ECE 375 Lab 8

Introduction to AVR Development Tools

**Lab Time: Tuesday 8-10**

Alexander Wood

Zack DeVita

# Introduction

Lab 8 required us to program two ATmega 128 micro controllers in order to create a remote controlled tekbot. The goal of this lab is for the students to use all the different methods taught in the class to combine them and create a working project. The first micro controller is programmed as the remote. This remote will poll the port d buttons to see if one of them has been pushed, if they have the remote will first send the robots address code. If this address code is validated, then the robot will receive the second byte of data which will contain the command the bot should execute. The bot is programmed to utilize interrupts to receive commands. When the RX complete interrupt is triggered, the bot will jump into a subroutine which will check the robot code address and if it matches the robots address, the next packet received will contain the desired command. This command will be interpreted by the bot to figure out what action the bot needs to take. In addition to this behavior, the bot was also to have limited “intelligence”. This was achieved by using the bumpers on the tekbot. When an interrupt is triggered on the tekbot by a bumper hit, the tekbot will reject incoming commands from the remote while it executes the behavior of reversing away from the object hit for 1 second, followed by 1 second of turning away from the bumper that was hit. The final step in lab 8 was to add the freeze tag functionality to the tekbot and controller. This functionality requires us to use a single button on the remote to send a freeze command to the robot. When the robot receives the freeze command it is to send out the freeze command which is to freeze any robot near enough to receive the signal. When our bot receives the freeze command it is to freeze for 5 seconds, and not respond to any command from the remote or bumpers. If the bot has been frozen 3 times, it is to lock in the halt position, and not respond to any command until it is reset. This completes the overview of the standard lab.

**Challenge:**

In the challenge we ae asked to add functionality to the remote control tekbot. The additional functionality is to implement our speed control functions from lab 7 into the controller and robot to allow for 15 levels of speed control. The second part of the challenge is to implement the bump bot behavior using a 16-bit counter that triggers behavior via interrupts.

# Program Overview

**Remote:**

The program for the remote starts with the standard initialization routine. Since the controller uses polling for the button pushes, there is no need to setup any interrupt vectors. Port d is configured for inputs, and the USART1 device is set up to transmit data based on the data from the polled buttons. When a button press is detected, the program jumps to the proper control subroutine, which stores the command code into a register. Once the value is in the register the subroutine calls the USART\_Tx function. This function is designed to send the 1-byte robot address. Once the TX is complete the function will then send the command that was set in the previous step.

**Robot:**

The program for the robot starts with the creation of the three needed interrupt vectors. The 3 needed vectors are the first two external interrupts to control the tekbots bumper behavior. The tekbot also needs to know when it is sent a signal. For this we used the USART RX interrupt. When there is a signal received by the tekbot, it will jump to the RXDATA function. The RXDATA function checks if the data is either the freeze command or a command from the remote. If the data RX has the bot address, the RX function will wait for the expected following command from the remote. The initialization routine is standard and configures the stack pointer, both ports B and D for their desired functionality, and the USART and timers.

**Challenge:**

The program written for the challenge portion of the lab has the same basic functionality as the robot program, but with a few key modifications. The robot code for the challenge still has external interrupts for the bumpers, and still receives USART commands via the remote. The primary difference is that the freeze functionality has been removed, and that a fast PWM counter has been utilized to show a variance in speed. With this implementation there are 15 different speeds that the robot can move in the forward direction, as well as, a halt, and a reverse. The LED lights controlled by the OCR0 and the OCR2 bits start out bright and become dimmer with each increase in speed until they are turned off. As the robot slows down the lights become brighter. When the robot finally comes to a halt, the lights controlled by the OCR0 and OCR2 bits will be fully bright, and the lights indicating forward movement will be off. When the robot is halting and the decelerate button is pressed subsequently, the robot goes into reverse. This is indicated by the lights controlled by the OCR0 and OCR2 bits being turned off, as well as, the lights indicating forward movement being turned off.

