ECE 375 Lab 8

Remotely Operated Vehicle

Lab Time: Monday 4-6

Austin Jay Wilmoth MXIV
Donald Henry Joyce III

1 Introduction

The purpose of this lab is to learn about USART by using USART to let one avr board control another. The transmitter will act as remote control for the second avr board. While the receiver will run the bot and have the bumber bot behavior. The lab gives insight on how communication between two or more avr boards can be implemented. This shows us that from simpler hardware a complex network can be made that accomplish a much larger goal.

2 Program Overview

This program is connecting to two separate avr boards together using infrared and USART communication. One board acts as the transmitter, or remote control. The other board acts as the receiver, running the bot based on the remote control commands. The transmitter sends two 8 bit messages to the receiver. The first message is a botID which acts as a key to make sure that only the correct bot is acting on the commands revived. The second message is the command message. The command is an 8 bit value that tells the avr/bumper bot what command to do. There are multiple commands that the bot can do these are move backwards, move forwards, move right, move left, halt, and freeze. The bot will perform these actions as well as the bumper bot functionality of moving backwards and turning when hitting a wisker on a wall. When the button/whisker is pressed/hit the action will be interrupted and bumper bot action will take over. There is also a freeze function which is a command that is sent from the transmitter to the receiver and then the receiver transmits to all other receivers to halt for five seconds.

3 Transmit's Initialization Routine

For the transmitter initialization routine the first thing that is done is to set up the stack pointer. The second thing that is done is setting up Port B and Port D. Port B was only set up for debugging. The next thing that is done is set up the botID value which is used to confirm that the correct board is being communicated with. The next thing that is done is setting the baud rate which is calculated using the double baud rate equation. This value is loaded into UBRR1. The next that is set is UCSR1A register. All that is needed to be loaded into this register is a one for the Double baud rate which is located at the U2X1 bit. The next thing that is done is to set up the UCSR1B register. All that is needed to be into the this register is one bit for Transmitter enabled. This is done by setting up the TXEN1 bit. The next thing to set up the UCSR1C register. This sets up the 8 data bits and 2 stop bits set up with the USBS1, UCSZ11, and UCSZ10 all being set to one. The next and final thing to do for the initialization routine is use sei which sets up all the interrupts.

4 Receive's Initialization Routine

The Receive initialization routine is very close to the Transmit initialization routine. The difference is in the UCSR1B register in that the receiver is enabled as well transmit interrupt for the freeze action. This is done by setting the RXCIE1 and RXEN1 bits to 1. The difference between Transmit and Receive is transmit uses polling and receive uses interrupts. So receive also has to use EIMSK

to set which buttons get masked or not. Only two buttons are used so the value loaded into EIMSK is 0b00000011. The next thing that is done is set up EICRA which controls when the interrupts will be detected. 0b00001010 is loaded into EICRA because 10 sets the interupt to be detected on the falling edgeand it needs to be done for interrupt 0 and interrupt 1. The next thing that is done X is pointed to BUFFER which is the area in memory that holds the received value.

5 Transmit's Main Routine

The main Subroutine in the transmit program is set up to using polling and check which buttons have been pressed on the board. It first takes in the value from PinD and then checks it against each button and depending on which button was pressed is calls the subroutine that runs the corresponding command. Once done with the command or no commands are matched it then jumps back to the start of main.

6 Receive's Main Routine

The main routine in the receive program is set up to do nothing but loop until an interrupts comes. This is because everything is set up using interrupt's so the program just needs to wait.

7 Transmit's button Subroutines

In the transmit program there are six buttons set up to send commands to the receiver. These are MoveForward, MoveBackwards, TurnRight, TurnLeft, Hault, and Freeze. These all work the same. Each one sets the register transfer to the bits that represent the code for the action and then call the transmit subroutine. The register transfer is then used by the transmit subroutine to send the code to the receiver.

