ECE 375 Lab 8

Remotely Operated Vehicle

**Lab Time: Thursday 1000-1200**

Eric Prather

# Prelab

*1. In this lab, you will be given a set of behaviors/actions that you need to have a proof-of-concept “toy” perform. Think of a toy you know of (or look around online for a toy) that is likely implemented using a microcontroller, and* ***describe the behaviors it performs****. Here is an example behavior: “If you press button X on the toy, it takes action Y (or makes sound Z)”.*

A cheap, simple electronic toy commonly distributed in the early 2000s was the [Tamagotchi](https://en.wikipedia.org/wiki/Tamagotchi). It operated on a battery power-supply and was about the size of a palm. It has three input modalities: Buttons on its front face. It has one output modality, its LCD screen. More complex iterations of the toy had more complex parts, including infrared devices, games, different input and output paradigms, and etcetera. A simplified Tamagotchi functions as follows:

1. If a minute passes without input, the screen turns off.
2. If Input is received while the screen is off, the screen turns on.
3. Every second, the LCD displays one of several predefined hardcoded binary textures based on:
   1. A looping counter
   2. The state of five internal variables
4. Pressing combinations of input buttons change one of the internal variables and overrides the LCD output with various predefined binary textures.
5. Four internal variables gradually increase over time at varying rates.
6. A fifth internal variable updates every several hours automatically. It cannot be interfaced with via the front-facing input buttons.

*2. For each behavior you described in the previous question, explain which microcontroller feature was likely used to implement that behavior, and give a brief code example indicating how that feature should be conﬁgured. Make your explanation as ATmega128-speciﬁc as possible (e.g., discuss which I/O registers would need to be conﬁgured, and if any interrupts will be used), and also mention if any additional mechanical and/or electronic devices are needed.*