## Initialization Routine

**Remote:**

The Remote initialization is as follows. Firs the stack pointer is setup, as it is required for handling ISR’s and subroutine calls. Next the X pointer is configured with the memory address of $0100 which will allow the program to save states to memory if need be. The initialization then moves to set up PORT D as all inputs with the exception of pin 3. This is because pin 3 of port d is tied to the USART infrared transmitter, this pin is set as an output for the ability to send signals via the infrared device, all of PORT D is set as pull up resistors. Next the initialization routine configures the USART1 device to transmit. This initialization includes setting the baud rate to 2400, using double data rate with 2 stop bits. This is set by first calculating the value for UBRR1 which will control the baud rate. The equation used to solve for this is . This value is rounded down to 832 since float values shouldn’t be used. This value is loaded high byte first, then low byte. Next the USART1 is set to use double data rate by writing a 1 to the U2X1 bit in the UCSR1A register. Next the UCSR1B is set up to enable transmit by writing a 1 to the TXEN1 bit. Finally, the USART’s UCSR1C register is modified to set asynchronous mode, to set two stop its and to set up the frame size as 8-bits. This is done by shifting a 0 to UMSEL1 (sets the mode to asynchronous), and ones to USBS1 (2 stop bits) and ones into both UCSZ11 and UCSZ10 (Setting two of the 3 bits needed for 8-bit format, the other register is already set to 0 by default in UCSR1B)

**Robot:**

The robot initialization is pretty much the same as the remote initialization routine. The only changes to the initialization for the robot is the setting up of 3 interrupt vectors to handle the hit right, hit left and receive data interrupts. The robot requires that PORTD be set as inputs for the two bumpers and the IR receiver on pin 2 of PORTD. The Tekbot also needs to be able to send the freeze signal if the freeze command is sent, therefore port d pin 3 is set up to send data. The robot also requires the ability to write data out to the tekbots motor controllers. For this PORTB is setup as all outputs. The only difference for the USART setup is that the robot is configured so it can both send and receive data, requiring the TXEN1 to be set to 1. The external interrupts mask is set up to allow interrupts from external interrupts 0 and 1, which will interrupt in the event that the tekbot bumpers are hit. Finally, the 16-bit timer is setup to use a prescale of 256, which allows us to a 1 second delay to be executed by the timer. The starting state of the robot is set to halt. This information is written to PORT B to set the behavior, it is also written to the memory location pointed to by X. This allows for the previous state of the robot to be saved in the event a command is received, that requires the bot to return to its original state after the action takes place. The final two differences between the remote initialization and the robot initialization is the freeze counter register needs to be set to 0 to ensure a good value, and the wait 1 function is called to initialize the time. (this resolved some strange behavior with the way my timer works.)

**Challenge:**

The only thing that differs for the initialization of the portion of the code for the challenge and the initialization for the main part of the lab is that the challenge utilizes both 8-bit timers, so TCCR0 and TCCR2 must be set up. The initial speed for the robot is also set to zero which is the equivalent of the halt position. Both timers are set to fast PWM mode with a toggle, and are both set to inverting mode as well.

## Main Routine

**Remote:**

The main routine in the remote is very different from any previous implementation in the lab. The duty of the main function on the remote is to poll the 6 buttons. The polling is done in steps, first the value of PIND is written to mpr. Now mpr has the value of the PIND which should contain any button pushes by the user. The value in MPR is andi with the value of a button push. The values are then compared, and if the button push is detected, the function jumps branches to the routine to store the appropriate command value and transmit function, then it is returned to main to start polling again. If the button being tested is not pressed the function does not branch, but jumps into the next button push poll to check its value. When a button press is detected, the polling function will jump to the appropriate command subroutine. This subroutine starts by loading the appropriate hex value into the cmd\_tx(r17) register. Next the subroutine makes a call to the Tx command which send the robo address and the command from the button push. This will be discussed in the TX subroutine description.

**Robot:**

The robot’s main is a simple wait loop. This endless loop runs until an interrupt is received, at which point the appropriate ISR is initiated.

**Challenge:**

The same functionality described above is used for the challenge as well.

## USART\_Tx Routine

**Remote:**

The USART\_Tx command is what is used to send data once a button push has been detected by the polling loop. When this function is entered the first thing it does is copies the value of the robot’s code into register 18, so it is staged for transmit when the transmitter is ready. The UCSR1A register is polled if the Tx complete flag is set, the robot code is moved into UDRE for transmission. Once the robocode is sent the subroutine jumps to the DATA subroutine which repeats the same polling process to see if the Tx buffer is empty. As soon as the Tx buffer is empty the command data is loaded and transmitted.

**Challenge:**

The bot for the challenge does not transmit signals so there is no USART\_Tx Routine for it.