8 Transmit Subroutine

The Transmit subroutine in the transmit program is set up to send first an address and then what ever is stored in the register titled transfer to the receiver. It does this by first checking that the bit UDRE1 is set which corresponds to whether or not the UDR1 buffer is cleared. If it is not then the program keeps looping but if the bit is set then address is transferred to UDR1 to be transmitted. The subroutine then checks again if UDRE1 is set, meaning the last transmission is done. Once it is done and the loop stops the value that is stored transfer is sent to UDR1 to be transmitted. The subroutine has now sent all the data that it needs to but is not done yet. The last thing that is done is a loop is made that checks if TXC1 has been set. This means that all the data has officially been sent to the receiver. Once that is done the TXC1 bit is cleared and the subroutine is done.

9 Receive's Movement Subroutines

Receive's movement subroutine includes MoveForwared, MoveBackward, TurnRight, TurnLeft, Halt_Sub. This happens after the command is determined and each one of these functions load the corresponding value into PORTB to control the movement of the Bot. Movement Subroutines also included RightBump and LeftBump. These control movement but are different than the ones talked about above because they are subroutines that correspond to interrupts so when two most right buttons/bumpers are triggered these subroutines are called and the movement of the bot is interrupted. These functions stop the prior command of movement and moves backwards for one second, then turns the opposite direction of the bumper that is triggered for one second then it moves forwards. These functions uses the Wait function and they clear the interrupts using EIFR.

10 Receive subroutine

The Receive function is called when the receive USART interrupt is activated. This function takes the value in UDR1 which is the value that is transmitted to the board. Then sets the RXC1 bit in UCSR1A register which says the board is done receiving to zero so it ready to receive again. This function then compares if the value that is received is the correct botID and if it is then it goes to REC2 to wait until the board is done receiving the next value and then takes that value and checks which command is called. But if the botID is not the correct value then it checks if the value is equal to 0b01010101 and if it is it jumps to GetFreezed. If it is not then then the value is not from the correct transmitter and it just calls reti so that it will wait for a new value. GetFreezed function will tell the bot to freeze for 5 seconds.

11 Freeze Tag

The way that the Freeze feature works is that a command is sent to the receiver by the transmitter in the normal fashion but when the receiver sees that it is the code for freeze it transmits a 8 bit code to all receivers in the area forcing them to freeze for 5 seconds. This works by when the freeze subroutine is called the bot then transmits in the same way as the transmitter minus sending the second 8 bits out to the other bots. The other bots, or this bot if the freeze commands is sent by other bots, gets this command and in the receive subroutine there is an extra branch set up to detect the freeze code and then a subroutine that is called getFreezed is called that simply freezes the bot for 5 seconds or halts forever if it has been frozen 3 times. To Make sure that the bot that sent the freeze command is not frozen the receive interupt is disabled while transmiting the freeze code.

12 Difficulties

At first this lab went fairly smooth, we coded a large amount of it and felt good, until we ran into it simply not working and where unsure why. This came down to us not fully understanding how USART worked. Once we got a much better grasp on how USART worked we continued forward until we got stuck on receive. What we where trying to do was store both the address and the code into a location in memory using X. The problem was that when ever we did this the data

would not get stored in x and instead was lost. We could not figure out why this was happening so eventually we decide to come from a diffident angle. This involved checking the address inside the receive subroutine instead of outside of it so that we could override the register that stored the received data. Once getting past that everything went fairly smooth.

13 Conclusion

The conclusion was that this lab help us better understand how USART worked. But it also helped combine all the things we have leaned in lab to let us create a program that did something noteworthy.

14 Source Code - Transmit

```
;***********************************************************
;*
;*  Transmit
;*
;* This Program is used to send commands from one AVR board
;*
;* to another that is programed to controll a robot
;*
;***********************************************************
;*
;*  Author: Austin Wilmoth and Donald Joyce
;*    Date: March 11, 2019
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def address = r17 ; holds the bot address
.def transfer = r18 ; holds the command to be transfered

.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
.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 Co
.equ Freeze =  0b11111000                        ;0b11111000 Freeze Action Cod
;***********************************************************
;* 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

;Ports
ldi mpr, $FF ; B is set up for output to be able to control
out DDRB, mpr ; the lights, for testing

ldi mpr, $00 ; D is set up for input to both take in button
out DDRD, mpr ; presses and to recive the TXD1 interupt

ldi mpr, $FF ; PortD is set to all ones as the buttons are
out PortD, mpr ; passive

ldi address, $4D ; this is the bot address


;USART1
;Set baudrate at 2400bps
ldi mpr, high(832)
```