1. The screen-sleep feature was likely inputted with a clock. If this were implemented on the ATMega128, it would look something like:  
   INIT:  
   ldi mpr 0b00000010; TCNT0 overflow “OCIE0”  
   out TIMSK, mpr; Timer Interrupt Mask Register  
   ldi mpr, 0b00000111 ; Prescaler bits (CS00,CS01,CS02 => clk\_T0S/111), normal bit  
    ; (WGM01:0=0) must be set. Clear COM01:0. Clear FOC.  
   out TCCR0, mpr ; write to the correct register: TCCR0. Does both prescaling and mode  
   INTERRUPT VECTOR CALL:  
   ldi mpr, 0x00 ; false  
   st SCREEN\_TOGGLE\_VALUE, mpr ; LCD driver on/off boolean
2. The screen-wakeup feature was likely done with buttons. If this were implemented on the ATMega128, it would look something like:  
   INIT:  
   ldi mpr, $00  
   out DDRD, mpr  
   ldi mpr, $FF  
   out PORTD, mpr  
   ldi mpr, ob00000111  
   sts EICRA, mpr  
   ldi mpr, (1<<INTF0) | (1<<INTF1) | (1<<INTF2)  
   out EIFR, mpr  
   ldi mpr, (1<<INT0) | (1<<INT1)  
   out EIMSK, mpr  
   INTERRUPT VECTOR CALLS:  
   ; Blah blah blah conditionals and branching based on input vector triggered  
   ldi mpr, 0x01 ; pretend this is true  
   st SCREEN\_TOGGLE\_VALUE, mpr ; LCD driver on/off boolean
3. The LCD animation feature was likely implemented with a clock, LCD driver, and set of hardcoded program memory values representing binary textures. Its usage of internal variables would involve data memory storage external to the space used by the LCD driver routines. If this were implemented on the ATMega128, it would look something like:  
   INIT:  
   ldi mpr 0b00000010; TCNT0 overflow “OCIE0”  
   out TIMSK, mpr; Timer Interrupt Mask Register  
   ldi mpr, 0b00000111 ; Prescaler bits (CS00,CS01,CS02 => clk\_T0S/111), normal bit  
    ; (WGM01:0=0) must be set. Clear COM01:0. Clear FOC.  
   out TCCR0, mpr ; write to the correct register: TCCR0. Does both prescaling and mode  
   INTERRUPT VECTOR CALL:  
   ldi YL, LOW(LCD\_DISPLAY\_AREA)  
   ldi YH, HIGH(LCD\_DISPLAY\_AREA)  
   ; Do conditional logic based on internal variables to know which address to get the binary texture from.   
   ldi ZL, LOW(TARGET\_TEXTURE)  
   ldi ZH, HIGH(TARGET\_TEXTURE)  
   ldi counter, 255  
   TEXTURE\_DISPLAY\_LOOP:  
   ld mpr, Z+   
   st Y+, mpr ; LCD driver on/off boolean  
   dec counter  
   cpi counter, 0x00  
   brne TEXTURE\_DISPLAY\_LOOP
4. The input feature which changes variables and outputs to the LCD likely used tri-state buffer input pins. If these input modalities were implemented on the ATMega128, they would look something like:  
   ldi mpr, $00  
   out DDRD, mpr  
   ldi mpr, $FF  
   out PORTD, mpr  
   ldi mpr, ob00000111  
   sts EICRA, mpr  
   ldi mpr, (1<<INTF0) | (1<<INTF1) | (1<<INTF2)  
   out EIFR, mpr  
   ldi mpr, (1<<INT0) | (1<<INT1)  
   out EIMSK, mpr  
   INTERRUPT VECTOR CALLS:  
   ; Blah blah blah conditionals and branching based on input vector triggered  
   ldi mpr, EQU\_SET\_PREDEFINED\_VALUE ; depends on variable to change, current value of variable, input set, etc.  
   st INTERNAL\_VARIABLE\_N, mpr ; Wherever the pertinent internal variable is stored in memory
5. The automatic incrementing of variables over various amounts of time were likely handled with a clock. If this were implemented on the ATMega128, it would look something like:  
   INIT:  
   ldi mpr 0b00000010; TCNT0 overflow “OCIE0”  
   out TIMSK, mpr; Timer Interrupt Mask Register  
   ldi mpr, 0b00000111 ; Prescaler bits (CS00,CS01,CS02 => clk\_T0S/111), normal bit  
    ; (WGM01:0=0) must be set. Clear COM01:0. Clear FOC.  
   out TCCR0, mpr ; write to the correct register: TCCR0. Does both prescaling and mode  
   INTERRUPT VECTOR CALL:  
   push mpr  
   in mpr, sreg  
   push mpr  
   push zl  
   push zh  
   ldi ZL, LOW(VARIABLE\_TO\_INCREMENT)  
   ldi ZH, HIGH(VARIABLE\_TO\_INCREMENT)  
   ld mpr, Z  
   inc mpr  
   st Z, mpr ; LCD driver on/off boolean  
   pop zh  
   pop zl  
   pop mpr  
   out sreg, mpr  
   pop mpr
6. Same as above

*3. Each ATmega128 USART module has two ﬂags used to indicate its current transmitter state: the Data Register Empty (UDRE) ﬂag and Transmit Complete (TXC) ﬂag. What is the diﬀerence between these two ﬂags, and which one always gets set ﬁrst as the transmitter runs? You will probably need to read about the Data Transmission process in the datasheet (including looking at any relevant USART diagrams) to answer this question.*

The data register empty flag (UDRE) is set when the USART Data Register (UDRn) register is ready to be written to. It triggers interrupt vector $003E (p60). The Transmit complete flag (TXC) is set when the final significant value in the Transmit Shift Register is shifted out. It triggers interrupt vector $0040 (p60).

Data moves from the UDRn to the transmit shift register. It is possible for the transmission frame to be empty, complete, or in process while there is new data waiting in UDRn. Pages 178-129 describe this relationship.

Notably, in standard USART procedures, UDRE is typically set before TXC. This is because that first the data is moved from UDRE to the shift register, clearing UDRn, and *then* TXC slowly shifts out the data inside of it. More information on these flags can be found on page 188 of the datasheet.