## RXDATA and RXDATA2 Routine

**Robot:**

This function is initiated by the ISR associated with the RX complete interrupt. This signals that the bot has received data. The first thing the subroutine does is grabs the received data from the RX buffer and first compares it to the freeze command. This is done because the freeze command from other bots will be received without a robot address, so the bot needs to be able to handle this capability. If the received command matches the freeze command, the function branches to the FREEZ\_BOT routine which will be discussed in the FREEZ\_BOT routine description. If the command received is not the freeze command, the RX function checks that the data received matches the robot code. If the code does not match, the skip function is called which disregards the data and returns the subroutine from the interrupt. If the robot code does match the expected code value, the function will jump to the RXDATA2 sub routine. This subroutine is responsible for receiving the command instruction. Once received it is then sent through a list of commands that checks if the value sent matches one of the expected commands. When a matching command value is found the current command will branch to the appropriate subroutine. This subroutine will move the appropriate command value into MPR, then it will jump to the WRT\_PORTB subroutine. This subroutine stores the command being written in the memory location pointed to by X. This is to deal with the hit right, hit left and freeze subroutines, as they will need access to the last command written in order to return to previous behavior. Next the subroutine will write the desired command to port b changing the state of the motor controller on the tekbot to the desired functionality and then the subroutine returns from the interrupt.

**Challenge:**

The bot code for the RXDATA and RXDATA2 routines for the challenge are identical to the routines used for the bot for the main part of the lab.

## FREEZ\_BOT Routine

**Robot:**

The freeze subroutine is entered by receiving the freeze command via the interrupt and the subsequent call to RXDATA. First the current value of portb is written to the memory location pointed to by X. This is to be able to restore the state if need be after the freeze is over. Next the halt value is written to the motor controller connected to PORTB. This stops the tekbot, then there are 5 calls to the wait one function, which causes the tekbot to stay frozen for 5 seconds. After the 5 second freeze, the frzCnt register is incremented. Now the frzCnt value is compared to 3. If frzCnt is equal to 3, the bot needs to freeze permanently until it is reset, and this is accomplished by jumping to the subroutine called DISABLE. This function turns off all interrupts to prevent the bot from accepting any other commands, writes the halt command to PORTB, then continuously loops on the disable command, which keeps the bot disabled.

**Challenge:**

This functionality is not utilized for the challenge portion of the lab.

## WAIT\_1 Routine

**Robot:**

The WAIT\_1 subroutine uses 16-bit timer 1 to count a 1 second delay. As soon as the function is entered, we push the X address, mpr, r18, and r30 to the stack to preserve any values, and allow the timer to use these registers to do timer operations. This 16-bit timer routine comes from Dr. Lee’s AVR book. First the 3036 is loaded into TCNT1. This sets the value the timer needs to start counting from in order to get a 1 second delay. The function then goes into the loop to poll for the TOV flag it be set which will indicate a 1 second delay. After the 1 second delay, we reset the TIFR by writing a 1 to it. And we reset the TCCR1B to the value of 0. This resolved some glitches we were having with the timer. After the wait has been executed, all the values that were stored on the stack are returned to their original locations, and the subroutine returns to where it was called from.

**Challenge:**

A WAIT\_1 function is utilized in the challenge, but it holds the same functionality as the code from the main part of the lab. The load value which is input into the TCNT1 registers has been modified to decrease the time for the wait. The original value used for a one second timer seems to pause for an unrealistic period of time so the value was decreased.

## HitRight Routine

**Robot:**

The hit right subroutine first disables interrupt to prevent them from stacking up. Next it writes the backup command to port b to get the bot to back up. This is followed by the wait 1 function that allows the bot to back up for 1 second. Now the turn left command is written to PORTB, and the wait 1 function is again called to allow the tekbot to turn for 1 second. Now that the behavior for the bumper hit is complete, the original value of the tekbot motion prior to the bumper hit is read from memory and written back to PORTB to return it to its initial state.

**Challenge:**

This functionality was kept in the challenge version of the code, and it is identical to the code used for the bot in the main portion of the lab. It is unnecessary for the purposes of the challenge, however can still be used.

## HitLeft Routine

**Robot:**

The hit left routine is identical to the hit right, except it turns right instead of left.

**Challenge:**

This functionality was kept in the challenge version of the code, and it is identical to the code used for the bot in the main portion of the lab. It is unnecessary for the purposes of the challenge, however can still be used.

# Conclusion

The requirements for this lab were essentially broken into two parts. For the first part of this lab we wrote an AVR assembly program which could output instructions via an infrared sensor and implemented it to an AtMega128 microcontroller. For the second part of this lab we wrote another program which could receive these transmitted instructions and act appropriately, and this was implemented on an additional AtMega128 microcontroller. This lab is essentially the culmination of all the material learned over the duration of the term. It required the use of communication through the onboard USART serial device, as well as, utilizing three of the four timers, polling for data, and external interrupts. This lab was an overall success; we were able to complete the lab. It was challenging to say the least. The lab proved to be an effective way to learn the material, and hopefully we can utilize the knowledge gained from this lab.

