ECE 375 LAB 8

Remotely Operated Vehicle

Lab Time: Tuesday 4-6

Sonia Camacho

Owen Markley

INTRODUCTION

The purpose of this final lab is to get us started with the USART and the ATMEGA 128 board. This lab was unlike any other one we have done before, this time we have 2 boards! The objective of this lab was to get our two AVR boards to connect with each other and communicate with one another. We use both boards to do different things, the first board is the remote for our "robot" (just our AVR boards) which is the transmitter which will be controlling what the receiver which is our robot and this will result in actions being done such as the bump bot action. We then had to implement a version of freeze tag which would still be controlled by the remote AVR board.

PROGRAM OVERVIEW

PART 1 – REMOTE CONTROL

In the first part of this lab we were told to pretend we are the TekToy corporation and we want to create a new toy for that is remote controlled. We want to use our old AVR boards that we have lying around, and we will be implementing a system that uses USART methods to interact one board between another. The first step is to build a proof of concept of the robot so that we could pitch the idea to the management. One is used as a remote and one is the robot. We were instructed to follow the following 6 steps which are; 1. You will need to configure the USART module on the robot board and on the remote board. Also, although the ATmega128's USART modules can communicate at rates as high as 2 × 106 bits per second (2 Mbps), the built- in IR transmitter and receiver cannot work this fast. Instead, you will use the (relatively) slow baud rate of 2400 bits per second. 2. We needed to configure the bot to move forward, move backward, turn right, turn left, and halt. A user should be able to select from these actions using the pushbuttons on the remote. Once the robot receives an action from the remote, it must continue performing that same action until a different action is received, without needing to receive the same action repeatedly from the remote. 3. To ensure that our bots can work properly while multiple robots can successfully run at the same time in the same room, we set the bot a distinct address, which the remote will transmit along with every selected action. 4. Our bots will send a 16 bit packet the first 8-bit value will be a "robot address" byte, which indicates which specific robot the packet is intended for. The second 8-bit value will be an "action code" byte, which indicates which action the user wants the robot to take. 5. Move Forward: 0b10110000, Move Backward 0b10000000, Turn Right 0b10100000, Turn Left 0b10010000, Halt 0b11001000, Future Use 0b11111000 6. Our robot must also perform the BumpBot behavior from lab 1.

PART 2 – FREEZE TAG

In the second part of this lab we were told to spice up the robot a bit and make it play freeze tag with the other AVR board. We are given a set of instructions to follow which start with 1. send the robot address first, and then the freeze action code and the specified address is 0b11111000. 2. When a robot receives a packet from its remote containing the freeze action code, the robot must immediately transmit a standalone 8-bit "freeze signal", 0b01010101. 3. Any robot that receives an 8-bit freeze signal should "freeze" for five seconds, except for the robot that just sent the freeze signal itself. 4. This step is very important our bot should go back to what it was doing before it was frozen but when the bot is frozen 3 times it should stop working until it is reset in other words it should remain in a frozen state until told otherwise.

Transmitter Routines

****

Initialization Routine

The initialization routine was mainly given from in the lab slides where we were given the frame format which wanted the data frame for 8 bits and the stop bit for 2 bits and the parity bit should be disabled and the asynchronous operation should be clarified. We also need a double data rate and to receive data sampled at the falling edge and we needed the baud rate to be 2400 bits per second. We also need the BotID to NOT be $2A because that is what the TA will use to test so we set ours up with 1A. After going to Justin Goins office hours, we figured out that almost all of these are by default set but just to be sure we made sure to set everything as the lab's slides indicated they wanted. We had to read over the data sheet a few times to make sure we were writing everything correctly. One of the major points was that we needed to make sure we were writing in with the high byte first for the baud rate because initially we were writing the low byte and this would activate the buffer and then create an issue where we had been writing garbage values into our 16 bit spot. We set up the different buttons that we needed to use in the rest of the program so we had to specify which buttons would be for what actions.

Main Routine

The Main routine is where we had to check and see what buttons were pressed and this would determine what function call, we were going to do. We used polling in the main for the forward backward left and right and for the freeze and halt we used interrupts.

