ECE 375 Lab 8

Remotely Operated Vehicle (USART)

Lab Time: Thursday 10:00 - 11:50

Jeremy Fischer

TA Signature

1 Introduction

The purpose of this lab was for the students to use all of the skills they have learned in lab this past term and incorporate them into one final project. The new AVR component learned in this lab was USART. Specifically, how to transmit and receive data via USART.

2 Remote

2.1 Internal Register Definitions and Constants

Below are the labels I gave definitions in this program. The Pin equates are used to decipher which button was pressed. Knowing this will tell the remote which command it should send, since each command is tied to a different action. The action codes at the bottom are all ORd with $80 since and action code must have a 1 as its MSB

```
.def mpr = R16                          ; Multi-Purpose Register
.def bttnPress = R17                    ; reg used to hold what button was pressed
.def sendCmmd = R18                     ; reg used to hold the cmmd to send
.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 mvFwdPin = 0                       ;Pin that will be 0 when mvFwdPin bttn is pressed
.equ mvBckPin = 1                       ;Pin that will be 1 when mvBckPin bttn is pressed
.equ trnRightPin = 4                    ;Pin that will be 4 when trnRightPin bttn is pressed
.equ trnLeftPin = 5                     ;Pin that will be 5 when trnLeftPin bttn is pressed
.equ haltPin = 6                        ;Pin that will be 6 when haltPin bttn is pressed
.equ freezePin = 7                      ;Pin that will be 7 when freezePin bttn is pressed
.equ roboAddr = 0b01101110      ;the Robot's address that is sent so it knows Cmmd is for it
.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 = ($80|1 << (EngEnR - 1)|1 << (EngEnL - 1))   ;0b11001000 Halt Action Code
.equ Freeze = (0b11111000)              ;freeze Action Code
```

2.2 Init()

The Init() function sets up the stack, sets Port D pins 0,1,4,5,6,7 to input and pull-up, and sets up USART1's transmitter to transmit with a buad-rate of 2400bps. Above, I excluded Port D's pin 2 and pin2 3 because RXD1 = PD2 and TXD1 = PD3.

2.3 Main()

In a busy waiting fashion the Main() function continuously pulls PIND and checks if a button was pressed. If the PIND is all ones, then no buttons were pressed, due to PORTD being set as

pull-up. If it is not all ones, then the read in value is compared with each possible button press and the respective action code is placed in the *sendCmmd* register and the Transmit() function is called.

2.4 Transmit()

Transmit first checks if there is a byte currently being transmitted. If there is, then transmit() loops until it is clear to load the robot's address into UDR1. Once the robot's address is sent, the action code that was loaded into the *sendCmmd* register is output to UDR1 to be transmitted.

3 Robot

3.1 Internal Register Definitions and Constants

The internal register definitions and constants in the Robot code are very similar to those defined in the Remote code, accept for the registers:

- *acceptNext*: This is a flag that is set when the byte received is the robot's address, meaning to accept the next incoming byte (the action).
- *recvdCmmd*: This register holds the value read in from UDR1, either the robot's address or an action code.
- *cmmd*: This register holds the actual command to be output to PORTB
- *timesFrozen*: This register holds the number of times the robot has been frozen. If it is more than three times then the robot stays frozen until it is reset.

```
.def acceptNext = R20                                   ;used as flag for receiving command
.def mpr = R16
.def waitcnt = R17                                      ;wait loop counter
.def recvdCmmd = R18
.def cmmd = R19
.def timesFrozen = R21                                  ;keeps track of times I've been frozen
.equ WTime = 100                                        ;(wait time in MS) / 10ms. So 1s
.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 maxTimeFrozen = 3                                  ;3 times frozen max
.equ FreezeSend = 0b01010101                            ;byte to send to freeze other robots
.equ BotAddress = 0b01101110                            ;The robots address
.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 << $EngEnL$)
.equ Freeze = 0b11111000 ;0b11111000 Freeze Action Code

3.2 Init()

The Init() function sets up the stack, sets Port D pins 0 and 1 to input and pull-up (left and right whisker), and sets up USART1's transmitter and receiver to a buad-rate of 2400bps. Next, interrupts INT0 and INT1 are set up to trigger on a falling edge when either the left or right whisker is hit. The receiver interrupt enable pin is also set up so that when USART receives something the ISR is triggered.

3.3 Main()

The main program repetitively outputs the *cmmd* register, which holds the robot's current command, out to PORTB.

3.4 HitRight()

HITRIGHT first pushes the registers, including SREG, to the stack to ensure the state is restored when the ISR is finished. HITRIGHT then backs up for a second by setting the MovBck command in PORTB and then calling the Wait subroutine. Then, HITRIGHT instructs the tekbot to turn left for a second by setting the TurnL command in PORTB and then calling the Wait subroutine. Finally, HITRIGHT pops the values off the stack back into their respective registers.

3.5 HitLeft()

HITLEFT first pushes the registers, including SREG, to the stack to ensure the state is restored when the ISR is finished. HITLEFT then backs up for a second by setting the MovBck command in PORTB and then calling the Wait subroutine. Then, HITLEFT instructs the tekbot to turn right for a second by setting the TurnR command in PORTB and then calling the Wait subroutine. Finally, HITLEFT pops the values off the stack back into their respective registers.

3.6 FreezeOthers()

The FreezeOthers() function first disables USART1's receiver and enables the transmitter. Then, FreezeOthers() checks if there is a byte currently being transmitted or received. If there is, then FreezeOthers() loops until it is clear to load the *FreezeSend* command into UDR1 to be transmitted. On FreezeOthers() exit, USART1's transmitter is disabled and the receiver is enabled.

3.7 CheckFreeze()

The CheckFreeze() function checks if the byte inside of *recvdCmmd* is the freeze signal sent from another robot. If it is, then CheckFreeze() turns off all interrupts, waits for five seconds, and then increments the *timesFrozen*. Finally, if *timesFrozen* is equal to *maxTimesFrozen* then the robot remains frozen in an infinite loop.

3.8 GetCMMD()

GetCMMD() runs through all possible action codes and compares them with the value in *recvdCmmd*. If there is a match, then the respective action to be output to the Robot is copied into the *cmmd* register which will be output to PORTB in the Main() function.

3.9 ExecCMMD()

ExecCMMD() is USART1's receiver's ISR. ExecCMMD() reads in the byte that is in UDR1 and then immediately calls the CheckFreeze() function to see if the read in byte was a freeze signal sent by another robot.

Next, the *acceptNext* register/flag is checked. If it isn't set then ExecCMMD() skips down to the NeedAddr() subroutine where it compares *recvdCmmd* with the robot's address. If the address matches, then the *acceptNext* flag is set so that the next read byte will be treated as an action for this robot.

If the *acceptNext* register/flag is checked, then the GetCMMD() function is called and the *acceptNext* flag is set back to 0.

3.10 WAIT()

WAIT exploits the fact that each instruction_i takes X_i processing time. WAIT ends up working out as 16 + 159975*waitcnt cycles or roughly waitcnt*10ms. So, for one second I set waitcnt to 100 since 100*10ms = 1000ms = 1 second.

4 Difficulties

The main obstacle I had with this lab was forgetting that I needed to disable the receiver when I wanted to transmit the freeze command to other robots in the area. It took me awhile to understand why it wasn't working!

5 Conclusion

This lab incorporated everything that we have learned in lab this term. It was awesome to see how far we have came since Lab-1! USART is widely used, so it was nice to understand how it is used and actually work with it hands on.

6 Source Code

Jeremy_Fischer_Lab8_Remote_Sourcecode.asm

```
;***********************************************************
;*
;* This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*  Author: Jeremy Fischer
;*    Date: Feb. 25th 2018
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = R16 ; Multi-Purpose Register
.def bttnPress = R17 ; reg used to hold what button was pressed
.def  sendCmmd = R18 ; reg used to hold the cmmd to send