# Source Code

Provide a copy of the source code. Here you should use a mono-spaced font and can go down to 8-pt in order to make it fit. Sometimes the conversion from standard ASCII to a word document may mess up the formatting. Make sure to reformate the code so it looks nice and is readable.

**Remote:**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Enter Name of file here

;\*

;\* Enter the description of the program here

;\*

;\* This is the TRANSMIT skeleton file for Lab 8 of ECE 375

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Author: Alex Wood and Zack DeVita

;\* Date: Enter Date

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "m128def.inc" ; Include definition file

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Internal Register Definitions and Constants

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.def mpr = r16 ; Multi-Purpose Register

.def ilcnt = r19

.def olcnt = r20

;Adding some defs for constants

.def cmd\_tx = 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 = ($80|1<<(EngEnR-1)|1<<(EngEnL-1)) ;0b11001000 Halt Action Code

.equ robocode = $35

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* 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) ;load the address of the high byte of RAMEND

out SPH, mpr ;Sore it in the high byte of the stack pointer

ldi mpr, low(RAMEND) ;load the address of the low byte of RAMEND

out SPL, mpr ;Sore it in the low byte of the stack pointer

;set up X location in RAM

ldi XH, $01

ldi XL, $00

;I/O Ports

;PORT D setup for all inputs

;ldi mpr, 0b00001000 ;Set Port D Data Direction Register for inputs

ldi mpr, $08 ;Set Port D Data Direction Register for inputs

out DDRD, mpr ;Send DDR selection to DDRD

;ldi mpr, 0b11111111 ;PORTB inputs pullup outputs set hihg

ldi mpr, $FF

out PORTD, mpr ;Store value in PORTD register

;USART1

;Set baudrate at 2400bps

;Enable transmitter

;Set frame format: 8 data bits, 2 stop bits

;Calculated using double data rate, UBRR value needs to be 833 = $0340

ldi mpr, high($0340) ;load high value of UBRR into MPR

sts UBRR1H, mpr ;store high byte of 833 in high byte of UBRR

ldi mpr, low($0340) ;load low value of UBRR into MPR

sts UBRR1L, mpr ;store low byte of 833 in low byte of UBRR

;Enable transmitter and UDRIE for tx complete and double data rate

ldi mpr, (1<<U2X1)

sts UCSR1A, mpr

;Enable Tx and enable UDRIE to alow flag set

ldi mpr, (1<<TXEN1)

sts UCSR1B, mpr

;Set frame format: 8 data bits, 2 stop bits

ldi mpr, (0<<UMSEL1 | 1<<USBS1 | 1<<UCSZ11 | 1<< UCSZ10)

sts UCSR1C, mpr

;Other

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Main Program

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN:

;TODO: ???

in mpr, PIND ;Look for signal on PIND

andi mpr, (1<<PIND7) ;This is to mask out all other values to check just bit pos 7

cpi mpr, (1<<PIND7) ;check to see if PORTD pin 7 has been pulled low, if true zero flag set in sreg

brne BUTT8 ;Branch

jmp NEXT0

NEXT0:

in mpr, PIND ;Look for signal on PIND

andi mpr, (1<<PIND6) ;This is to mask out all other values to check just bit pos 7

cpi mpr, (1<<PIND6) ;check to see if PORTD pin 7 has been pulled low, if true zero flag set in sreg

brne BUTT7 ;Branch

jmp NEXT1

NEXT1:

in mpr, PIND ;Look for signal on PIND

andi mpr, (1<<PIND5) ;This is to mask out all other values to check just bit pos 7

cpi mpr, (1<<PIND5) ;check to see if PORTD pin 7 has been pulled low, if true zero flag set in sreg

brne BUTT6 ;Branch

jmp NEXT2

NEXT2:

in mpr, PIND ;Look for signal on PIND

andi mpr, (1<<PIND4) ;This is to mask out all other values to check just bit pos 7

cpi mpr, (1<<PIND4) ;check to see if PORTD pin 7 has been pulled low, if true zero flag set in sreg

brne BUTT5 ;Branch

jmp NEXT3

NEXT3:

in mpr, PIND ;Look for signal on PIND

andi mpr, (1<<PIND1) ;This is to mask out all other values to check just bit pos 7

cpi mpr, (1<<PIND1) ;check to see if PORTD pin 7 has been pulled low, if true zero flag set in sreg

brne BUTT2 ;Branch

jmp NEXT4

NEXT4:

in mpr, PIND ;Look for signal on PIND

andi mpr, (1<<PIND0) ;This is to mask out all other values to check just bit pos 7

cpi mpr, (1<<PIND0) ;check to see if PORTD pin 7 has been pulled low, if true zero flag set in sreg