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

;Enable transmitter
ldi mpr, (1<<U2X1) ; sets to double boadrate
sts UCSR1A, mpr

ldi mpr, (1<<TXEN1) ; set to enable transmition
sts UCSR1B, mpr

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

sei ; enable interupts

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
in mpr, PIND ; read in from the buttons
cpi mpr, 0b11111110 ; check if the button 0 is pressed
brne Bck
rcall MoveForward
rjmp MAIN
Bck:
cpi mpr, 0b11111101 ; check if button 1 is pressed
brne TrnR
rcall MoveBackward
rjmp MAIN

TrnR:
cpi mpr, 0b11101111 ; check if button 4 is pressed
brne TrnL ; skip 2 and 3 as the are used by USART
rcall TurnRight
rjmp MAIN
TrnL:
cpi mpr, 0b11011111 ; check if button 5 is pressed
brne Hlt
rcall TurnLeft
rjmp MAIN
Hlt:
cpi mpr, 0b10111111 ; check if button 6 is pressed
brne Fre
rcall Halt_Sub
```

```
rjmp MAIN
Fre:
cpi mpr, 0b01111111 ; check if button 7 is pressed
brne Skip2
rcall Freeze_sub
rjmp Main
Skip2:

rjmp MAIN ; otherwise start over

;***********************************************************
;* Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------
; Sub: MoveForward
; Desc: Sets the code to be transfered to be the code to move the
;       the bot forward
;-----------------------------------------------------------
MoveForward:
ldi transfer, MovFwd
rcall Transmit

ret

;-----------------------------------------------------------
; Sub: MoveBackwards
; Desc: Sets the code to be transfered to be the code to move the
;       the bot backwards
;-----------------------------------------------------------
MoveBackward:
ldi transfer, MovBck
rcall Transmit

ret

;-----------------------------------------------------------
; Sub: TurnRight
; Desc: Sets the code to be transfered to be the code to make the
;       bot turn right
;-----------------------------------------------------------
TurnRight:
ldi transfer, TurnR
rcall Transmit

ret
```

```
;-----------------------------------------------------------------
; Sub: TurnLeft
; Desc: Sets the code to be transfered to be the code to make the
;       bot turn Left
;-----------------------------------------------------------------
TurnLeft:
ldi transfer, TurnL
rcall Transmit

ret
;-----------------------------------------------------------------
; Sub: Halt_Sub
; Desc: Sets the code to be transfered to be the code to make the
;       bot stop moving
;-----------------------------------------------------------------
Halt_Sub:
ldi transfer, Halt
rcall Transmit

ret

;-----------------------------------------------------------------
; Sub: Freeze_Sub
; Desc: Sets the code to be transfered to be the code to make the
;       bot send out the freeze command to other bots
;-----------------------------------------------------------------
Freeze_sub:
ldi transfer, Freeze
rcall Transmit

ret

;-----------------------------------------------------------------
; Sub: Transmit
; Desc: Sends both the address of the bot and the command that is
;       to be done by the bot
;-----------------------------------------------------------------
Transmit:
LDS mpr, UCSR1A ; check that the transmit buffer is empty
SBRS mpr, UDRE1
rjmp Transmit ; if not loop

STS UDR1, address ; if so put the address into UDR1 to be sent

Loop_1:
LDS mpr, UCSR1A ; check again that the transmit buffer if
```

```
SBRS mpr, UDRE1 ; empty meaning that the first thing is sent
rjmp Loop_1

STS UDR1, transfer ; if it is send that command

Loop_2:
LDS mpr, UCSR1A ; check if the data has finished transmiting
SBRS mpr, TXC1
rjmp Loop_2