.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 mvFwdPin = 0 ;Pin that will be 0 when mvFwdPin bttn is pressed
.equ mvBckPin = 1 ;Pin that will be 1 when mvBckPin bttn is pressed
.equ trnRightPin = 4 ;Pin that will be 4 when trnRightPin bttn is pressed
.equ trnLeftPin = 5 ;Pin that will be 5 when trnLeftPin bttn is pressed
.equ haltPin = 6 ;Pin that will be 6 when haltPin bttn is pressed
.equ freezePin = 7 ;Pin that will be 7 when freezePin bttn is pressed

;.equ  roboAddr = 0b01101110 ;the Robot's address that is sent so it knows Cmmd is for it
.equ  roboAddr = 42
; 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 =    ($80|1<<(EngEnR-1)|1<<(EngEnL-1)) ;0b11001000 Halt Action Code
.equ Freeze =  (0b11111000) ;freeze 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
;* - Init I/O Ports
;* - Init USART0
;* - Set baudrate at 2400bps
;* - Enable transmitter
;* - Set frame format: 8 data bits, 2 stop bits
;***********************************************************
INIT:
;Init Stack Pointer
ldi mpr, low(RAMEND) ;Low Byte of End SRAM Address
out SPL, mpr         ;Write byte to SPL
ldi mpr, high(RAMEND);High Byte of End SRAM Address
out SPH, mpr         ;Write byte to SPH

;Init ports
ldi mpr, $00    ;set portD to input
out DDRD, mpr
ldi mpr, 0b11110011    ;Set port D to pull up
out PORTD, mpr ;RXD1 = PD2, TXD1 = PD3

ldi mpr, (1<<U2X1)
sts UCSR1A, mpr

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

;USART0 frame format: 8 data, 2 stop bits, asynchronous
ldi mpr, (0<<UMSEL1 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C, mpr       ; UCSR0C in extended I/O space

ldi mpr, (1<<TXEN1)   ;enable the transmitter
```

```
sts UCSR1B, mpr


;***********************************************************
;* Main Program
;***********************************************************
MAIN:
;REMEMBER buttons are passive, therefore PINx = 0 means pressed.
in bttnPress, PIND ;read in the button presses
cpi  bttnPress, $FF
breq MAIN ;if nothing is pressed, loop

;go through all possible button presses to see if one is pressed.
;if the bit isn't cleared, meaning that button wasn't pressed,
;then skip the ldi command and check the next pin
sbrs bttnPress, mvFwdPin
ldi sendCmmd, MovFwd ;load MovFwd bit pattern

sbrs bttnPress, mvBckPin
ldi sendCmmd, MovBck ;load MovBck bit pattern

sbrs bttnPress, trnRightPin
ldi sendCmmd, TurnR ;load TurnR bit pattern

sbrs bttnPress, trnLeftPin
ldi sendCmmd, TurnL ;load TurnL bit pattern

sbrs bttnPress, haltPin
ldi sendCmmd, Halt ;load Halt bit pattern

sbrs bttnPress, freezePin
ldi sendCmmd, Freeze ;load Halt bit pattern

rcall TRANSMIT  ;send the new command to robot

rjmp MAIN

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

;-----------------------------------------------------------
; TRANSMIT:
; Desc: Transmits the robots address to the robot so it knows
; if the next byte coming is meant for it.
; Then, transmit the newly read command to the robot
```

```
;-----------------------------------------------------------
TRANSMIT:
push mpr

cpi sendCmmd, $00
breq EXITTRANSMIT  ;if there is no command, exit


ROBOTADDR:
lds mpr, UCSR1A
sbrs  mpr, UDRE1  ;If byte is still in UDR, wait till it shifts to transmit reg
rjmp  ROBOTADDR
ldi mpr, roboAddr
sts   UDR1, mpr     ;Load the roboAddr to UDR0 to to sent.