brne BUTT1 ;Branch

jmp MAIN

;Move forward command, send bin val 10110000

BUTT8:

ldi cmd\_tx, $B0 ;Move forward command

rcall USART\_Tx ;Send Command

jmp MAIN

;Move forward backward, send bin val 10000000

BUTT7:

ldi cmd\_tx, $80 ;Move backward command

rcall USART\_Tx ;Send Command

jmp MAIN

;Turn left comand, send bin valu 10010000

BUTT6:

ldi cmd\_tx, $90 ;Turn left command

rcall USART\_Tx ;Send Command

jmp MAIN

;Turn right comand, send bin valu 10100000

BUTT5:

ldi cmd\_tx, $A0 ;Turn right command

rcall USART\_Tx ;Send Command

jmp MAIN

;Halt command, send bin value 11001000

BUTT2:

ldi cmd\_tx, $C8 ;Halt command

rcall USART\_Tx ;Send Command

jmp MAIN

;NOt used, reserved command 11111000

BUTT1:

ldi cmd\_tx, $F8 ;Unused comand

rcall USART\_Tx ;Send Command

jmp MAIN

; ldi mpr, $55

/\*BUTT1: ;Temp freeze button

;ldi cmd\_tx, $F8 ;Unused comand

rcall USART\_Tx2 ;Send Command

jmp MAIN

ldi mpr, $55\*/

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Functions and Subroutines

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

USART\_Tx:

ldi r18, robocode

lds mpr, UCSR1A ;UCSRA has the flag for the Tx buffer, this cant be accesed in the same wau it can when using usart1

sbrs mpr, UDRE1

rjmp USART\_Tx

sts UDR1, r18 ;Move data into Ready to send register

rjmp DATA

ret

DATA:

lds mpr, UCSR1A ;UCSRA has the flag for the Tx buffer, this cant be accesed in the same wau it can when using usart1

sbrs mpr, UDRE1

rjmp DATA

sts UDR1, cmd\_tx ;Move data into Ready to send register

rcall Wait

ret

/\*USART\_Tx2:

ldi r18, $55

lds mpr, UCSR1A ;UCSRA has the flag for the Tx buffer, this cant be accesed in the same wau it can when using usart1

sbrs mpr, UDRE1

rjmp USART\_Tx2

sts UDR1, r18 ;Move data into Ready to send register

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

ldi r18, 10

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 r18 ; 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

**Robot:**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Enter Name of file here

;\*

;\* Enter the description of the program here

;\*

;\* This is the RECEIVE skeleton file for Lab 8 of ECE 375

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Author: Alex Wood and Zack DeVita

;\* Date: Enter Date

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "m128def.inc" ; Include definition file

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Internal Register Definitions and Constants

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.def mpr = r16 ; Multi-Purpose Register

.def frzCnt = r20 ;Counts times frozen

.def cmd\_Tx = r21

.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 = $35;(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

;Should have Interrupt vectors for:

;- Right whisker

.org $0002

rcall HITRIGHT

reti

;- Left whisker

.org $0004

rcall HITLEFT

reti

;- USART receive

.org $003C ;This is the USART 1 RX interupt

rcall RXDATA

reti

.org $0046 ; End of Interrupt Vectors

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Program Initialization

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INIT:

;Stack Pointer (VERY IMPORTANT!!!!)

ldi mpr, high(RAMEND)

out SPH, mpr ;set up stack pointer high address

ldi mpr, low(RAMEND)

out SPL, mpr ;set up stack pointer low address

;I/O Ports

;PORTB setup

ldi mpr, $FF ;set pins as output

out DDRB, mpr ;store value

ldi mpr, $00 ;set 7:4 as pull down

out PORTB, mpr

;PORTD setup

ldi mpr, $00 ;set as input

out DDRD, mpr

ldi mpr, $03 ;set pull up resistor

out PORTD, mpr

;USART1

;Set baudrate at 2400bps

ldi mpr, high($0340)

sts UBRR1H, mpr

ldi mpr, low($0340)

sts UBRR1L, mpr

;Enable receiver and enable receive interrupts

ldi mpr, (1<<U2X1) ;Set Rx speed to double data rate

sts UCSR1A, mpr ;Store settings in USART Control Red A

ldi mpr, (1<<RXEN1 | 1<<RXCIE1)

sts UCSR1B, mpr

;Set frame format: 8 data bits, 2 stop bits

ldi mpr, (0<<UMSEL1 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)