cbr mpr, TXC1 ; if so clear the finished bit
STS UCSR1A, mpr


ret

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

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

15 Source Code - Receive

```
;***********************************************************
;*
;* Receive
;*
;* Enter the description of the program here
;*
;* This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*  Author: Austin Wilmoth and Donald Joyce
;*    Date: Enter Date
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
```

```
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register

.def address = r17
.def command = r18

.def waitcnt = r19 ; Wait Loop Counter
.def ilcnt = r20 ; Inner Loop Counter
.def olcnt = r21 ; Outer Loop Counter
.def Fcount = r22 ; counts how many times bot has been frozen

.equ FTime = 255
.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 = $4D;(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 MovFwdC =  ($80|1<<(EngDirR-1)|1<<(EngDirL-1)) ;0b10110000 Move Forwar
.equ MovBckC =  ($80|$00) ;0b10000000 Move Backward Action Code
.equ TurnRC =   ($80|1<<(EngDirL-1)) ;0b10100000 Turn Right Action Code
.equ TurnLC =   ($80|1<<(EngDirR-1)) ;0b10010000 Turn Left Action Code
.equ HaltC =    ($80|1<<(EngEnR-1)|1<<(EngEnL-1)) ;0b11001000 Halt Action C
.equ FreezeC = 0b11111000
;***********************************************************
;* 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
.org $0002
rjmp LeftBump
;- Right whisker
.org $0004
rjmp RightBump
;- USART receive
.org $003C
rjmp Receive
; some subroutine?
.org $0046 ; End of Interrupt Vectors
;rjmp Receive
;***********************************************************
;* 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, (0<<PD0)|(0<<PD1)|(0<<PD2) ;This sets up the buttons for input for
out DDRD, mpr ;the value is loaded into DDRD to set the port as input

ldi mpr, $03 ; This set the port for input
out PORTD, mpr

ldi mpr, $FF ;Sets DDRB as output
out DDRB, mpr

ldi mpr, $00 ;sets Port b as output which lets the led be turned on
out PortB, mpr

ldi Fcount, 0

;USART1
;Set baudrate at 2400bps
ldi mpr, high(832) ;set the baud rate using the double baud rate equation
sts UBRR1H, mpr
ldi mpr, low(832)
```

```
sts UBRR1L, mpr

ldi mpr, (1<<U2X1) ;sets the baud rate bit
sts UCSR1A, mpr

ldi mpr, (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1) ;sets the Receive enable and tra
sts UCSR1B, mpr

ldi mpr, (1<<USBS1)|(1<<UCSZ11)|(1<<UCSZ10) ;sets the 8 bit rate and 2 stop
sts UCSR1C, mpr

;External Interrupts
;Set the External Interrupt Mask
ldi mpr, 0b00000011 ;set up the two buttons as inputs or as bumpers
out EIMSK, mpr
;Set the Interrupt Sense Control to falling edge detection
ldi mpr, 0b00001010 ;set up the two buttons interrupts as falling edge dete
sts EICRA, mpr

sei
;Other
ldi XL, low(BUFFER) ;set up X and Y to point to the buffer which stores the
ldi XH, high(BUFFER)
rcall Resetx ;resets the value that x pointing to


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


rjmp MAIN ;infinity loops so that it waits for just the interrupts

;***********************************************************
;* Functions and Subroutines
;***********************************************************
MoveForward:
ldi mpr, MovFwd ;loads the value of the command into port B
out PORTB, mpr
ret

MoveBackward:
ldi mpr, MovBck ;loads the value of the command into port B
out PORTB, mpr
ret
```

```
TurnRight:
ldi mpr, TurnR ;loads the value of the command into port B
out PORTB, mpr
ret

TurnLeft:
ldi mpr, TurnL ;loads the value of the command into port B
out PORTB, mpr
ret

Halt_Sub:
ldi mpr, Halt ;loads the value of the command into port B
out PORTB, mpr
ret

;-----------------------------------------------------------------
; 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
reti ; Return from subroutine

GetFreezed:
push command
in command, PORTB
```

```
inc Fcount

rcall Halt_sub

cpi Fcount, 3
breq FreezeForever

ldi waitcnt, FTime ; Wait for 1 second
rcall Wait
ldi waitcnt, FTime
rcall Wait

out PORTB, command
pop command

ret

FreezeForever:
rjmp FreezeForever

Freezer: ;this function transmits the freeze value is tranmitted to the oth