SENDDATA:
lds mpr, UCSR1A
sbrs  mpr, UDRE1  ;If byte is still in UDR, wait till it shifts to transmit reg
rjmp  SENDDATA
sts   UDR1, sendCmmd  ;Load the sendCmmd to UDR0 to to sent.

EXITTRANSMIT:
;cbr mpr, UDRE1
;sts UCSR1A, mpr
ldi sendCmmd, $00  ;reset the sendCmmd
pop mpr
ret
```

Jeremy_Fischer_Lab8_Robot_Sourcecode.asm

```
;***********************************************************
;*
;* This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*  Author: Jeremy Fischer
;*    Date: Feb. 25th 2018
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def  acceptNext = R20 ;used as flag for recieving command
.def  mpr = R16
.def  waitcnt = R17 ;wait loop counter
.def  recvdCmmd = R18
.def  cmmd = R19
.def  timesFrozen = R21  ;keeps track of times I've been frozen

.equ  WTime = 100 ;(wait time in MS) / 10ms. So 1s

.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  maxTimeFrozen = 3 ;3 times frozen max
.equ  FreezeSend = 0b01010101 ;byte to send to freeze other robots

.equ  BotAddress = 43 ;The robots address

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

```
.equ Freeze =  (0b11111000) ;0b11111000 Freeze Action Code

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

;***********************************************************
;* Interrupt Vectors
;***********************************************************
.org $0000 ;Beginning of IVs
rjmp  INIT ; Reset interrupt
.org $0002 ;INT0 => pin0, PORTD
rcall HITRIGHT ;run hitright ISR
reti
.org $0004 ;INT1 => pin1, PORTD
rcall HITLEFT ;run hitleft ISR
reti
.ORG $003C ;RXC was triggered
rjmp  EXECCMMD ;Execute the command
reti
.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Initialize Stack Pointer
ldi R16, low(RAMEND)  ;Low Byte of End SRAM Address
out SPL, R16  ;Write byte to SPL
ldi R16, high(RAMEND)  ;High Byte of End SRAM Address
out SPH, R16  ;Write byte to SPH

;Init ports
ldi mpr, (1<<PE1)   ;Set Port E pin 0 (RXD0) for input (implicitly)
out DDRE, mpr ;Port E pin 1 (TXD0) for output
;Initialize Port B for output
ldi   mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL)
out   DDRB, mpr          ;Set PortB to be output

; Initialize Port D for input
ldi   mpr, (0<<WskrL)|(0<<WskrR)
out   DDRD, mpr          ;Set PortD to be input
ldi   mpr, (1<<WskrL)|(1<<WskrR)
out   PORTD, mpr         ;Set PortD to be pull-up
```

```
;Initialize external interrupts (to trigger on falling edge)
ldi   mpr, (1<<ISC11)|(0<<ISC10)|(1<<ISC01)|(0<<ISC00) ;'10' is falling edge
sts   EICRA, mpr         ;Use sts, EICRA is in extended I/O space

;Set the External Interrupt Mask
ldi   mpr, (1<<INT0)|(1<<INT1)  ;setting EIMSKx to 1 enables it to be deteced
out   EIMSK, mpr



;Set baud rate at 2400 bps
ldi mpr, (1<<U2X1)
sts UCSR1A, mpr

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

;USART1 frame format: 8 data, 2 stop bits, asynchronous
ldi mpr, (0<<UMSEL1 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C, mpr       ; UCSR0C in extended I/O space

;enable the reciever and the RX Complete Interrupt Enable
ldi mpr, (1<<TXEN1)|(1<<RXEN1)|(1<<RXCIE1)
;ldi mpr, (1<<RXEN1)|(1<<RXCIE1)

sts UCSR1B, mpr
ldi waitcnt, Wtime

ldi acceptNext, 0
ldi cmmd, MovFwd
;Turn on interrupts
sei

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
; Move Robot Forward
out   PORTB, cmmd ;tell TekBot to move forward
rjmp  MAIN ;infinite loop. End of program.

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