*4. Each ATmega128 USART module has one ﬂag used to indicate its current receiver state (not including the error ﬂags). For USART1 speciﬁcally, what is the name of this ﬂag, and what is the interrupt vector address for the interrupt associated with this ﬂag? This time, you will probably need to read about Data Reception in the datasheet to answer this question.*

**Note:** For identically behaving flags, for some reason, the “n” in the pin name provided in the datasheet must be substituted with the number of the register the flag is being applied to, despite the fact that all “n” registers have the exact same evaluation.

The answer to this prelab question can be pulled specifically from HW3. More precisely:

The **name** of the flag is “USART Receive Complete” (RXC1) – see page 188 of the datasheet. It is in USART Control and Status Register A (UCSR1A). The **interrupt vector** associated with this flag is $003C – see page 59 of the datasheet.

# Introduction

This lab is the cumulative demonstration of AVR development skills acquired over the course of ECE375. It requires a bottom-up knowledge of everything from memory management and I/O to interrupts and timers. This project will use Universal Synchronous/Asynchronous Reception and Transmission (USART) to allow two separate boards to communicate with each other. While in this lab the medium of transmission is infrared light, any medium would be acceptable with this hardware module so long as it preserves binary serialization data.

Taking advantage of this enabling technology, this lab synthesizes full-duplex serial communication. Establishing the same USART settings on both boards will be trivial, as they will both have their flash written to by the same programmers. USART is relatively simple because between devices, there only needs to be a link between the TX/RX pins.

## Part 1 Specifications

Overall, the data transmitted will be decomposed into five separate data values that can be taken by the remote-controlled board. Each data value will result in the receiving board conducting a different output action. Furthermore, a rudimentary addressing system should be built into the protocol so that up to 2^7 pairs of boards can be active on the same medium without multiple recipients responding to the same command (provided, of course, that they were configured correctly when their flash memory was written to).

The bump-bot behavior from lab 1 will also be replicated for posterity on the recipient board.

## Part 2 Specifications

A 6th data form will be added to the previous five with a separate implication for the recipient board’s behavior. This one is unique as it is to **briefly switch the recipient board to a transmission board** and send the unaddressed broadcast 0b01010101. Any recipient receiving this signal will in turn execute yet *another* separate behavior. In this case, this varies based on the number of times 0b01010101 has been received. Specifically:

1. Received 0, 1, or 2 times: Disable interrupts and the main cycle for 5 seconds (Likely using TCNT0 or an idle loop).
2. Received 3 or more times: Disable interrupts and relative-jump to an infinite loop.

# Program Overview

Two programs will be written: One for the remote control board and one for the recipient board. Notably, the recipient board will use several of the functions from Lab 1, and as such the pertinent documentation there-from is copied. Address $2A is forbidden.

## Macros

; Note: it is assumed that these macros are all used for USART0, *not* USART1.

ADDRESS = 0d69 ; nice

BAUD\_LOW = 0b (See later in document)

BAUD\_HIGH = 0b (See later in document)

; *Question: What is U2x = 0,1? This may change the desired baud rate to 832 instead of 416*

; There is no macro for TX\_A or RX\_A as they are read-only

TX\_B = Interrupt on transmission complete. Interrupt on data register empty. Enable transmitter.

(1 << TXCIE1) | (1 << TXEN1)

RX\_B = Interrupt on reception complete. Enable receiver. 8 bits of data.

(1 << RXCIE1) | (1 << RXEN1)

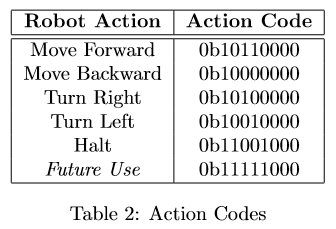
TX\_C = USART Mode Synchronous. Disable parity. Two stop bits. 8 bits of data. Falling edge.

