrupts and require pull-up resistors. Port B pins are LED outputs.

The UBRR1 registers are initialized to a value of 832 with the double data rate enabled in UCSR1A for a baud rate of 2400bps. The receiver and receive complete interrupt are enabled in UCSR1B. The transmitter is enabled in UCSR1B for the purpose of sending Freeze signals to other receivers. The frame format (8 data bits, 2 stop bits) is configured in UCSR1C.

The EIMSK and EICRA registers are configured so that INT1:0 (PIND1:0, left and right whiskers) trigger interrupts on the falling edge. Interrupt vectors for these pins are initialized for HitRight and HitLeft.

All defined registers are initialized to zero. The TekBot initialized to move forward. The interrupt flag is set to enable interrupts.

4.2.1 Challenge

The control registers for Timer/Counter1 are configured for normal mode with a prescale of 1024 to achieve a 1 second delay. The control register for the 8-bit timers (Timer/Counter0 and Timer/Counter2) are configured for Fast PWM mode, non-inverting, without prescaling. The TIMSK enables the Timer/Counter1 Overflow Interrupt for the 1 second delay and the 8-bit timers' Output Compare Match Interrupts for the engine enable pulse width modulation.

Port B LEDs 3:0 indicate the speed level, which is initialized to 15 (full speed). The OCRs are set to zero to reflect full speed on the engine enables (fully dim).

5 Main Routines

5.1 Transmit

The polling method is used to repeatedly check all PIND inputs. When a PIND input signal is received, a relative call is made to its corresponding subroutine.

5.2 Receiver

The receiver operates entirely on interrupts.

6 Subroutines

6.1 Transmitter

6.1.1 Action Code Loading

Functions `MOV_FWD`, `MOV_BCK`, `TURN_R`, `TURN_L`, `HALT`, and `FRZ_ATK` each load and transmit the Bot ID followed by their respective action code. Port B is updated to indicate the most recent transmit.

6.1.2 TRANSMIT

The transmitter waits for the USART1 Data Register to be empty (by checking UCSR1A) before loading it with a predefined action code and transmitting the signal.

6.1.3 Challenge

Functions `HALT` and `FRZ_ATK` are replaced with `SPEED_UP` and `SPEED_DOWN`.

6.2 Receiver

6.2.1 HitRight

The current action is pushed to the stack. The receiver is disabled in UCSR1B. The TekBot moves backwards and turns left for 1 second each. The queue is cleared and the receiver is re-enabled before returning to the previous action.

6.2.2 HitLeft

The current action is pushed to the stack. The receiver is disabled in UCSR1B. The TekBot moves backwards and turns right for 1 second each. The queue is cleared and the receiver is re-enabled before returning to the previous action.

6.2.3 FREEZE

The current action is pushed to the stack. The Freeze Signal is flashed for 100ms as an indication that this TekBot was successfully attacked by the oppoenent. The receiver interrupts and external interrupts are disabled in UCSR1B and EIMSK, and the TekBot is halted. At this point, the

TekBot is stopped and will not respond to any signals sent by the transmitter or the whisker inputs.

The Freeze Count is incremented after disabling interrupts and halting. If it is the third time being frozen, the function enters an infinite loop so that it remains halted and unresponsive until the TekBot is reset.

If it has not been frozen three times, the function delays for 5 seconds before re-enabling the interrupts, clearing the queue, and returning to the previous action by popping from the stack.

6.2.4 RECEIVE

This function is called via receive complete interrupt. The data register is loaded and will either cause a freeze, a Action Check register set, a freeze attack, or a TekBot movement. The receive function is ended immediately after an action is taken.

The data is first compared to the Freeze Code to determine if this receiver is being frozen by another receiver. If a Freeze Signal is received, the FREEZE function is called and RECEIVE ends.

If the Action Check register is not set, the data is compared the Bot ID. If the data matches the Bot ID, the signal is being sent by the corresponding transmitter, and the Action Check register is set so that the receiver can read the next data signal for a command.

If the Action Check register is set, the data is compared to the Freeze Attack action code. The Freeze Attack sequence disables the receiver to avoid freezing itself via reflected IR signal. The Freeze signal is then loaded and transmitted before re-enabling the receiver after a brief delay.

If the Freeze Attack is not called, the data is compared to all action codes to determine if it is a valid TekBot movement command. The MSB in a valid command is shifted out (to the left) before sending the resulting TekBot action to the LEDs.

6.2.5 TRANSMIT

The receiver waits for the USART1 Data Register to be empty (by checking UCSR1A) before loading it with a predefined action code and transmitting the signal. The receiver is only capable of sending Freeze signals during a Freeze Attack. The Freeze Signal is flashed for 100ms as an indicator.

6.2.6 Delay10ms

A triply nested loop creates a delay for any factor of 10ms as defined by the wait count register. Delays of 100ms are frequently used for signal flashes and to avoid switch debouncing.

6.2.7 Challenge

For all instances of changing the TekBot movement on the Port B LED outputs, the engine controls and speed indicators must be able to be adjusted without overriding one another. The set/clear bits in register, and/or, and and/or immediate instructions are used throughout the subroutines to avoid this error.

The HitRight and HitLeft subroutines call a different 1 second delay function which uses a Timer/Counter1 Overflow Interrupt. Because these functions are already occuring as a result of an external interrupt, the global interrupt flag is cleared and set at the start of the subroutine to enable interrupts. The Output Compare Match Interrupts are disabled for the 8-bit timers.

The FREEZE function saves and returns to the previous engine speed by pushing and popping the OCRs to and from the stack. The OCRs must be fully set during the 5 second delay so that the engines are fully disabled.

The RECEIVE function checks for each valid command during the action check before performing its corresponding action. The checks for the Speed Up and Speed Down action codes disable and re-enable the receive interrupts to avoid receiving double-commands from the transmitter as it calls the SPEED_UP and SPEED_DOWN subroutines.