Forward Routine

The forward routine is where we displayed the LED lights on the transmitter board to show a certain pattern and then we had to transmit the robot ID and then transmit the action code which is the part that the other board will check and verify the code and then perform the certain action

Back Routine

The Back routine is where we displayed the LED lights on the transmitter board to show a certain pattern and then we had to transmit the robot ID and then transmit the action code which is the part that the other board will check and verify the code and then perform the certain action

Left Routine

The Left routine is where we displayed the LED lights on the transmitter board to show a certain pattern and then we had to transmit the robot ID and then transmit the action code which is the part that the other board will check and verify the code and then perform the certain action

right Routine

The right routine is where we displayed the LED lights on the transmitter board to show a certain pattern and then we had to transmit the robot ID and then transmit the action code which is the part that the other board will check and verify the code and then perform the certain action

Halt Routine

In the halt routine we needed send the action code to the robot for it to properly perform the action that was specified here we did not send in the id first but rather we loaded the freeze

freeze Routine

Here we did the same thing only this time we had to send the freeze signal

****

Receiver Routine

****

Initialization Routine

The initialization routine was mainly given from in the lab slides where we were given the frame format which wanted the data frame for 8 bits and the stop bit for 2 bits and the parity bit should be disabled and the asynchronous operation should be clarified. We also need a double data rate and to receive data sampled at the falling edge and we needed the baud rate to be 2400 bits per second. We also need the BotID to NOT be $2A because that is what the TA will use to test so we set ours up with 1A. After going to Justin Goins office hours, we figured out that almost all of these are by default set but just to be sure we made sure to set everything as the lab's slides indicated they wanted. We had to read over the data sheet a few times to make sure we were writing everything correctly. One of the major points was that we needed to make sure we were writing in with the high byte first for the baud rate because initially we were writing the low byte and this would activate the buffer and then create an issue where we had been writing garbage values into our 16 bit spot.

Main Routine

The Main routine is just an infinite loop of the code calling itself over and over again to detect if anything was hit and if any action should be performed as a result of this.

Receive Routine

The receive routine is where the meat of the program happens so this is where we have the parts that get checked and the parts that will perform the actions being transmitted to the board. So, we start off by having the data and stuff be cleared and properly loaded with the correct values. Then we check the freeze and we check our counter to see if we are at 3 hits and if we are then we will effectively break but if we are not, we will resume like normal. We then go through a series of checks to see what action we will be performing so we check If the forward signal was transmitted and then we check to see if the back and the turn left or right or the freeze and we check and make sure we are properly saving the last routine that we perform and that way we can re-use it later.

Disable Routine

The Disable routine is where we created an infinite loop and we create blinking lights that will tell the TA that our board as been hit 3 times and is no longer in possible use for other features.

SendFreeze Routine

The send freeze routine is where we send another signal to other boards and we need to make sure not to get our own board caught in an infinite loop so we have to enable the receiver and the interrupts. We then used what we had from our transmitter file to then wait a little bit and then start sending signals.

HitRight Routine

The HitRight routine first moves the TekBot backwards for roughly 1 second by first sending the Move Backwards command to PORTB followed by a call to the Wait routine. Upon returning from the Wait routine, the Turn Left command is sent to PORTB to get the TekBot to turn left and then another call to the Wait routine to have the TekBot turn left for roughly another second. Finally, the HitRight Routine sends a Move Forward command to PORTB to get the TekBot moving forward and then returns from the routine.

HitLeft Routine

The HitLeft routine is identical to the HitRight routine, except that a Turn Right command is sent to PORTB instead. This then fills the requirement for the basic BumpBot behavior.

Wait Routine

The Wait routine requires a single argument provided in the *waitcnt* register. A triple-nested loop will provide busy cycles as such that $16 + 159975 \cdot waitcnt$ cycles will be executed, or roughly $waitcnt \cdot 10\mathrm{ms}$. In order to use this routine, first the *waitcnt* register must be loaded with the number of 10ms intervals, i.e. for one second, the *waitcnt* must contain a value of 100. Then a call to the routine will perform the precision wait cycle.

****

Additional Questions

None for this lab☺

Difficulties

Some difficulties with this lab is that it was during the end of the term when we all have our own projects going on and we all need to focus at a lot of things at once. We also had to deal with multiple schedules and trying to meet at a good time for everyone. We also had a hard time visualizing the signals and what was going on because we are all CS majors and so we are used to being able to print out as we go to make sure we have the correct things in the correct place. Hardware is a little different where you would need to get an oscilloscope in order to see which bot is working and Justin said we could also use an old phone that would be able to transmit inferred light which would be able to see what was going on.