```
;-----------------------------------------------------------
; 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 equation
;  for the number of clock cycles in the wait loop:
; ((3 * R19 + 3) * R18 + 3) * waitcnt + 13 + call
;-----------------------------------------------------------
WAIT: ;Begin a function with a label
push R18
push R19

OLoop:
ldi R18, 224 ;Load middle-loop count
MLoop:
ldi R19, 237   ;Load inner-loop count
ILoop:
dec R19   ;Decrement inner-loop count
brne Iloop  ;Continue inner-loop
dec R18    ;Decrement middle-loop
brne Mloop  ;Continue middle-loop
dec waitcnt    ;Decrement outer-loop count
brne OLoop  ;Continue outer-loop

pop R19
pop R18
ret  ;Return from subroutine

;-----------------------------------------------------------
; EXECCMMD:
; Desc: First checks the Address sent. If it is this bot's
; address, then the next byte, the command, is executed
;-----------------------------------------------------------
EXECCMMD:
;Save variable by pushing them to the stack
push  mpr
push  waitcnt

lds recvdCmmd, UDR1

rcall CHECKFREEZE ;check if recvdCmmd was freeze from other robot

cpi acceptNext, 1
brne NEEDADDR ;Get the bot address
```

```
rcall GETCMMD ;execute command
ldi acceptNext, 0; ;reset acceptNext
rjmp EXIT ;jmp to exit

NEEDADDR:
cpi recvdCmmd, BotAddress
brne EXIT ;if address != bot, exit
ldi acceptNext, 1 ;if address == bot, get command next byte recieved
EXIT:
;Restore variable by popping them from the stack in reverse order

pop   waitcnt
pop   mpr
ret ;return to caller

;-----------------------------------------------------------
; GETCMMD:
; Desc: executes the command. recvdCmmd holds the command
; MovFwd = 0b01100000 Move Forward Action Code
; MovBck = 0b00000000 Move Backward Action Code
; TurnR  = 0b01000000 Turn Right Action Code
; TurnL  = 0b00100000 Turn Left Action Code
; Halt   = 0b10010000 Halt Action Code
;-----------------------------------------------------------
GETCMMD:
push mpr

ldi mpr, ($80|MovFwd) ;is this the move forward cmmd
cp recvdCmmd, mpr  ;Compare Skip if Equal
breq FWD


ldi mpr, ($80|MovBck) ;is this the move backward cmmd
cp recvdCmmd, mpr ;Compare Skip if Equal
breq BCK

ldi mpr, ($80|TurnR) ;is this the turn right cmmd
cp recvdCmmd, mpr ;Compare Skip if Equal
breq TRNR

ldi mpr, ($80|TurnL) ;is this the turn left cmmd
cp recvdCmmd, mpr ;Compare Skip if Equal
breq TRNL

ldi mpr, ($80|Halt) ;is this the halt cmmd
```

```
cp recvdCmmd, mpr ;Compare Skip if Equal
breq HLT

ldi mpr, (Freeze) ;is this the Freeze cmmd
cp recvdCmmd, mpr ;Compare Skip if Equal
breq FRZ

FWD:
ldi cmmd, MovFwd ;yes, then load MovFwd into the cmmd
rjmp EXITCMMD
BCK:
ldi cmmd, MovBck ;yes, then load MovBck into the cmmd
rjmp EXITCMMD
TRNR:
ldi cmmd, TurnR ;yes, then load TurnR into the cmmd
rjmp EXITCMMD
TRNL:
ldi cmmd, TurnL ;yes, then load TurnL into the cmmd
rjmp EXITCMMD
HLT:
ldi cmmd, Halt ;yes, then load Halt into the cmmd
rjmp EXITCMMD
FRZ:
rcall FREEZEOTHERS
rjmp EXITCMMD

EXITCMMD:
pop mpr
ret



;-----------------------------------------------------------
; CHECKFREEZE:
; Desc: Checks if I just got frozen
;-----------------------------------------------------------
CHECKFREEZE:
push mpr
push recvdCmmd
push cmmd

cpi  recvdCmmd, FreezeSend ;if I did just get the freeze signal from another
brne EXITCHECK