6.2.8 SPEED_UP

If the speed factor is not 15, increments the speed factor and updates LEDs 3:0 by masking out 7:4. The speed factor times the speed constant is subtracted from the maximum speed (255) to get the relative LED intensity. The OCRs are updated with the new LED intensity.

6.2.9 SPEED_DOWN

If the speed factor is not zero, decrements the speed factor and updates LEDs 3:0 by masking out 7:4. The speed factor times the speed constant is subtracted from the maximum speed (255) to get the relative LED intensity. The OCRs are updated with the new LED intensity.

6.2.10 WAIT

Timer/Counter1 Overflow Interrupt is used to implement a 1 second delay for the HitRight/HitLeft sequence. The Overflow bit is reset by writing 1 to TIFR. The TCNT1 register is set to a calculated value of 49911 to produce a 1 second delay. The Wait for Overflow loop delays for 1 second before the continue register is set by the overflow interrupt.

6.2.11 TIMER1_OVF

When TCNT1 reaches MAX, 1 second has passed. This Overflow Interrupt is called and the continue register is set so that the Wait for Overflow loop in WAIT will break.

7 Difficulties

A general difficulty throughout the receiver code was receiving incorrect signals or partial signals from the transmitter, causing the receiver to display the Bot ID (which it had most recently received) on the LED outputs. This issue was resolved by implementing a hard check on the action code for each valid command before shifting and loading the action onto the LEDs.

There were many instances where brief 100ms delays and receive disables were included to avoid switch debouncing and reflected signals. This error mainly occured during RECEIVE and the speed changing functions for the challenge code. The RECEIVE function would receive multiple queued commands and cause errors within the subroutine. The speed changing functions would often queue multiple times within one press, so a brief delay as well as a disabling of the receiver to avoid receiving a second reflected signal. This error-handling was already described for avoiding self-freezes.

8 Conclusion

In this lab, we used the USART module on the TekBot AVR board to communicate between multiple TekBots. The transmitter and reciever were able to communicate with various different signals, which included verification of the correct remote, multiple movement actions, and freeze actions. The challenge code included different features to the remote control system.

9 Source Code

9.1 Transmitter

```
;***********************************************************
;*
;* This is the TRANSMIT file for Lab 8 of ECE 375
;*
;*  Author: Nicholas Kim and Srikar Valluri
;*    Date: 2/25/2022
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def mpr2 = r17

.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:
; 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 Halt_Code = ($80|1<<(EngEnR-1)|1<<(EngEnL-1)) ;0b11001000 Halt Action Code
.equ Frz_Atk_Code = 0b11111000 ;0b11111000 Freeze Attack Action Code

.equ BotAddress = $3A

;***********************************************************
;* 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, low(RAMEND)
out SPL, mpr
ldi mpr, high(RAMEND)
out SPH, mpr

;I/O Ports
; Port D for button inputs with pull-up resistors
ldi mpr, 0b00001000 ; TXD1 remains an output
out DDRD, mpr
ldi mpr, 0b11110011 ; Pins 3 and 2 are not button inputs
out PORTD, mpr

; Port B for LEDs, initially off
ldi mpr, $FF
```

```
out DDRB, mpr
ldi mpr, $00
out PORTB, mpr

;USART1
;Set baudrate at 2400bps
ldi mpr, high(832)
sts UBRR1H,mpr
ldi mpr, low(832)
sts UBRR1L,mpr

; Set Double Data rate
ldi mpr, (1<<U2X1)
sts UCSR1A,mpr

;Enable transmitter
ldi mpr, (1<<TXEN1)
sts UCSR1B,mpr

;Set frame format: 8 data bits, 2 stop bits
ldi mpr, (1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C,mpr

;External Interrupts
;Set the External Interrupt Mask
;ldi mpr, 0b11110011
;out EIMSK, mpr

;Set the Interrupt Sense Control to falling edge detection
;ldi mpr, 0b00001010
;sts EICRA, mpr
;ldi mpr, 0b10101010
;out EICRB, mpr

;Other
;sei ; Enable interrupts

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
; Load PIND into mpr
in mpr, PIND
andi mpr, 0b11110011

; Check PIND0
```

```
cpi mpr, 0b11110010
brne CHECK1 ; if this pin not pressed, check next
rcall MOV_FWD
rjmp MAIN
CHECK1:
; Check PIND1
cpi mpr, 0b11110001
brne CHECK4 ; if this pin not pressed, check next
rcall MOV_BCK
rjmp MAIN
CHECK4:
; Check PIND4
cpi mpr, 0b11100011
brne CHECK5 ; if this pin not pressed, check next
rcall TURN_R
rjmp MAIN
CHECK5:
; Check PIND5
cpi mpr, 0b11010011
brne CHECK6 ; if this pin not pressed, check next
rcall TURN_L
rjmp MAIN
CHECK6:
; Check PIND6
cpi mpr, 0b10110011
brne CHECK7 ; if this pin not pressed, check next
rcall HALT
rjmp MAIN
CHECK7:
; Check PIND7
cpi mpr, 0b01110011
brne MAIN ; if this pin not pressed, check next
rcall FRZ_ATK
rjmp MAIN

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

;***********************************************************
;* Function: MOV_FWD
;* Description: Transmits the BotAddress and the Move
;* Forward action code to the receiver
;***********************************************************
MOV_FWD:
push mpr
```

```
ldi mpr2, 0b00000001
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, MovFwd ; Load and send Move Foward action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: MOV_BCK
;* Description: Transmits the BotAddress and the Move
;* Backward action code to the receiver
;***********************************************************
MOV_BCK:
push mpr

ldi mpr2, 0b00000010
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, MovBck ; Load and send Move Backward action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: TURN_R
;* Description: Transmits the BotAddress and the Turn
;* Right action code to the receiver
;***********************************************************
TURN_R:
push mpr

ldi mpr2, 0b00010000
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, TurnR ; Load and send Turn Rightdf action code
rcall TRANSMIT
```

```
pop mpr
ret ; Return

;**********************************************************
;* Function: TURN_L
;* Description: Transmits the BotAddress and the Turn
;* Left action code to the receiver
;**********************************************************
TURN_L:
push mpr

ldi mpr2, 0b00100000
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, TurnL ; Load and send Turn Left action code
rcall TRANSMIT

pop mpr
ret ; Return

;**********************************************************
;* Function: HALT
;* Description: Transmits the BotAddress and the Halt
;* action code to the receiver
;**********************************************************
HALT:
push mpr

ldi mpr2, 0b01000000
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, Halt_Code ; Load and send Halt action code
rcall TRANSMIT

pop mpr
ret ; Return

;**********************************************************
;* Function: FRZ_ATK
;* Description: Transmits the BotAddress and the Freeze
;* Attack action code to the receiver
```

```
;***********************************************************
FRZ_ATK:
push mpr

ldi mpr2, 0b10000000
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, Frz_Atk_Code ; Load and send Freeze Attack action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: TRANSMIT
;* Description: Waits for the USART1 Data Register to be
;* empty before loading it with mpr and transmitting the signal
;***********************************************************
TRANSMIT:
lds mpr2, UCSR1A
andi mpr2, 0b00100000
cpi mpr2, 0b00100000 ; Proceed if UDR1 is empty
breq SKIP_T
rjmp TRANSMIT ; and ready to transmit data
SKIP_T:
sts UDR1, mpr ; Send action code through UDR1

ret ; Return

;***********************************************************
;* Stored Program Data
;***********************************************************

;***********************************************************
;* Additional Program Includes
;***********************************************************
```