CONCLUSION

In conclusion this lab was extremely challenging, but it had to be in order to test our knowledge of the USART and how the receivers and transition would work. We were able to get our board communicating in the correct manner after first trying to just get a proper signal from one board to another. Then from there some of the things were a mix of other labs such as the bump bot activity and that motion that needed to be performed according to certain buttons being pressed. We were also able to make it play freeze tag with each other which was pretty cool and we were able to see just how much the AVR boards could handle and we could actually use a board to talk to another board through signals that the human eye cannot see which was pretty cool. Overall a very good lab that was hard at the same time.

SOURCE CODE

```
TX code: ;***********************************************************

;*

;*

;*title

;*

;*description

;*

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

;*

;*  Author: SONIA CAMACHO AND OWEN MARKLEY

;*    Date: 3/6/2020

;*

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

.include "m128def.inc"    ; Include definition file

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

;* Internal Register Definitions and Constants

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

.def mpr = r16    ; Multi-Purpose Register

.def data = r17    ; USART data

.def waitcnt = r23    ; Wait Loop Counter

.def ilcnt = r24    ; Inner Loop Counter

.def olcnt = r25    ; Outer Loop Counter

.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 Button0 = 1 ;setting up the buttons that we will use

.equ Button1 = 2

.equ Button3 = 4

.equ Button4 = 5

.equ Button5 = 6

.equ Button6 = 7

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

.equ robotID = $1A   ; Robot ID for this robot

; Use these action codes between the remote and robot

; 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

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

;* Start of Code Segment

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

.cseg      ; Beginning of code segment

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

;* Interrupt Vectors

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

.org $0000

  rjmp INIT

  ;this is the interrupt that will trigger the freeze signal

.org $0002

  rcall FreezeSignal

  reti
```

```
  ;this is the interrupt that will trigger the stop signal
.org $0004
  rcall Stop
  reti
  ;the usually included
.org $0046
;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
 ;Stack Pointer (VERY IMPORTANT!!!!)
  ldi  mpr, low(RAMEND)
  out  SPL, mpr  ; Load SPL with low byte of RAMEND
  ldi  mpr, high(RAMEND)
  out  SPH, mpr  ; Load SPH with high byte of RAMEND
 ;I/O Ports
  ;Initialize Port B for output
  ldi mpr, 0b11111111
  out DDRB, mpr ; Set the DDR register for Port B
  ldi mpr, $00
  out PORTB, mpr
  ; Initialize Port D for input
  ldi mpr, (1<<PD3)|(0<<Button0)|(0<<Button1)|(0<<Button3)|(0<<Button4)|(0<<Button5)|(0<<Button6)
  out DDRD, mpr ; Set the DDR register for Port D
  ldi mpr, (1<<Button0)|(1<<Button1)|(1<<Button3)|(1<<Button4)|(1<<Button5)|(1<<Button6)
  out PORTD, mpr
  ; Initialize external interrupts
  ; Set the Interrupt Sense Control to falling edge
  ldi mpr, (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10)
  ;EIMSK = 00001010
  sts EICRA, mpr ; Use sts, EICRA in extended I/O space
  ; Set the External Interrupt Mask
```

```
 ldi mpr, (1<<INT0)|(1<<INT1)
 out EIMSK, mpr ; Turn on interrupts
;USART1
 ;Initalize USART1
 ldi mpr, (1<<U2X1) ; Set double data rate
 sts UCSR1A, mpr
 ;Set baudrate at 2400bps
 ldi mpr, high(832) ; Load high byte of baudrate
 sts UBRR1H, mpr ; UBRR01 in extended I/O space