sts UCSR1C, mpr

;External Interrupts

;Set the External Interrupt Mask

ldi mpr, $03

out EIMSK, mpr

;Set the Interrupt Sense Control to falling edge detection

ldi mpr, $AA

sts EICRA, mpr

;16-it timer from homerwork 3

ldi mpr, $04 ;set prescale to 256

out TCCR1B, mpr

;Set up X pointer to display recieved data

ldi XH, $01

ldi XL, $00

;Set initial command to halt

ldi mpr, Halt

st X, mpr

out PORTB, mpr

ldi frzCnt, 0 ; count for number of freeze commands received

rcall WAIT\_1

sei ; enable interrupts

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Main Program

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN:

rjmp MAIN ; wait for interrupt

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Functions and Subroutines

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RXDATA: ;Get address from buffer

lds mpr, UDR1 ;Move recieved data from buffer to mpr

cpi mpr, BotAddress ; if address is from our remote

breq RXDATA2 ; do this

cpi mpr, $55 ; if we receive a freeze from another bot

breq FREEZ\_BOT ; do this

rjmp SKIP

RXDATA2: ;get Data from buffer

lds r19, UCSR1A

andi r19, (1<<RXC1) ; checks the receive finish flag

cpi r19, 0b10000000

brne RXDATA2 ; loops until entire command is received

lds mpr, UDR1 ; loads command into mpr

;check if comand is move forward

cpi mpr, $B0

breq FWDCMD\_RX ;jump to appropriate cmd

;check if comand is move backward

cpi mpr, $80

breq BCKCMD\_RX ;jump to appropriate cmd

;check if comand is turn right

cpi mpr, $A0

breq TRNRGT\_RX ;jump to appropriate cmd

;check if comand is move forward

cpi mpr, $90

breq TRNLFT\_RX ;jump to appropriate cmd

;check if comand is move forward

cpi mpr, $C8

breq HALT\_RX ;jump to appropriate cmd

;check if Command is freeze

cpi mpr, $F8

breq EXCMD\_RX

;Write forward comand to MPR so it can be written to PORTB

FWDCMD\_RX:

ldi mpr, MovFwd

rjmp WRT\_PORTB

;Write Backward comand to MPR so it can be written to PORTB

BCKCMD\_RX:

ldi mpr, MovBck

rjmp WRT\_PORTB

;Write Turn Right comand to MPR so it can be written to PORTB

TRNRGT\_RX:

ldi mpr, TurnR

rjmp WRT\_PORTB

;Write Turn Left comand to MPR so it can be written to PORTB

TRNLFT\_RX:

ldi mpr, TurnL

rjmp WRT\_PORTB

;Write Halt comand to MPR so it can be written to PORTB

HALT\_RX:

ldi mpr, Halt

rjmp WRT\_PORTB

rjmp SKIP

; command for freeze signals from other bots

FREEZ\_BOT:

ldi mpr, Halt

out PORTB, mpr ; halts bot

rcall WAIT\_1 ; for 5 seconds

rcall WAIT\_1

rcall WAIT\_1

rcall WAIT\_1

rcall WAIT\_1

inc frzCnt ; increment frzCnt

cpi frzCnt, 3

breq DISABLE ; if received 3 commands then disable bot

ld mpr, X

out PORTB, mpr

reti ;return from function call

; Disable bot forever

DISABLE:

cli

ori mpr,$0F

out PORTB, mpr

rjmp DISABLE

; command for to have bot send out freeze command

EXCMD\_RX:

ldi mpr, (1<<TXEN1) ; disable receive and enable transmit

sts UCSR1B, mpr

in r23, PINB ; store initial value of PINB

ldi cmd\_tx, $55

sts UDR1, cmd\_tx ; transmit freeze command

rcall WAIT\_1 ; wait 1 second for transmit to complete

ldi mpr, (1<<RXEN1 | 1<<RXCIE1) ; re-enable receive on bot

sts UCSR1B, mpr

rcall WAIT\_1

out PORTB, r23 ; write bot instruction back to PORTB

rjmp SKIP ;jump to skip to leave bit in current state

;Write command to output

WRT\_PORTB:

st X, mpr

out PORTB, mpr ;Send decoded comand to motors

;reti ;return from function call

SKIP:

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* End RX Function

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HITRIGHT:

;move backward for 1 second

ldi mpr, MovBck

out PORTB, mpr

rcall WAIT\_1 ;call wait 1 function

;Turn left for 1 second

ldi mpr, TurnL

out PORTB, mpr

rcall WAIT\_1 ;call wait 1 function

;return robot to original state

ld mpr, X

out PORTB, mpr

ret ; Return to interupt call to return from interupt call

HITLEFT:

;move backward for 1 second

ldi mpr, MovBck

out PORTB, mpr

rcall WAIT\_1 ;call wait 1 function

;Turn left for 1 second

ldi mpr, TurnR

out PORTB, mpr

rcall WAIT\_1 ;call wait 1 function

;return robot to original state

ld mpr, X

out PORTB, mpr

ret ; Return to interupt call to return from interupt call

WAIT\_1:

push XH

push XL

push mpr

ldi mpr, high(3036) ;load value for delay

out TCNT1H, mpr

ldi mpr, low(3036)

out TCNT1L, mpr

ldi mpr, $04

out TCCR1B, mpr

LOOP:

in mpr, TIFR ;read in TOV1

andi mpr, $04 ;Check if TOV1 is set

breq LOOP ;Loop if TOV1 not set

ldi mpr, $04 ;Reset TOV1

out TIFR, mpr ;Write 1 to reset

ldi mpr, $00

out TCCR1B, mpr

pop mpr

pop XL

pop XH

ret

**Challenge:**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Enter Name of file here

;\*

;\* Enter the description of the program here

;\*

;\* This is the RECEIVE skeleton file for Lab 8 of ECE 375

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Author: Alex Wood and Zack DeVita

;\* Date: Enter Date

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "m128def.inc" ; Include definition file

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Internal Register Definitions and Constants

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.def mpr = r16 ; Multi-Purpose Register

.def frzCnt = r20 ;Counts times frozen

.def Speed = r21

.def incr = r22

.def Input = 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 BotAddress = $35;(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 = (1<<EngEnR|1<<EngEnL) ;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 = $00 ;0b10010000 Halt Action Code

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Start of Code Segment

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.cseg ; Beginning of code segment

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Interrupt Vectors

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.org $0000 ; Beginning of IVs

rjmp INIT ; Reset interrupt

;Should have Interrupt vectors for:

;- Right whisker

.org $0002

rcall HITRIGHT

reti

;- Left whisker

.org $0004

rcall HITLEFT

reti

;- USART receive

.org $003C ;This is the USART 1 RX interupt

rcall RXDATA

reti

.org $0046 ; End of Interrupt Vectors

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Program Initialization

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INIT:

;Stack Pointer (VERY IMPORTANT!!!!)

ldi mpr, high(RAMEND)

out SPH, mpr ;set up stack pointer high address

ldi mpr, low(RAMEND)

out SPL, mpr ;set up stack pointer low address

;I/O Ports

;PORTB setup

ldi mpr, $FF ;set pins 7:4 as outputs and 3:0 diabled

out DDRB, mpr ;store value

ldi mpr, $00 ;set 7:4 as pull down

out PORTB, mpr

;PORTD setup

ldi mpr, $00 ;set as all inputs

out DDRD, mpr

ldi mpr, $FF ;set pull up resistors

out PORTD, mpr

;USART1

;Set baudrate at 2400bps

ldi mpr, high($0340)

sts UBRR1H, mpr

ldi mpr, low($0340)

sts UBRR1L, mpr

;Enable receiver and enable receive interrupts

ldi mpr, (1<<U2X1) ;Set Rx speed to double data rate

sts UCSR1A, mpr ;Store settings in USART Control Red A

ldi mpr, (1<<RXEN1 | 1<<RXCIE1)

sts UCSR1B, mpr

;Set frame format: 8 data bits, 2 stop bits

ldi mpr, (0<<UMSEL1 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)

sts UCSR1C, mpr

;External Interrupts

;Set the External Interrupt Mask

ldi mpr, $03

out EIMSK, mpr

;Set the Interrupt Sense Control to falling edge detection

ldi mpr, $AA

sts EICRA, mpr

;16-it timer from homerwork 3

ldi mpr, $04 ;set prescale to 256

out TCCR1B, mpr

; Configure 8-bit Timer/Counters

ldi mpr, 0b01111001 ; Activate Fast PWM mode with toggle

out TCCR0, mpr ; (inverting), and no prescalar

out TCCR2, mpr ; (inverting), and no prescalar

; Set initial speed, display on Port B pins 3:0

ldi Speed, $00 ; Set speed initially to zero

out OCR0, Speed

out OCR2, Speed

;Set up X pointer to display recieved data

ldi XH, $01

ldi XL, $00

ldi frzCnt, 0 ; count for number of freeze commands received

ldi incr, 17 ; set increment amount to 17

rcall WAIT\_1

sei ; enable interrupts

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Main Program

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN:

rjmp MAIN ; wait for interrupt

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Functions and Subroutines

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RXDATA: ;Get address from buffer