(1 << UMSEL1) | (1 << USBS1) | (1 << USCZ11) | (1 << USCZ10) | (1 << UCPOL1)

RX\_C = USART Mode Synchronous. Disable Parity. Two stop bits. 8 bits of data. Falling edge

(1 << UMSEL1) | (1 << USBS1) | (1 << USCZ11) | (1 << USCZ10)

These are also macros:



## Both Boards

### USART Initialization

In this lab, we will use USART 1, not USART 0. All of USART 1 is in extended IO space, unlike USART0.

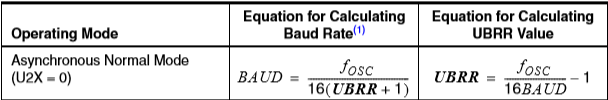
USART must be initialized with the following characteristics:

* Baud Rate = 2400 bits per second.
* Data Frame: 8-bit
* Stop bit: 2-bit
* Parity bit: disable

The Lab PowerPoint provides a reminder of all the USART control registers so that, unlike in homework 3, the datasheet does not need to be consulted. Because of the high volume of registers described, this is omitted from the lab report. Consult the lab slides for a specific register description.

A calculation for what values need to be written to the USART registers as well as some initialization pseudocode was provided in Homework 3. This can be observed as follows:

The data sheet explains all of the interrupt vectors that are pertinent on page 59. A detailed description of what USART- or Universal Synchronous and Asynchronous serial Receiver and Transmitter- can be found starting on page 170.

The calculation for the baud rate is as follows, from page 173 of the datasheet:  


Therefore, to attain the operations to be performed to attain BAUD = 2400, we go through the following procedure:

Similar math can be found on page 193 (16Mhz examples on 196) of the data sheet, with additional descriptions of error rate. The datasheet recommends a value of either 416 (the number we calculated) or 832 (0b\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_), but it was difficult to discern the difference.

To make Lab 8 run more smoothly, the specific flag sets for the transmission and reception of registers are defined with the .equ macro. For a list of macros, see the above section.

Here is my **pseudocode**, to be inserted after the label initUSART0, largely inspired by the snippet on page 176 of the datasheet:

; Eric Prather

; This code executes after interrupt vector $0026, USART0 Data Register Empty

; Configure USART0 (Port E, pin 1) as output or input

ldi mpr, 0b000000X0 ; Pin 1 of port E, where X is output or input

out DDRE, mpr ; this is the data direction register for port E, as requested.  
ldi mpr, $00

out PORTE, mpr ; Just to be safe

; Set baud rate to 2,400 (See above calculations)

ldi mpr BAUD\_LOW  
sts UBRR1L, mpr  
ldi mpr BAUD\_HIGH  
sts UBRR1H, mpr

The following code takes in specific flags depending on if it is the transmitter or the receiver.

; Enable interrupts and stuff

ldi r16 USART\_B\_FLAGS\_[TX/Rx] ; 0b00101000  
sts UCSR1B, r16

; Set asynchronous mode and frame format

ldi r16 USART\_C\_FLAGS\_[TX/Rx] ;(1<<UPM01) | (1<<UPM00) | (USCZ01) (USCZ00) ; 0b00110110

sts UCSR1C, r16

; End of code segment

## Transmitter Board

### Initialization Routine

The initialization of the transmitter is very simple. It takes port D as **interrupt** input, as has been done in prior labs, and runs the USART initialization routine with TX. It enables interrupts to help control the USART and interrupting input. Notably, **bounce is not an issue** in this lab because it is the responsibility of the receiver to handle repeated messages.

USART initialization is called, as described earlier in the document.

### Main Routine

Loops infinitely.