 ldi mpr, low(832) ; Load low byte of baudrate
 sts UBRR1L, mpr

 ; Set 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 transmitter
 ldi mpr, (1<<TXEN1)
 sts UCSR1B, mpr
 sei ; Enable global interrupt
;***********************************************************
;* Main Program
;***********************************************************
MAIN:
;checking to see if our forward button has been hit and if so call the function
sbis PIND, 7
rjmp Forward ;calling the forward function
;checking to see if the backwards button has been hit and if so call the function
sbis PIND, 6
rjmp Back ;calling the back function
;checking to see if the left button has been hit and if so call the function
sbis PIND, 5
```

```
 rjmp Left ;calling the left function

 ;checking to see if the right button has been hit and if so call the function

 sbis PIND, 4

 rjmp Right ;calling the function

  rjmp MAIN ;loop back into main

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

;

;* Functions and Subroutines

;

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

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

;* FORWARD

;* Forward will transmit the forward command

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

Forward:

  ldi mpr, 0b10011001 ;display this pattern onto the LED of the transmitter

  out PORTB, mpr

;Transmit robotid

fwd1:

  lds mpr, UCSR1A ; Loop until UDR1 is empty

  sbrs mpr, UDRE1 ; skip if bit in register set

  rjmp fwd1 ;send back into the loop

  ldi  data, robotID ;load the robot ID

  sts UDR1, data ; Move data to transmit data buffer

;Transmit move code

fwd2:

  lds mpr, UCSR1A ; Loop until UDR1 is empty

  sbrs mpr, UDRE1 ; skip if bit in register set

  rjmp fwd2  ;send back into the loop

  ldi   data, MovFwd ;load in the actual move forward action code to be transmitted which is
0b10110000

  sts UDR1, data ; Move data to transmit data buffer
```

```
rjmp MAIN ;big loop into main

;***********************************************************
;* BACK
;* Back will transmit the backwards command
;***********************************************************
Back:
  ldi mpr, 0b01100110 ;display this pattern to the LEDs on the transmitter board
  out PORTB, mpr
;Transmit robotid
bwd1:
  lds mpr, UCSR1A ; Loop until UDR1 is empty
  sbrs mpr, UDRE1 ; skip if bit in register set
  rjmp bwd1 ;send back into the loop
  ldi  data, robotID ;load the robot ID
  sts UDR1, data ; Move data to transmit data buffer
;Transmit move code
bwd2:
  lds mpr, UCSR1A ; Loop until UDR1 is empty
  sbrs mpr, UDRE1;skip if bit in register set
  rjmp bwd2 ;jump back into the loop
  ldi   data, MovBck ;here we are sending the different action code which will be for moving backwards
  sts UDR1, data ; Move data to transmit data buffer
rjmp MAIN ;go back to main
;***********************************************************
;* LEFT
;* LEFT will transmit the left moving command
;***********************************************************
Left:
  ldi mpr, 0b11000000 ;display this onto the LED
  out PORTB, mpr
```

```
;Transmit robotid
lft1:
  lds mpr, UCSR1A ; Loop until UDR1 is empty
  sbrs mpr, UDRE1 ; skip if bit in register set
  rjmp lft1 ;send back into the loop
  ldi  data, robotID ;load the robot ID
  sts UDR1, data ; Move data to transmit data buffer

;Transmit move code
lft2:
  lds mpr, UCSR1A ; Loop until UDR1 is empty
  sbrs mpr, UDRE1 ; skip if bit in register set
  rjmp lft2  ;send back into the loop
  ldi  data, TurnL ;sending in the left turning action code to the robot
  sts UDR1, data ; Move data to transmit data buffer
rjmp MAIN ;go back into main

;**********************************************************
;* RIGHT
;* RIGHT will transmit the right moving command