9.2 Receiver

```
;***********************************************************
;*
;* This is the RECEIVE file for Lab 8 of ECE 375
;*
;*  Author: Nicholas Kim and Srikar Valluri
;*    Date: 3/1/2022
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def mpr2 = r20 ; Multi-Purpose Register
.def continue = r17 ; Used for Delay1 function
.def frz_cnt = r18 ; Freeze Count
.def loop_cnt = r19 ; Loop Count
.def action_check = r21 ; Check for action code if set

.def waitcnt = r22
.def olcnt = r23
.def ilcnt = r24

.def mpr3 = r25

.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 = $3A ;(Enter your robot's address here (8 bits))

;////////////////////////////////////////////////////////////
;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
.equ Freeze_Code = 0b01010101 ;0b01010101 Freeze 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:
;- Right whisker
.org $0002
rjmp HitRight

;- Left whisker
.org $0004
rjmp HitLeft

;- USART receive
.org $003C
rjmp RECEIVE

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Stack Pointer (VERY IMPORTANT!!!!)
ldi mpr, low(RAMEND)
out SPL, mpr
ldi mpr, high(RAMEND)
out SPH, mpr

;I/O Ports
; Port B for LED outputs (initially off)
ldi mpr, $FF
out DDRB, mpr
ldi mpr, $00
out PORTB, mpr

; Port D for button inputs with pull-up resistors
ldi mpr, 0b11111100
out DDRD, mpr
ldi mpr, 0b00000011
out PORTD, mpr

;USART1
;Set baudrate at 2400bps
```

```
ldi mpr, high(832)
sts UBRR1H,mpr
ldi mpr, low(832)
sts UBRR1L,mpr

; Set Double Data rate
ldi mpr, (1<<U2X1)
sts UCSR1A,mpr

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

;Set frame format: 8 data bits, 2 stop bits
ldi mpr, (1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C,mpr

;External Interrupts
;Set the External Interrupt Mask
ldi mpr, 0b00000011
out EIMSK, mpr

;Set the Interrupt Sense Control to falling edge detection
ldi mpr, 0b00001010
sts EICRA, mpr

; Initialize Registers
ldi continue,$00
ldi frz_cnt,$00
ldi loop_cnt,$00
ldi action_check,$00

ldi waitcnt,$0A ; for 100ms delay
ldi olcnt,$00
ldi ilcnt,$00

ldi mpr3,$00

; Initialize TekBot Forward Movement
ldi mpr, MovFwd ; Load Move Forward Command
out PORTB, mpr ; Send command to motors

;Other
sei ; Enable interrupts
```

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

;***********************************************************
;* Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------
; Func: HitRight
; Desc: This function is called whenever the INT0 is activated.
; Since the right whisker is activated, it moves backwards for a
; second, turns left, and moves forward again.
;-----------------------------------------------------------
HitRight:

push mpr
in mpr, PORTB ; Save current action
push mpr

; Disable receiver
ldi mpr, (0<<RXCIE1 | 0<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

; Move Backwards for a second
ldi mpr, MovBck ; Load Move Backward command
out PORTB, mpr ; Send command to port
ldi waitcnt, 100 ; Load 1000(ms) into waitcnt
rcall Delay10ms ; Delay one second

; Turn left for a second
ldi mpr, TurnL ; Load Turn Left Command
out PORTB, mpr ; Send command to port
ldi waitcnt, 100 ; Load 1000(ms) into waitcnt
rcall Delay10ms ; Delay one second

; Move Forward again
ldi mpr, MovFwd ; Load Move Forward command
out PORTB, mpr ; Send command to port

; Clear queue
ldi mpr, $FF ; Clear EIFR by setting all bits
out EIFR, mpr ; to avoid queued interrupts

; Re-enable receiver and receive interrupts
```

```
ldi mpr, (1<<RXCIE1 | 1<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

pop mpr
out PORTB, mpr ; Resume previous action
pop mpr

reti

;----------------------------------------------------------
; Func: HitLeft
; Desc: This function is called whenever the INT1 is activated.
; Since the left whisker is activated, it moves backwards for a
; second, turns right, and moves forward again.
;----------------------------------------------------------
HitLeft:

push mpr
in mpr, PORTB ; Save current action
push mpr

; Disable receiver
ldi mpr, (0<<RXCIE1 | 0<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