;It is the Freeze command
cli   ;turn off interrrupts
```

```
ldi cmmd, Halt ;load Halt into the cmmd
out PORTB, cmmd ;Halt the robot

inc timesFrozen ;increment the times ive been frozen
cpi timesFrozen, maxTimeFrozen
breq STAYFROZEN

ldi   waitcnt, 250 ;wait for 2.5 seconds. (250 * 10ms) = 2500ms
rcall WAIT
ldi   waitcnt, 250 ;wait for 5.5 seconds. (250 * 10ms) = 2500ms
rcall WAIT

;I havn't reached maxTimeFrozen yet
rjmp EXITCHECK ;exit the check

STAYFROZEN:
rjmp STAYFROZEN ;stay here till reset

EXITCHECK:
pop cmmd
out PORTB, cmmd ;load what robot was ding previously

pop recvdCmmd
pop mpr
sei ;reset the interrupt flag
ret

;-----------------------------------------------------------
; FREEZEOTHERS:
; Desc: Send Freeze signal to other robots
;-----------------------------------------------------------
FREEZEOTHERS:
push mpr

; Disable the receiver
ldi mpr, (0<<RXCIE1)|(1<<TXCIE1)|(0<<RXEN1)|(1<<TXEN1)
sts UCSR1B, mpr

TRANSMIT:
lds mpr, UCSR1A
sbrs  mpr, UDRE1  ;If byte is still in UDR, wait till it shifts to transmit reg
rjmp  TRANSMIT
ldi mpr, FreezeSend
sts   UDR1, mpr     ;Load the roboAddr to UDR0 to to sent.

; Enable the receiver
```

```
ldi mpr, (1<<RXCIE1)|(1<<TXCIE1)|(1<<RXEN1)|(1<<TXEN1)
sts UCSR1B, mpr

pop mpr
ret


;-----------------------------------------------------------
; HITLEFT:
; Desc: This function is an ISR that handles the tekbot
; hitting its left trigger. The tekbot backs up
; for one second, then turns right for one second
;-----------------------------------------------------------
HITLEFT: ; Begin a function with a label
;Save variable by pushing them to the stack
push  mpr
push  waitcnt
in    mpr, SREG
push  mpr

; Move Backwards for a second
ldi   mpr, MovBck
out   PORTB, mpr
ldi   waitcnt, WTime
rcall Wait

; Turn left for a second
ldi   mpr, 1<<(EngDirL)
out   PORTB, mpr
ldi   waitcnt, WTime
rcall Wait

;clear out any staged INT0 or INT1
sbr mpr, (1<<WskrR)|(1<<WskrL) ;setting bits in flag register to 1 since pull up resistor
out EIFR, mpr ;cancel out any staged interrupts

;Restore variable by popping them from the stack in reverse order
pop   mpr
out   SREG, mpr
pop   waitcnt
pop   mpr
ret ;return to caller

;-----------------------------------------------------
; HITRIGHT:
; Desc: This function is an ISR that handles the tekbot
```

```
; hitting its right trigger. The tekbot backs up
; for one second, then turns left for one second
;-----------------------------------------------------------
HITRIGHT: ; Begin a function with a label
;Save variable by pushing them to the stack
push  mpr
push  waitcnt
in    mpr, SREG
push  mpr

; Move Backwards for a second
ldi   mpr, MovBck
out   PORTB, mpr
ldi   waitcnt, WTime
rcall Wait

; Turn left for a second
ldi   mpr, 1<<(EngDirR)
out   PORTB, mpr
ldi   waitcnt, WTime
rcall Wait

;clear out any staged INT0 or INT1
sbr mpr, (1<<WskrR)|(1<<WskrL) ;setting bits in flag register to 1 since pull up resistor
out EIFR, mpr ;cancel out any staged interrupts

;Restore variable by popping them from the stack in reverse order
pop   mpr
out   SREG, mpr
pop   waitcnt
pop   mpr
ret ;return to caller
```