;**********************************************************
Right:
  ldi mpr, 0b00000011 ;opposite of the left turn we will display this on the LED's
  out PORTB, mpr
;Transmit robotid
rht1:
 lds mpr, UCSR1A ; Loop until UDR1 is empty
 sbrs mpr, UDRE1 ; skip if bit in register set
 rjmp rht1 ;send back into the loop
 ldi  data, robotID ;load the robot ID
 sts UDR1, data ; Move data to transmit data buffer
;Transmit move code
```

```
rhT2:

  lds mpr, UCSR1A ; Loop until UDR1 is empty

  sbrs mpr, UDRE1 ; skip if bit in register set

  rjmp rht2 ;send back into the loop

  ldi  data, TurnR ;load the right turning action code so that the robot will recieve the same
thing

  sts UDR1, data ; Move data to transmit data buffer

rjmp MAIN ;go into main




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

;* STOP

;* STOP will transmit the halt non moving command

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

Stop:

  ldi mpr, 0b00000000 ;display this onto the LED of transmitter board

  out PORTB, mpr

;Transmit robotid

hlt:

 lds mpr, UCSR1A ; Loop until UDR1 is empty

 sbrs mpr, UDRE1 ; skip if bit in register set

 rjmp hlt ;send back into the loop

 ldi  data, 0b01010101 ;load the freeze commanbd

 sts UDR1, data ; Move data to transmit data buffer

;Transmit move code

hlt2:

  lds mpr, UCSR1A ; Loop until UDR1 is empty

  sbrs mpr, UDRE1 ; skip if bit in register set

  rjmp hlt2 ;send back into the loop

  ldi  data, Halt ;load up the halt to be sent over to the robot

  sts UDR1, data ; Move data to transmit data buffer

ret
```

```
;*********************************************************
;* FreezeSignal
;* this will transmit the freeze signal
;*********************************************************
FreezeSignal:
  ldi mpr, 0b01010101 ;display on LED
  out PORTB, mpr
;Transmit robotid
frze:
  lds mpr, UCSR1A ; Loop until UDR1 is empty
 sbrs mpr, UDRE1 ; skip if bit in register set
 rjmp frze ;send back into the loop
 ldi  data, robotID ;load the robot ID
 sts UDR1, data ; Move data to transmit data buffer
;Transmit move code
frze2:
  lds mpr, UCSR1A ; Loop until UDR1 is empty
  sbrs mpr, UDRE1 ; skip if bit in register set
  rjmp frze2 ;send back into the loop
  ldi  data, Freeze ;load in the actual command to freezeeeee
  sts UDR1, data ; Move data to transmit data buffer
ret
;tis the end
Reciver code:
;*********************************************************
;*
;*  Lab 8 Robot (Receiver)
;*
;*  Program receives instruction from the remote and instructs
;*  the robot to act according to received instructions
;*
```

```
;*
;***********************************************************
;*
;*   Author: Owen Markley, Alex Molotkov, Sonia Camacho
;*     Date: 3/1/19
;*
;***********************************************************

.include "m128def.inc"      ; Include definition file

;***********************************************************
;*  Internal Register Definitions and Constants
;***********************************************************
.def  mpr = r16       ; Multi-Purpose Register
.def  data = r17
.def  lastTrans = r18
.def  lastDir = r22
.def  freezeCount = r23

.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 WTime = 100      ; Time to wait in wait loop ; set to 500 to make 5 s
.def waitcnt = r19      ; Wait Loop Counter
.def ilcnt = r20      ; Inner Loop Counter
.def olcnt = r21      ; Outer Loop Counter

.equ  BotAddress = $1A;(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

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

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

.org  $0002         ;- Left whisker
    rcall HitLeft
    reti

.org  $0004         ;- Right whisker
    rcall HitRight
    reti

.org  $003C         ;- USART1 receive
    rjmp Receive
```