; Move Backwards for a second
ldi mpr, MovBck ; Load Move Backward command
out PORTB, mpr ; Send command to port
ldi waitcnt, 100 ; Load 1000(ms) into waitcnt
rcall Delay10ms ; Delay one second

; Turn right for a second
ldi mpr, TurnR ; Load Turn Right Command
out PORTB, mpr ; Send command to port
ldi waitcnt, 100 ; Load 1000(ms) into waitcnt
rcall Delay10ms ; Delay one second

; Move Forward again
ldi mpr, MovFwd ; Load Move Forward command
out PORTB, mpr ; Send command to port

; Clear queue
ldi mpr, $FF ; Clear EIFR by setting all bits
out EIFR, mpr ; to avoid queued interrupts

; Re-enable receiver and receive interrupts
```

```
ldi mpr, (1<<RXCIE1 | 1<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

pop mpr
out PORTB, mpr ; Resume previous action
pop mpr

reti

;-----------------------------------------------------------
; Func: FREEZE
; Desc: "Freeze" for 5 seconds. Halts, disables whisker response,
; and disables transmitter command response
;-----------------------------------------------------------
FREEZE:
push mpr
in mpr, PORTB ; Save current action
push mpr

; Flash Freeze signal for 100ms second as an indicator
in mpr2, PORTB ; Save previous action

ldi mpr, $55
out PORTB, mpr ; Flash Freeze signal
ldi waitcnt, 10
rcall Delay10ms

out PORTB, mpr2 ; Return to previous action

; Immediately disable interrupts
;Disable receiver, receive interrupts,
; and tramsmitter for Freeze Attack
ldi mpr, $00
sts UCSR1B,mpr

;Clear the External Interrupt Mask
ldi mpr, $00
out EIMSK, mpr

; Halt
ldi mpr, Halt ; Load Halt Command
out PORTB, mpr ; Send command to port

inc frz_cnt ; Increment Freeze Count

; If third time being frozen, stay frozen
```

```
cpi frz_cnt, $03
brne SKIP_3F
FROZEN:
rjmp FROZEN ; Infinite Loop

SKIP_3F:

; Wait for 5 seconds
ldi loop_cnt, $05
WAIT_5:
ldi waitcnt, 100
rcall Delay10ms ; Delay one second
dec loop_cnt ; Decrement count
brne WAIT_5 ; Wait for 5 loops

; Re-enable interrupts
;Enable receiver, receive interrupts,
; and tramsmitter for Freeze Attack
ldi mpr, (1<<RXCIE1 | 1<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

;Set the External Interrupt Mask
ldi mpr, 0b00000011
out EIMSK, mpr

; Clear queue
ldi mpr, $FF ; Clear EIFR by setting all bits
out EIFR, mpr ; to avoid queued interrupts

pop mpr
out PORTB, mpr ; Resume previous action
pop mpr

ret ; Return from FREEZE

;-----------------------------------------------------------------
; Sub: Delay10ms
; 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
;-----------------------------------------------------------------
Delay10ms:
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

;------------------------------------------------------------
; Func: RECEIVE
; Desc: Read in data from Receive Data Buffer
;------------------------------------------------------------
RECEIVE:
push mpr
push mpr2

ldi waitcnt, 10
rcall Delay10ms ; Brief delay to avoid switch debouncing

lds mpr, UDR1

; Check for Freeze
cpi mpr, Freeze_Code
brne SKIP_F ; If not being frozen, skip
rcall FREEZE
rjmp END_R

SKIP_F:
; If previous receive was correct address, check for action
cpi action_check,$FF
breq ACT_CHECK
; Otherwise, check for address

ADR_CHECK:
cpi mpr, BotAddress
brne END_R ; Skip all if wrong address

; If address is correct
```

```
ldi action_check,$FF ; Load action check
rjmp END_R

ACT_CHECK:

ldi action_check,$00 ; Clear action check

; Check for Freeze Attack action code
cpi mpr, 0b11111000
brne SKIP_ATK ; Skip attack sequence

; Freeze Attack Sequence
; Disable receiver to avoid self-freeze
ldi mpr, (0<<RXCIE1 | 0<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

; Send out Freeze signal
ldi mpr, 0b01010101
rcall TRANSMIT

ldi waitcnt, 10
rcall Delay10ms ; Brief delay to avoid self-freeze

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

rjmp END_R

SKIP_ATK:
; Check each command code to determine if valid or junk
CHECK_FWD:
cpi mpr, 0b10110000 ; Move Forward action code
brne CHECK_BCK
rjmp VALID
CHECK_BCK:
cpi mpr, 0b10000000 ; Move Back action code
brne CHECK_R
rjmp VALID
CHECK_R:
cpi mpr, 0b10100000 ;Turn Right action code
brne CHECK_L
rjmp VALID
CHECK_L:
cpi mpr, 0b10010000 ; Turn Left action code
brne CHECK_HALT
```

```
rjmp VALID
CHECK_HALT:
cpi mpr, 0b11001000
brne END_R

VALID:
; If not Freeze Attack, load action code onto LEDs
lsl mpr ; TekBot movement is Action Code
out PORTB, mpr ; with MSB (1) shifted out

rjmp END_R

END_R:

pop mpr2
pop mpr
reti ; Return from interrupt

;***********************************************************
;* Function: TRANSMIT
;* Description: Waits for the USART1 Data Register to be
;* empty before loading it with mpr and transmitting the signal
;***********************************************************
TRANSMIT:
lds mpr2, UCSR1A
andi mpr2, 0b00100000
cpi mpr2, 0b00100000 ; Proceed if UDR1 is empty
breq EMPTY
rjmp TRANSMIT ; and ready to transmit data
EMPTY:
sts UDR1, mpr ; Send action code through UDR1

; Flash Freeze signal for 100ms as an indicator
in mpr2, PORTB ; Save previous action

out PORTB, mpr ; Flash Freeze signal

ldi waitcnt, 10
rcall Delay10ms ; Brief delay to flash signal

out PORTB, mpr2 ; Return to previous action

ret ; Return

;***********************************************************
;* Stored Program Data
```

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

;***********************************************************
;* Additional Program Includes
;***********************************************************
```