### PortD Input Handling ~~Interrupt N~~

All port D interrupts function the same, but use a different message macro. Their structure is:

1. Write an address message to the USART data register
2. Display the message to be sent to port B.
3. Disable port D interrupts
4. Set the data memory COMMAND space to be the command corresponding to the interrupt PIN

### USART Data Register Empty

1. Load the command at COMMAND
2. Write it to the USART data register
3. Disable USART Data Register Empty Interrupt

### Transmission Complete

1. Disable transmission complete interrupt
2. Enable PortD interrupt
3. Flush PortD interrupt flags

## Recipient Board

### Initialization Routine (PART 1)

The recipient board needs to be ready to handle a wide variety of situations. But the initialization is simple. Bump bot behavior is described in Initialization Routine Part 2. The specific part to this lab is:

1. Initialize USART for input (See USART routines)
2. Set state to be listening for own address

### USART Reception Complete

If this robot has received its address during the last USART Reception Complete command:

1. Perform conditional branching based on the 6 possible data messages that must currently be in the USART Data Register
   1. Most behaviors are one or two lines, and refer to Lab 1 behaviors detailed later.
2. Reset state to listening for own address
3. Return from interrupt

Otherwise:

1. Compare USART data register to own address
2. If own address, set state to be listening for instruction to execute.
3. Compare USART register to 0b01010101
4. If match, perform an idle loop for 5 seconds, then:
   1. If this is the third time this has happened, go back to the beginning of the loop
   2. Reset state to listening for own address
   3. Return from interrupt

### Transmission Complete

This will only happen after the **freeze** command. Because it branches away from USART reception complete, we need to replicate its behavior as well as switch back to USART reception mode. Notably:

1. Set USART to reception mode
2. Reset state to listening for own address

### Broadcast Freeze

This is only necessary to implement for part 2 of the lab.

1. Set USART to transmission mode
2. Sent 0b01010101

Then wait for the Transmission Complete interrupt

--

The following routines are the same as they were in lab 1:

### Initialization Routine (PART 2)

The initialization routine provides a one-time initialization of key registers that allow the BumpBot program to execute correctly. First the Stack Pointer is initialized, allowing the proper use of function and subroutine calls. Port B was initialized to all outputs and will be used to direct the motors. Port D was initialized to inputs and will receive the whisker input. Finally, the Move Forward command was sent to Port B to get the TekBot moving forward.

### Main Routine

The Main routine executes a simple polling loop that checks to see if a whisker was hit. This is accomplished by first reading 8-bits of data from PINE and masking the data for just the left and right whisker bits. This data is checked to see if the right whisker is hit and if so, then it calls the HitRight routine. The Main routine then checks to see if the left whisker is hit and if so, then it calls the HitLeft routine. Finally a jump command is called to move the program back to the beginning of the Main Routine to repeat the process.

### 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 cycles will be executed, or roughly . 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

There are no additional questions for Lab 8.

# Difficulties

I had more difficulties with this lab than any other lab in the term, save maybe lab 7. Notably:

* My lab partner seems to have dropped the lab section entirely
* I could not figure out exactly what my flag macros should have been without exorbitant trial and error. Notably, I assumed that we were supposed to be using synchronous mode when we were really using asynchronous mode.
* I had to write a very high volume of code, taking me a long time.
* I had to rely heavily on the debugger due to a lack of available boards.
* I had to switch my core PORTD-parsing strategy half-way through the lab from interrupts to parsing. This was following the recommendation by a TA to switch my strategy after they claimed that external interrupts were incompatible with USART
* I only had one board to test with.
* The simulator did not simulate USART and interrupts for this project.

# Conclusion

This lab served as an excellent capstone to the ECE375 course. It demonstrated a functionality of the ATMega128 board with such potential and operating with such specific parameters that, in implementing it, my assembly language programming skills were thoroughly assessed on several fronts. I worked extensively with IO in a variety of forms, even switching data direction indicators live during operation.

Conducting this lab also taught me a valuable lesson about the importance of frequent prototyping. Unlike previous labs, I could not rely on the simulator to write out and test all of my code before deploying it to the board for a hardware test. Because of this, I ended up deploying frequently and making heavy use of the boards supplied by the TAs in the lab. Funnily enough, because of the rapid feedback I got from the tests I ran and the peers who observed them, I think I may have completed Lab 8 faster than labs 7 and 6. I had expected this to be the longest lab of the term, because it was two weeks long, so this was a pleasant surprise.