ldi mpr, $00 ; Temporarily disable interrupts

out EIMSK, mpr

lds mpr, UDR1 ;Move recieved data from buffer to mpr

cpi mpr, 0b01010101

breq FREEZ\_BOT

cpi mpr, BotAddress ;Check that it is from the proper remote

brne SKIP

RXDATA2: ;get Data from buffer

lds r19, UCSR1A

andi r19, (1<<RXC1)

cpi r19, 0b10000000

brne RXDATA2

lds mpr, UDR1

;check if comand is move forward

cpi mpr, $B0

breq FWDCMD\_RX ;jump to appropriate cmd

;check if comand is move backward

cpi mpr, $80

breq BCKCMD\_RX ;jump to appropriate cmd

;check if comand is turn right

cpi mpr, $A0

breq TRNRGT\_RX ;jump to appropriate cmd

;check if comand is move turn left

cpi mpr, $90

breq TRNLFT\_RX ;jump to appropriate cmd

;check if comand is halt

cpi mpr, $C8

breq HALT\_RX ;jump to appropriate cmd

;check if comand is move freeze

cpi mpr, $F8

breq SKIP ;jump to appropriate for this lab SKIP

rjmp SKIP ; in case none of the previous commands are

;Write forward comand to MPR so it can be written to PORTB

FWDCMD\_RX:

cpi Speed, $0F ; check if speed is at max

breq SKIP

in mpr, OCR0 ; put OCR value into mpr

add mpr, incr ; increment register by 17

out OCR0, mpr ; Set compare value

out OCR2, mpr ; Set compare value

inc Speed ; increment current speed

ldi mpr, (1<<EngDirR|1<<EngDirL);|1<<EngEnR|1<<EngEnL)

rjmp WRT\_PORTB

;Write Backward comand to MPR so it can be written to PORTB

BCKCMD\_RX:

cpi Speed, $01 ; check if speed is at min

breq HALT\_RX

cpi Speed, $01

brlt REVERSE

in mpr, OCR0 ; put OCR value into mpr

sub mpr, incr ; increment register by 17

out OCR0, mpr ; Set compare value

out OCR2, mpr ; Set compare value

dec Speed ; increment current speed

ldi mpr, (1<<EngDirR|1<<EngDirL);|1<<EngEnR|1<<EngEnL)

rjmp WRT\_PORTB

REVERSE:

ldi mpr, $FF

ldi Speed, $00 ; Set speed initially to zero

out OCR0, mpr

out OCR2, mpr

ldi mpr, $00

rjmp WRT\_PORTB

;Write Turn Right comand to MPR so it can be written to PORTB

TRNRGT\_RX:

ldi mpr, TurnR

rjmp WRT\_PORTB

;Write Turn Left comand to MPR so it can be written to PORTB

TRNLFT\_RX:

ldi mpr, TurnL

rjmp WRT\_PORTB

;Write Halt comand to MPR so it can be written to PORTB

HALT\_RX:

ldi Speed, $00

ldi mpr, $00

rjmp WRT\_PORTB

;Write command to output

WRT\_PORTB:

st X, mpr

out PORTB, mpr ;Send decoded comand to motors

rcall WAIT\_100

SKIP: ;will not RX command because robocode doesnt match

ldi mpr, $03 ; Enable interrupts

out EIMSK, mpr

reti ;return from function call

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* End RX Function

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HITRIGHT:

;move backward for 1 second

ldi mpr, MovBck

out PORTB, mpr

rcall WAIT\_1 ;call wait 1 function

;Turn left for 1 second

ldi mpr, TurnL

out PORTB, mpr

rcall WAIT\_1 ;call wait 1 function

;return robot to original state

ld mpr, X

out PORTB, mpr

ret ; Return to interupt call to return from interupt call

HITLEFT:

;move backward for 1 second

ldi mpr, MovBck

out PORTB, mpr

rcall WAIT\_1 ;call wait 1 function

;Turn left for 1 second

ldi mpr, TurnR

out PORTB, mpr

rcall WAIT\_1 ;call wait 1 function

;return robot to original state

ld mpr, X

out PORTB, mpr

ret ; Return to interupt call to return from interupt call

; function waits for 1 second

WAIT\_1:

push XH

push XL

push mpr

ldi mpr, high(1500) ;load value for delay

out TCNT1H, mpr

ldi mpr, low(1500)

out TCNT1L, mpr

ldi mpr, $04

out TCCR1B, mpr

LOOP:

in mpr, TIFR ;read in TOV1

andi mpr, $04 ;Check if TOV1 is set

breq LOOP ;Loop if TOV1 not set

ldi mpr, $04 ;Reset TOV1

out TIFR, mpr ;Write 1 to reset

ldi mpr, $00

out TCCR1B, mpr

pop mpr

pop XL

pop XH

ret