10 Challenge Code

10.1 Transmitter

```
;***********************************************************
;*
;* This is the TRANSMIT file for Lab 8 Challenge of ECE 375
;*
;*  Author: Nicholas Kim and Srikar Valluri
;*    Date: 2/25/2022
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def mpr2 = r17

.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:
; 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<<(EngEnR-1)|1<<(EngEnL-1)) ;0b11001000 Speed Up Action Code
.equ SpeedDown = 0b11111000 ;0b11111000 Speed Down Action Code

.equ BotAddress = $1A

;***********************************************************
;* 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, low(RAMEND)
out SPL, mpr
ldi mpr, high(RAMEND)
out SPH, mpr

;I/O Ports
; Port D for button inputs with pull-up resistors
ldi mpr, 0b00001000 ; TXD1 remains an output
out DDRD, mpr
ldi mpr, 0b11110011 ; Pins 3 and 2 are not button inputs
out PORTD, mpr

; Port B for LEDs, initially off
ldi mpr, $FF
out DDRB, mpr
ldi mpr, $00
out PORTB, mpr

;USART1
;Set baudrate at 2400bps
ldi mpr, high(832)
sts UBRR1H,mpr
ldi mpr, low(832)
sts UBRR1L,mpr

; Set Double Data rate
ldi mpr, (1<<U2X1)
sts UCSR1A,mpr

;Enable transmitter
ldi mpr, (1<<TXEN1)
```

```
sts UCSR1B,mpr

;Set frame format: 8 data bits, 2 stop bits
ldi mpr, (1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C,mpr

;External Interrupts
;Set the External Interrupt Mask
;ldi mpr, 0b11110011
;out EIMSK, mpr

;Set the Interrupt Sense Control to falling edge detection
;ldi mpr, 0b00001010
;sts EICRA, mpr
;ldi mpr, 0b10101010
;out EICRB, mpr

;Other
;sei ; Enable interrupts

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
; Load PIND into mpr
in mpr, PIND
andi mpr, 0b11110011

; Check PIND0
cpi mpr, 0b11110010
brne CHECK1 ; if this pin not pressed, check next
rcall MOV_FWD
rjmp MAIN
CHECK1:
; Check PIND1
cpi mpr, 0b11110001
brne CHECK4 ; if this pin not pressed, check next
rcall MOV_BCK
rjmp MAIN
CHECK4:
; Check PIND4
cpi mpr, 0b11100011
brne CHECK5 ; if this pin not pressed, check next
rcall TURN_R
rjmp MAIN
CHECK5:
```

```
; Check PIND5
cpi mpr, 0b11010011
brne CHECK6 ; if this pin not pressed, check next
rcall TURN_L
rjmp MAIN
CHECK6:
; Check PIND6
cpi mpr, 0b10110011
brne CHECK7 ; if this pin not pressed, check next
rcall SPEED_UP
rjmp MAIN
CHECK7:
; Check PIND7
cpi mpr, 0b01110011
brne MAIN ; if this pin not pressed, check next
rcall SPEED_DOWN
rjmp MAIN

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

;***********************************************************
;* Function: MOV_FWD
;* Description: Transmits the BotAddress and the Move
;* Forward action code to the receiver
;***********************************************************
MOV_FWD:
push mpr

ldi mpr2, 0b00000001
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, MovFwd ; Load and send Move Foward action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: MOV_BCK
;* Description: Transmits the BotAddress and the Move
;* Backward action code to the receiver
;***********************************************************
```

```
MOV_BCK:
push mpr

ldi mpr2, 0b00000010
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, MovBck ; Load and send Move Backward action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: TURN_R
;* Description: Transmits the BotAddress and the Turn
;* Right action code to the receiver
;***********************************************************
TURN_R:
push mpr

ldi mpr2, 0b00010000
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, TurnR ; Load and send Turn Rightdf action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: TURN_L
;* Description: Transmits the BotAddress and the Turn
;* Left action code to the receiver
;***********************************************************
TURN_L:
push mpr

ldi mpr2, 0b00100000
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
```

```
ldi mpr, TurnL ; Load and send Turn Left action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: SPEED_UP
;* Description: Transmits the BotAddress and the Speed
;* Up action code to the receiver
;***********************************************************
SPEED_UP:
push mpr

ldi mpr2, 0b01000000
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, SpeedUp ; Load and send Speed Up action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: SPEED_DOWN
;* Description: Transmits the BotAddress and the Speed
;* Down action code to the receiver
;***********************************************************
SPEED_DOWN:
push mpr

ldi mpr2, 0b10000000
out PORTB, mpr2

ldi mpr, BotAddress ; Load and send BotAddress
rcall TRANSMIT
ldi mpr, SpeedDown ; Load and send Speed Down action code
rcall TRANSMIT

pop mpr
ret ; Return

;***********************************************************
;* Function: TRANSMIT
;* Description: Waits for the USART1 Data Register to be
;* empty before loading it with mpr and transmitting the signal
;***********************************************************
TRANSMIT:
lds mpr2, UCSR1A
andi mpr2, 0b00100000
cpi mpr2, 0b00100000 ; Proceed if UDR1 is empty
breq SKIP_T
rjmp TRANSMIT ; and ready to transmit data
SKIP_T:
sts UDR1, mpr ; Send action code through UDR1

ret ; Return

;***********************************************************
;* Stored Program Data
;***********************************************************

;***********************************************************
;* Additional Program Includes
;***********************************************************
```