# Source Code

Because the code for this project was incredibly long, I developed it with the function descriptions in this document rather than as comments. I hope this improves legibility.

Additionally, as there are two files in this lab, I include both separately here.

## Transmission

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

;\*

;\* Prather\_Eric\_Lab8\_tx\_sourcecode.asm

;\*

;\* Remote Controller

;\*

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

;\*

;\* Author: Eric Prather

;\* Date: February 27, 2020

;\*

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

.include "m128def.inc" ; Include definition file

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

;\* Shared Macros

;\*

;\* These macros are exactly the same in all files

;\* Uncomment desired BAUD number, comment undesired.

;\*

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

.equ ADDRESS = 69 ; Nice.

; Baud rate should be 2400

; Baud # (single): 416

;.equ BAUD\_LOW = 0b10100000

;.equ BAUD\_HIGH = 0b00000001

; Baud # (double): 832

.equ BAUD\_LOW = 0b01000000

.equ BAUD\_HIGH = 0b00000011

; Flag descriptions in lab report

.equ TX\_A = (1 << U2X1)

.equ RX\_A = (1 << U2X1)

.equ TX\_B = (1<<TXEN1) ; Interrupts: TXCIE1 and UDRE1

.equ RX\_B = (1<<RXEN1)

.equ TX\_C = (1<<USBS1) | (1<<UCSZ11) | (1<<UCSZ10) | (1<<UCPOL1) ;|(1<<UMSEL1)

.equ RX\_C = (1<<USBS1) | (1<<UCSZ11) | (1<<UCSZ10);| (1<<UCPOL1) ;|(1<<UMSEL1)

.equ UDRE\_INTERRUPT\_INVERSE\_MASK = 0b10111111

.equ TXC\_INTERRUPT\_INVERSE\_MASK = 0b11011111

.equ EXI\_MASK = 0b11110011 ; Push buttons

.equ MOVE\_FORWARD = 0b10110000

.equ MOVE\_BACKWARD = 0b10000000

.equ TURN\_RIGHT = 0b10100000

.equ TURN\_LEFT = 0b10010000

.equ HALTSIG = 0b11001000

.equ FREEZE\_CMD = 0b11111000

.equ FREEZE\_BROADCAST = 0b01010101

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

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

;\* Internal Register Definitions and Constants

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

.def mpr = r16 ; Multi-Purpose Register

.def mpr2 = r17 ; RESERVED MP-register

.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

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

;\* Start of Code Segment

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

.cseg ; Beginning of code segment

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

;\* Interrupt Vectors

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

.org $0000 ; Beginning of IVs

rjmp INIT ; Reset interrupt

.org $0002;INT0addr

rjmp TS\_0

.org $0004;INT1addr

rjmp TS\_1

.org $000A;INT4addr

rjmp TS\_4

.org $000C;INT5addr

rjmp TS\_5

.org $000E;INT6addr

rjmp TS\_6

.org $0010;INT7addr

rjmp TS\_7

; USART Transmit Complete

.org $0040

rcall ResetInterruptsOnTransmissionComplete

reti

; USART Recv Complete

;.org $003C

; nop

; USART Data register Empty

.org $003E

rcall TransmitMPR2

reti

; See page 171 of datasheet for interrupt execution order

.org $0046 ; End of Interrupt Vectors

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

;\* Program Initialization

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

INIT:

;Stack Pointer (VERY IMPORTANT!!!!) (Copied from Lab 1)

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 (Copied from Lab 1)