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

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

  ;I/O Ports B out D in
  ldi mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL)
  out DDRB, mpr
  ldi mpr, (0<<EngEnL)|(0<<EngEnR)|(0<<EngDirR)|(0<<EngDirL)
  out PORTB, mpr

  ldi mpr, (0<<PD2)|(0<<WskrL)|(0<<WskrR)|(1<<PD3)
  out DDRD, mpr
  ldi mpr, (1<<WskrL)|(1<<WskrR)
  out PORTD, mpr

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

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

```
  ldi  mpr,  (1<<TXEN1)|(1<<RXEN1)|(1<<RXCIE1)|(0<<UCSZ12)  ;Enable  receiver  and  enable  receive
interrupts

  sts UCSR1B, mpr

  ldi mpr, (0<<UMSEL1)|(0<<UPM11)|(0<<UPM10)|(1<<USBS1)|(1<<UCSZ10)|(1<<UCSZ11) ;Set frame format:
8 data bits, 2 stop bits

  sts UCSR1C, mpr


  ldi lastTrans, $00

  ldi lastDir, $00;0b01100000

  ldi freezeCount, $00


  sei


  ;External Interrupts

  ldi mpr, $03 ;Set the External Interrupt Mask

  out EIMSK, mpr

  ldi mpr, (1<<ISC11)|(0<<ISC10)|(1<<ISC01)|(0<<ISC00) ;Set the Interrupt Sense Control to falling
edge detection

  sts EICRA, mpr


  ;Other


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

;*  Main Program

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

MAIN:

  ;TODO: ???


  rjmp  MAIN



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

;*  Functions and Subroutines
```

```
;*********************************************************
;-----------------------------------------------------------
;Receive
;-----------------------------------------------------------
Receive:
        clr mpr
        clr data
        lds data, UDR1

checkRecFrZ:  ldi mpr, 0b01010101
        cp data, mpr
        brne checkHandshake
        inc freezeCount
        cpi freezeCount, 3
        breq dis
        ldi mpr, Halt
        out PORTB, mpr
        ;ldi lastDir, Halt
        ldi  waitcnt, WTime
        rcall Wait
        rcall Wait
        rcall Wait
        out PORTB, lastDir
        rjmp end

dis:      rcall Disable



checkHandshake: ldi mpr, BotAddress
        cp mpr, lastTrans
        brne end
```

```
checkMovFwd:  ldi mpr, 0b10110000

        cp data, mpr

        brne checkMovBck

        ldi mpr, MovFwd

        out PORTB, mpr

        ldi lastDir, MovFwd

        rjmp end


checkMovBck:  ldi mpr, 0b10000000

        cp data, mpr

        brne checkTurnR

        ldi mpr, MovBck

        out PORTB, mpr

        ldi lastDir, MovBck

        rjmp end


checkTurnR:   ldi mpr, 0b10100000

        cp data, mpr

        brne checkTurnL

        ldi mpr, TurnR

        out PORTB, mpr

        ldi lastDir, TurnR

        rjmp end


checkTurnL:   ldi mpr, 0b10010000

        cp data, mpr

        brne checkHalt

        ldi mpr, TurnL

        out PORTB, mpr

        ldi lastDir, TurnL

        rjmp end
```

```
checkHalt:    ldi mpr, 0b11001000

        cp data, mpr

        brne checkSendFrZ

        ldi mpr, Halt

        out PORTB, mpr

        ldi lastDir, Halt

        rjmp end


checkSendFrZ: ldi mpr, 0b11111000

        cp data, mpr

        brne end

        rcall SendFreeze




end:      mov lastTrans, data

        reti

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

;Wait

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

Wait:

    push  waitcnt     ; Save wait register

    push  ilcnt     ; Save ilcnt register

    push  olcnt     ; Save olcnt register


Loop: ldi   olcnt, 224    ; load olcnt register

OLoop:  ldi   ilcnt, 237    ; load ilcnt register

ILoop:  dec   ilcnt     ; decrement ilcnt

    brne  ILoop     ; Continue Inner Loop

    dec   olcnt   ; decrement olcnt

    brne  OLoop     ; Continue Outer Loop

    dec   waitcnt   ; Decrement wait
```

```
    brne  Loop      ; Continue Wait loop

    pop   olcnt   ; Restore olcnt register
    pop   ilcnt   ; Restore ilcnt register
    pop   waitcnt   ; Restore wait register
    ret       ; Return from subroutine
;-------------------------------------------------------------
;HitLeft
;-------------------------------------------------------------
HitLeft:
      cli

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

      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

      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

      out PORTB, lastDir

      ldi mpr, 0b11111111
      out EIFR, mpr ;pushes to register
```

```
	rcall EmptyUSART

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

	sei
leftEnd:  reti

;------------------------------------------------------------
;HitRight
;------------------------------------------------------------
HitRight:
	cli

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

	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

	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

	out PORTB, lastDir
```

```
	ldi mpr, 0b11111111
	out EIFR, mpr ;pushes to register

	rcall EmptyUSART

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

	sei
rightEnd: reti

;-------------------------------------------------------------
;EmptyUSART
;-------------------------------------------------------------
EmptyUSART:
	lds mpr, UCSR1A
	sbrs mpr, RXC1
	ret
	lds mpr, UDR1
	rjmp EmptyUSART

;-------------------------------------------------------------
;Disable
;-------------------------------------------------------------
Disable:

infLoop:
	ldi mpr, 0b11110000
	out PORTB, mpr
```

```
      ldi  waitcnt, WTime
      rcall Wait
      ldi mpr, 0b00000000
      out PORTB, mpr
      ldi  waitcnt, WTime
      rcall Wait
      rjmp infLoop

      reti

;------------------------------------------------------------
;SendFreeze:
;------------------------------------------------------------

SendFreeze:
        ldi mpr, (1<<TXEN1)|(0<<RXEN1)|(1<<RXCIE1)|(0<<UCSZ12) ;Enable receiver and enable receive interrupts
        sts UCSR1B, mpr

transmitting:
        lds mpr, UCSR1A
        sbrs mpr, UDRE1
        rjmp transmitting
        ldi mpr, 0b01010101
        sts UDR1, mpr

        ldi waitcnt, 10
        rcall Wait

        ldi mpr, (1<<TXEN1)|(1<<RXEN1)|(1<<RXCIE1)|(0<<UCSZ12) ;Enable receiver and enable receive interrupts
        sts UCSR1B, mpr
```

```
        ret


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


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