10.2 Receiver

```
;***********************************************************
;*
;* This is the RECEIVE file for Lab 8 Challenge of ECE 375
;*
;*  Author: Nicholas Kim and Srikar Valluri
;*    Date: 3/1/2022
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def mpr2 = r20 ; Multi-Purpose Register
.def continue = r17 ; Used for Delay1 function
.def frz_cnt = r18 ; Freeze Count
.def loop_cnt = r19 ; Loop Count
.def action_check = r21 ; Check for action code if set

.def waitcnt = r22
```

```
.def olcnt = r23
.def ilcnt = r24

.def multi = r25 ; Brightness multiplier (applied to const)

.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 = $1A ;(Enter your robot's address here (8 bits))

.equ const = 17 ; brightness width

;///////////////////////////////////////////////////////////////
;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 Freeze_Code = $55 ;0b01010101 Freeze 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:
;- Right whisker
.org $0002
rjmp HitRight

;- Left whisker
.org $0004
rjmp HitLeft

;- Timer/Counter1 (for delays)
```

```
.org $001C
rjmp TIMER1_OVF

;- USART receive
.org $003C
rjmp RECEIVE

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Stack Pointer (VERY IMPORTANT!!!!)
ldi mpr, low(RAMEND)
out SPL, mpr
ldi mpr, high(RAMEND)
out SPH, mpr

;I/O Ports
; Port B for LED outputs (initially off)
ldi mpr, $FF
out DDRB, mpr
ldi mpr, $00
out PORTB, mpr

; Port D for button inputs with pull-up resistors
ldi mpr, 0b11111100
out DDRD, mpr
ldi mpr, 0b00000011
out PORTD, mpr

;USART1
;Set baudrate at 2400bps
ldi mpr, high(832)
sts UBRR1H,mpr
ldi mpr, low(832)
sts UBRR1L,mpr

; Set Double Data rate
ldi mpr, (1<<U2X1)
sts UCSR1A,mpr

;Enable receiver and enable receive interrupts,
; and tramsmitter for Freeze Attack
ldi mpr, (1<<RXCIE1 | 1<<RXEN1 | 1<<TXEN1)
```

```
sts UCSR1B,mpr

;Set frame format: 8 data bits, 2 stop bits
ldi mpr, (1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C,mpr

;External Interrupts
;Set the External Interrupt Mask
ldi mpr, 0b00000011
out EIMSK, mpr

;Set the Interrupt Sense Control to falling edge detection
ldi mpr, 0b00001010
sts EICRA, mpr

;Timer/Counter1 Initialization
; Configure 16-bit Timer/Counters
ldi mpr, 0b00000000
out TCCR1A, mpr
ldi mpr, 0b00000101 ; prescaling of 1024 to get 1s count
out TCCR1B, mpr
ldi mpr, 0b00000000
sts TCCR1C, mpr

; Enable Overflow Interrupt for Timer/Counter1
; CHALLENGE: enable Output Compare Match for 0/2
ldi mpr, 0b10000110 ; Set TOEI1, OCIE2, and OCIE0
out TIMSK, mpr

; Configure 8-bit Timer/Counters
ldi mpr, 0b01101001 ; Fast PWM Mode, non-inverting
out TCCR0, mpr
out TCCR2, mpr
; no prescaling

; Initialize TekBot Forward Movement at full speed
ldi mpr, 0b01101111 ; Load Move Forward Command and multi = 15
out PORTB, mpr ; Send command to motors

; Set initial speed, display on Port B pins 3:0
; Set OCRs to zero (low LED intensity = full speed)
ldi mpr, 0b00000000
out OCR0, mpr
out OCR2, mpr

; Initialize Registers
```

```
ldi continue,$00
ldi frz_cnt,$00
ldi loop_cnt,$00
ldi action_check,$00

ldi waitcnt,$0A ; for 100ms delay
ldi olcnt,$00
ldi ilcnt,$00

ldi multi,15 ; Full speed

;Other
sei ; Enable interrupts

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

;***********************************************************
;* Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------
; Func: HitRight
; Desc: This function is called whenever the INT0 is activated.
; Since the right whisker is activated, it moves backwards for a
; second, turns left, and moves forward again.
;-----------------------------------------------------------
HitRight:

push mpr
in mpr, PORTB ; Save current action
push mpr

; Clear interrupt flag in order to generate another interrupt
cli ; Clear interrupt flag

; Disable receiver
ldi mpr, (0<<RXCIE1 | 0<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

; Clear the External Interrupt Mask
ldi mpr, $00
out EIMSK, mpr
```

```
; Disable Output Compare Match for 0/2
ldi mpr, 0b00000100 ; Set TOEI1
out TIMSK, mpr

; Enable interrupts to allow Overflow Interrupt
sei ; Set interrupt flag

ldi continue, $00

; Move Backwards for a second
in mpr, PORTB ; Save speed indicator
cbr mpr, 0b01100000 ; Only clear pins 5 and 6
out PORTB, mpr ; Send command to port
rcall WAIT

; Turn left for a second
; Toggle on pin 5 from previous command to turn left
in mpr, PORTB ; Save speed indicator
sbr mpr, 0b00100000 ; Set pin 5 without altering other pins
out PORTB, mpr ; Send command to port
rcall WAIT

; Clear queue
ldi mpr, $FF ; Clear EIFR by setting all bits
out EIFR, mpr ; to avoid queued interrupts

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

; Set the External Interrupt Mask
ldi mpr, 0b00000011
out EIMSK, mpr

; Enable Overflow Interrupt for Timer/Counter1
; CHALLENGE: enable Output Compare Match for 0/2
ldi mpr, 0b10000110 ; Set TOEI1, OCIE2, and OCIE0
out TIMSK, mpr

pop mpr
out PORTB, mpr ; Resume previous action
pop mpr

reti

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

```
; Func: HitLeft
; Desc: This function is called whenever the INT1 is activated.
; Since the left whisker is activated, it moves backwards for a
; second, turns right, and moves forward again.
;-----------------------------------------------------------
HitLeft:

push mpr
in mpr, PORTB ; Save current action
push mpr

; Clear interrupt flag in order to generate another interrupt
cli ; Clear interrupt flag

; Disable receiver
ldi mpr, (0<<RXCIE1 | 0<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