;ldi mpr, 0b11111111 ;remove this just for testing
;out PORTB, mpr

ldi mpr, (1<<RXCIE1)|(0<<RXEN1)|(1<<TXEN1) ;sets the Receive enable and tra
sts UCSR1B, mpr

LDS mpr, UCSR1A ;loads the value UCSR1A into mpr
SBRS mpr, UDRE1 ;checks if the UDRE1 bit says that the buffer is cleared
rjmp Freezer ;if it is not cleared then it keeps looping until the it is
ldi mpr, 0b01010101 ;if the buffer is cleared then the command is loaded in
STS UDR1, mpr


Loop_2:
LDS mpr, UCSR1A ;checks if the transmit is done
SBRS mpr, TXC1
rjmp Loop_2 ; if the transmit is not done then it loops until it is

cbr mpr, TXC1 ;clears the transmit bit in UCSR1A
STS UCSR1A, mpr

ldi mpr, (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1) ;sets the Receive enable and tra
sts UCSR1B, mpr
```

```
ret

RightBump:
push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;

ldi mpr, (0<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1) ;sets the Receive enable and tra
sts UCSR1B, 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

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

ldi mpr, $FF
out EIFR, mpr

ldi mpr, (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1) ;sets the Receive enable and tra
sts UCSR1B, mpr

pop mpr ; Restore program state
out SREG, mpr ;
pop waitcnt ; Restore wait register
pop mpr ; Restore mpr
reti

LeftBump:
push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;

ldi mpr, (0<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1) ;sets the Receive enable and tra
```

```
sts UCSR1B, 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

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

ldi mpr, $FF
out EIFR, mpr

ldi mpr, (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1) ;sets the Receive enable and tra
sts UCSR1B, mpr

pop mpr ; Restore program state
out SREG, mpr ;
pop waitcnt ; Restore wait register
pop mpr ; Restore mpr
reti

Receive:

lds mpr, UDR1 ;loads the received value into the buffer
st X, mpr

lds mpr, UCSR1A ;clears the receive done bit in UCSR1A
cbr mpr, RXC1
STS UCSR1A, mpr

ld r3, X ;checks if the botID is correct one
ldi mpr, BotAddress

cp mpr, r3 ;if the other botID is not correct it jumps to Skip
brne Skip

Rec2:
```

```
lds   mpr,    UCSR1A ;checks if the RXC1 bits which means it is done receiv
sbrs  mpr,    RXC1
rjmp  Rec2 ;if it is not done then it loops until it is
lds   mpr,    UDR1 ;loads the value into the memory buffer
st    X,      mpr
lds   mpr,    UCSR1A
cbr   mpr,    RXC1 ;sets the bit so it is cleared meaning it is not done re
sts   UCSR1A, mpr
rcall Commands ;calls command to see if which command was called


Skip:
ldi mpr, 0b01010101 ;if the wrong botID is received it checks if the freeze
cp mpr, r3
brne Skip3 ;if it was not received then it skips to Skip3
rcall GetFreezed

Skip3:
reti ;returns to main to wait again

ResetX:
ldi XL, low(BUFFER) ;resets the X value to initial value that it was set to
ldi XH, high(BUFFER)
ret

Commands:

ld mpr, x ;loads the value in the memory buffer

cpi mpr, MovFwdC ;checks if the value means MovFwd command is called
brne Bck ;if it is not then skips to check the next one
rcall MoveForward
Bck:
cpi mpr, MovBckC ;checks if the value means MovBck command is called
brne TrnR ;if it is not then skips to check the next one
rcall MoveBackward

TrnR:
cpi mpr, TurnRC ;checks if the value means TurnR command is called
brne TrnL ;if it is not then skips to check the next one
rcall TurnRight

TrnL:
cpi mpr, TurnLC ;checks if the value means TurnL command is called
brne Hlt ;if it is not then skips to check the next one
rcall TurnLeft
```

```
Hlt:
cpi mpr, HaltC ;checks if the value means Halt command is called
brne Fre ;if it is not then it skips to check the next one
rcall Halt_Sub
Fre:
cpi mpr, FreezeC ;checks if the value means Freeze command is called
brne Skip2 ;if it is not then it skips to check the next one
rcall Freezer
Skip2:
rcall ResetX ;resets x to point to the memory buffer
ret
;***********************************************************
;* Stored Program Data
;***********************************************************
.dseg
.org $0100
BUFFER: ;memory buffer
.byte 4

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