; Clear the External Interrupt Mask
ldi mpr, $00
out EIMSK, mpr

; Disable Output Compare Match for 0/2
ldi mpr, 0b00000100 ; Set TOEI1
out TIMSK, mpr

; Enable interrupts to allow Overflow Interrupt
sei ; Set interrupt flag

ldi continue, $00

; Move Backwards for a second
in mpr, PORTB ; Save speed indicator
cbr mpr, 0b01100000 ; Only clear pins 5 and 6
out PORTB, mpr ; Send command to port
rcall WAIT

; Reset TOV1 by writing 1 to it
ldi mpr, 0b00000100
out TIFR, mpr

; Turn left for a second
; Toggle on pin 6 from previous command to turn right
in mpr, PORTB ; Save speed indicator
sbr mpr, 0b01000000 ; Set pin 6 without altering other pins
out PORTB, mpr ; Send command to port
```

```
rcall WAIT

; Clear queue
ldi mpr, $FF ; Clear EIFR by setting all bits
out EIFR, mpr ; to avoid queued interrupts

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

; Set the External Interrupt Mask
ldi mpr, 0b00000011
out EIMSK, mpr

; Enable Overflow Interrupt for Timer/Counter1
; CHALLENGE: enable Output Compare Match for 0/2
ldi mpr, 0b10000110 ; Set TOEI1, OCIE2, and OCIE0
out TIMSK, mpr

pop mpr
out PORTB, mpr ; Resume previous action
pop mpr

reti

;------------------------------------------------------------
; Func: FREEZE
; Desc: "Freeze" for 5 seconds. Halts, disables whisker response,
; and disables transmitter command response
;------------------------------------------------------------
FREEZE:
push mpr
in mpr, PORTB ; Save current action
push mpr
in mpr, OCR0
push mpr
in mpr, OCR2
push mpr

; Flash Freeze signal for 100ms as an indicator
in mpr2, PORTB ; Save previous action

ldi mpr, $55
out PORTB, mpr ; Flash Freeze signal
ldi waitcnt, 10
rcall Delay10ms
```

```
out PORTB, mpr2 ; Return to previous action

; Immediately disable interrupts
;Disable receiver, receive interrupts,
; and tramsmitter for Freeze Attack
ldi mpr, $00
sts UCSR1B,mpr

;Clear the External Interrupt Mask
ldi mpr, $00
out EIMSK, mpr

; Halt
in mpr, PORTB ; Save speed indicator
andi mpr, 0b00001111 ; Mask out engine control
ori mpr, 0b10010000 ; Disable engines
out PORTB, mpr

ldi mpr, 0b11111111
out OCR0, mpr
out OCR2, mpr

inc frz_cnt ; Increment Freeze Count

; If third time being frozen, stay frozen
cpi frz_cnt, $03
brne SKIP_3F
FROZEN:
rjmp FROZEN ; Infinite Loop

SKIP_3F:

; Wait for 5 seconds
ldi loop_cnt, $05
WAIT_5:
ldi waitcnt, 100
rcall Delay10ms ; Delay one second
dec loop_cnt ; Decrement count
brne WAIT_5 ; Wait for 5 loops

; Re-enable interrupts
;Enable receiver, receive interrupts,
; and tramsmitter for Freeze Attack
ldi mpr, (1<<RXCIE1 | 1<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr
```

```
;Set the External Interrupt Mask
ldi mpr, 0b00000011
out EIMSK, mpr

; Clear queue
ldi mpr, $FF ; Clear EIFR by setting all bits
out EIFR, mpr ; to avoid queued interrupts

pop mpr
out OCR2, mpr
pop mpr
out OCR0, mpr
pop mpr
out PORTB, mpr ; Resume previous action
pop mpr

ret ; Return from FREEZE

;------------------------------------------------------------
; Func: SPEED_DOWN
; Desc: Increases the OCR0 and OCR2 to decrease the speed
; of the TekBot motors
;------------------------------------------------------------
SPEED_DOWN: ; Begin a function with a label

; If needed, save variables by pushing to the stack
push mpr
push mpr2

ldi waitcnt, 10
rcall Delay10ms ; Brief delay to avoid switch debouncing

cpi multi, 0b00000000 ; if multi is zero
breq SKIP_DEC ; cannot decrease speed, so skip

dec multi ; decrease multi
mov mpr, multi

in mpr2, PORTB ; Save engine control
andi mpr2, 0b11110000 ; Mask out previous speed indicator

or mpr, mpr2 ; OR with engine direction bits
out PORTB, mpr ; output new multi onto PORTB

ldi mpr, const
```

```
mul multi, mpr ; multiply multi with 17
ldi mpr, 255
sub mpr, r0 ; subtract from 255 to get LED intensity

out OCR0, mpr ; output new LED intensity
out OCR2, mpr

SKIP_DEC:

; Clear queue
ldi mpr, $FF ; Clear EIFR by setting all bits
out EIFR, mpr ; to avoid queued interrupts

; Restore any saved variables by popping from stack
pop mpr2
pop mpr

ret ; Return from Speed Down

;------------------------------------------------------------
; Func: SPEED_UP
; Desc: Decreases the OCR0 and OCR2 to increase the speed
; of the TekBot motors
;------------------------------------------------------------
SPEED_UP: ; Begin a function with a label

; If needed, save variables by pushing to the stack
push mpr
push mpr2

ldi waitcnt, 10
rcall Delay10ms ; Brief delay to avoid switch debouncing

cpi multi, 0b00001111 ; if multi is 15
breq SKIP_INC ; cannot increase speed, so skip

inc multi ; increase multi
mov mpr, multi

in mpr2, PORTB ; Save engine control
andi mpr2, 0b11110000 ; Mask out previous speed indicator

or mpr, mpr2 ; OR with engine direction bits
out PORTB, mpr ; output new multi onto PORTB

ldi mpr, const
```

```
mul multi, mpr ; multiply multi with 17
ldi mpr, 255
sub mpr, r0 ; subtract from 255 to get LED intensity

out OCR0, mpr ; output new LED intensity
out OCR2, mpr

SKIP_INC:

; Clear queue
ldi mpr, $FF ; Clear EIFR by setting all bits
out EIFR, mpr ; to avoid queued interrupts

; Restore any saved variables by popping from stack
pop mpr2
pop mpr

ret ; Return from Speed Up

;------------------------------------------------------------
; Func: Wait
; Desc: One second delay using Timer/Counter1 Interrupt
;------------------------------------------------------------
WAIT:
push mpr

; Reset TOV1 by writing 1 to it
ldi mpr, 0b00000100
out TIFR, mpr

; Initialize TCNT1 for 1s
ldi mpr, high(49911)
out TCNT1H, mpr
ldi mpr, low(49911)
out TCNT1L, mpr

; Wait for TOV1 to be set before continuing
;WAIT_TOV1:
; Polling method
;in mpr, TIFR ; Check if TOV1 is set
;andi mpr, 0b00000100
;breq WAIT_TOV1 ; Wait if TOV1 is not set

; Overflow Interrupt method
; Wait for Timer/Counter1 Overflow Interrupt
WAIT_OVF:
```

```
cpi continue, $FF ; Will be set during interrupt
brne WAIT_OVF ; Wait if continue is not set

; Reset continue by clearing
ldi continue, $00

; Restore any saved variables by popping from stack
pop mpr

ret ; Return from WAIT

;-----------------------------------------------------------------
; Sub: Delay10ms
; 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
;-----------------------------------------------------------------
Delay10ms:
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
;------------------------------------------------------------
; Func: TIMER1_OVF
; Desc: Sets the continue register at timer overflow
;------------------------------------------------------------
TIMER1_OVF:
push mpr

; Set continue register so subroutine can proceed
ldi continue, $FF
```

```
; Debug with LED
;ldi mpr, $02
;out PORTB, mpr

; Reset TOV1 by writing 1 to it
ldi mpr, 0b00000100
out TIFR, mpr

pop mpr

reti ; Return from Overflow Interrupt

;-----------------------------------------------------------
; Func: RECEIVE
; Desc: Read in data from Receive Data Buffer
;-----------------------------------------------------------
RECEIVE:
push mpr
push mpr2

ldi waitcnt, 10
rcall Delay10ms ; Brief delay to avoid switch debouncing

lds mpr, UDR1

; Check for Freeze
cpi mpr, Freeze_Code
brne SKIP_F
rcall FREEZE
rjmp END_R

SKIP_F:
; If previous receive was correct address, check for action
cpi action_check,$FF
breq ACT_CHECK
; Otherwise, check for address

ADR_CHECK:
cpi mpr, BotAddress
brne END_R ; Skip all if wrong address

; If address is correct
ldi action_check,$FF ; Load action check
rjmp END_R
```

```
ACT_CHECK:

ldi action_check,$00 ; Clear action check

CHECK_FWD:
cpi mpr, 0b10110000 ; Move Forward action code
brne CHECK_BCK

; If Move Forward command received
in mpr, PORTB ; Save speed indicator
ori mpr, 0b01100000 ; Only set pins 5 and 6
out PORTB, mpr ; Send command to port

rjmp END_R
CHECK_BCK:
cpi mpr, 0b10000000 ; Move Back action code
brne CHECK_R

; If Move Back command received
in mpr, PORTB ; Save speed indicator
andi mpr, 0b10011111 ; Only clear pins 5 and 6
out PORTB, mpr ; Send command to port

rjmp END_R
CHECK_R:
cpi mpr, 0b10100000 ;Turn Right action code
brne CHECK_L

; If Turn Right command received
in mpr, PORTB ; Save speed indicator
andi mpr, 0b10011111 ; Only clear pins 5 and 6
ori mpr, 0b01000000 ; Only set pin 6
out PORTB, mpr ; Send command to port

rjmp END_R
CHECK_L:
cpi mpr, 0b10010000 ; Turn Left action code
brne CHECK_SPD_UP

; If Turn Left command received
in mpr, PORTB ; Save speed indicator
andi mpr, 0b10011111 ; Only clear pins 5 and 6
ori mpr, 0b00100000 ; Only set pin 5
out PORTB, mpr ; Send command to port

rjmp END_R
```

```
CHECK_SPD_UP:
cpi mpr, 0b11001000 ; Speed Up action code
brne CHECK_SPD_DOWN

; If Speed Up command received
; Disable receiver to avoid double-command
ldi mpr, (0<<RXCIE1 | 0<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

rcall SPEED_UP

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

rjmp END_R
CHECK_SPD_DOWN:
cpi mpr, 0b11111000 ; Speed Down action code
brne END_R

; If Speed Down command received
; Disable receiver to avoid double-command
ldi mpr, (0<<RXCIE1 | 0<<RXEN1 | 1<<TXEN1)
sts UCSR1B,mpr

rcall SPEED_DOWN

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

rjmp END_R

END_R:

pop mpr2
pop mpr
reti ; Return from interrupt

;**********************************************************
;* Function: TRANSMIT
;* Description: Waits for the USART1 Data Register to be
;* empty before loading it with mpr and transmitting the signal
;**********************************************************
TRANSMIT:
lds mpr2, UCSR1A
```

```
andi mpr2, 0b00100000
cpi mpr2, 0b00100000 ; Proceed if UDR1 is empty
breq EMPTY
rjmp TRANSMIT ; and ready to transmit data
EMPTY:
sts UDR1, mpr ; Send action code through UDR1

; Flash Freeze signal for 100ms as an indicator
in mpr2, PORTB ; Save previous action

out PORTB, mpr ; Flash Freeze signal

ldi waitcnt, 10
rcall Delay10ms ; Brief delay to flash signal

out PORTB, mpr2 ; Return to previous action

ret ; Return

;***********************************************************
;* Stored Program Data
;***********************************************************

;***********************************************************
;* Additional Program Includes
;***********************************************************
```