; Initialize Port B for output

ldi mpr, $FF ; Set Port B Data Direction Register

out DDRB, mpr ; for output

ldi mpr, ADDRESS ; Initialize Port B Data Register

out PORTB, mpr ; so all Port B outputs are ADDRESS

; Initialize Port D for input

ldi mpr, $00 ; Set Port D Data Direction Register

out DDRD, mpr ; for input

ldi mpr, $FF ; Initialize Port D Data Register

out PORTD, mpr ; so all Port D inputs are Tri-State

;USART1

;Set baudrate at 2400bps

ldi mpr, BAUD\_LOW

sts UBRR1L, mpr

ldi mpr, BAUD\_HIGH

sts UBRR1H, mpr

;Enable transmitter

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

rcall BaudTransmit

; Set interrupt flag

sei

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

;\* Main Program

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

MAIN:

in mpr, PIND

andi mpr, 1

cpi mpr, 1

breq POLL1

rcall TS\_0

POLL1:

in mpr, PIND

andi mpr, 2

cpi mpr, 2

breq POLL4

rcall TS\_1

POLL4:

in mpr, PIND

andi mpr, 16

cpi mpr, 16

breq POLL5

rcall TS\_4

POLL5:

in mpr, PIND

andi mpr, 32

cpi mpr, 32

breq POLL6

rcall TS\_5

POLL6:

in mpr, PIND

andi mpr, 64

cpi mpr, 64

breq POLL7

rcall TS\_6

POLL7:

in mpr, PIND

andi mpr, 128

cpi mpr, 128

breq MAIN

rcall TS\_7

rjmp MAIN

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

;\* Functions and Subroutines

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

; TS = "Transmit Symbol" interrupt

TS\_START:

rjmp TS\_END

TS\_0:

ldi mpr2, MOVE\_FORWARD

rjmp TS\_END

TS\_1:

ldi mpr2, MOVE\_BACKWARD

rjmp TS\_END

TS\_4:

ldi mpr2, TURN\_RIGHT

rjmp TS\_END

TS\_5:

ldi mpr2, TURN\_LEFT

rjmp TS\_END

TS\_6:

ldi mpr2, HALT

rjmp TS\_END

TS\_7:

ldi mpr2, FREEZE\_CMD

rjmp TS\_END

TS\_END:

;Logic

out PORTB, mpr2 ; Show message being transmitted

ldi mpr, ADDRESS

sts UDR1, mpr ; Transmit ADDRESS over USART1

; Mask next Interrupts (Only listen for UDRE)

ldi mpr, TX\_B | (1<<UDRIE1) ; TXEN1 | UDRIE1

sts UCSR1B, mpr

;ldi mpr, 0 ; Ignore all external interrupt

;out EIMSK, mpr

; Cleanup

; No cleanup to do

ret

TransmitMPR2: ; Calls when UDRE interrupt triggers

;enter stack frame

push mpr

in mpr, sreg

push mpr

;end enterstack frame

sts UDR1, mpr2 ; Send the actual command

; Mask next Interrupts (Only listen for TXC)

ldi mpr, TX\_B | (1<<TXCIE1)

sts UCSR1B, mpr

;Cleanup

ldi mpr, TX\_A | (1<<TXC1) ; Clear transmission complete interrupt

sts UCSR1A, mpr

; exit stack frame

pop mpr

out sreg, mpr

pop mpr

ret

; end exit stack frame

ResetInterruptsOnTransmissionComplete: ; calls on TXC1

push mpr

in mpr, sreg

push mpr

; Mask next Interrupts (Only Listen for external interrupts)

ldi mpr, TX\_B

sts UCSR1B, mpr

;Cleanup

ldi mpr, TX\_A | (1<<TXC1) ; Clear transmission complete interrupt

sts UCSR1A, mpr

ret

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

;\* Shared functions

;\*

;\* These functions are exactly the same in all files

;\*

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

BaudTransmit:

; Begin stack frame

push mpr

in mpr, sreg

push mpr

; End begin stack frame

; Configure Port D for output

in mpr, DDRD

ori mpr, 0b00001000

out DDRD, mpr

; Set baud control registers

ldi mpr, TX\_A

sts UCSR1A, mpr

ldi mpr, TX\_B

sts UCSR1B, mpr

ldi mpr, TX\_C

sts UCSR1C, mpr

; Cleanup

; Exit stack frame

pop mpr

out sreg, mpr

pop mpr

ret

; End exit stack frame

BaudRecv:

; Begin stack frame

push mpr

in mpr, sreg

push mpr

; End begin stack frame

; Configure Port D for input

in mpr, DDRD

andi mpr, 0b11111011

out DDRD, mpr

; Set baud control registers

ldi mpr, RX\_A

sts UCSR1A, mpr

ldi mpr, RX\_B

sts UCSR1B, mpr

ldi mpr, RX\_C

sts UCSR1C, mpr

; Cleanup

; Exit stack frame

pop mpr

out sreg, mpr

pop mpr

ret

; End exit stack frame

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

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

;\* Stored Program Data

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

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

;\* Additional Program Includes

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

## Reception

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

;\*

;\* Prather\_Eric\_Lab8\_rx\_sourcecode.asm

;\*

;\* Reciever robot

;\*

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

;\*

;\* Author: Eric Prather

;\* Date: February 27, 2020

;\*

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

.include "m128def.inc" ; Include definition file

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

;\* Shared Macros

;\*

;\* These macros are exactly the same in all files

;\* Uncomment desired BAUD number, comment undesired.

;\*

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

.equ ADDRESS = 69 ; Nice.

; Baud rate should be 2400

; Baud # (single): 416

;.equ BAUD\_LOW = 0b10100000

;.equ BAUD\_HIGH = 0b00000001

; Baud # (double): 832

.equ BAUD\_LOW = 0b01000000

.equ BAUD\_HIGH = 0b00000011

; Flag descriptions in lab report

.equ TX\_A = (1 << U2X1)

.equ RX\_A = (1 << U2X1)

.equ TX\_B = (1<<TXEN1) ; Interrupts: TXCIE1 and UDRE1

.equ RX\_B = (1<<RXEN1)

.equ TX\_C = (1<<USBS1) | (1<<UCSZ11) | (1<<UCSZ10) | (1<<UCPOL1) ;|(1<<UMSEL1)

.equ RX\_C = (1<<USBS1) | (1<<UCSZ11) | (1<<UCSZ10);| (1<<UCPOL1) ;|(1<<UMSEL1)

.equ UDRE\_INTERRUPT\_INVERSE\_MASK = 0b10111111

.equ TXC\_INTERRUPT\_INVERSE\_MASK = 0b11011111

.equ EXI\_MASK = 0b11110011 ; Push buttons

.equ MOVE\_FORWARD = 0b10110000

.equ MOVE\_BACKWARD = 0b10000000

.equ TURN\_RIGHT = 0b10100000

.equ TURN\_LEFT = 0b10010000

.equ HALTSIG = 0b11001000

.equ FREEZE\_CMD = 0b11111000

.equ FREEZE\_BROADCAST = 0b01010101

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

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

;\* Internal Register Definitions and Constants

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

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

;\* Variable and Constant Declarations

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

.def mpr = r16 ; Multi-Purpose Register

.def last\_msg = r20 ; reserved

.def waitcnt = r17 ; Wait Loop Counter

.def ilcnt = r18 ; Inner Loop Counter

.def olcnt = r19 ; Outer Loop Counter

.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 = ADDRESS ;(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:

;- Left whisker

; Not using interrupt, just polling

;- Right whisker

; Not using interrupt, just polling

;- USART receive

; USART Transmit Complete

;.org $0040

; rcall ResetInterruptsOnTransmissionComplete

; reti

; USART Recv Complete

.org $003C

rcall RECV\_USART

reti

; USART Data register Empty

;.org $003E

; rcall TransmitMPR2

; reti

.org $0046 ; End of Interrupt Vectors

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

;\* Program Initialization

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

INIT:

;Stack Pointer (VERY IMPORTANT!!!!) (Copied from Lab 1)

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 (Copied from Lab 1)

; Initialize Port B for output

ldi mpr, $FF ; Set Port B Data Direction Register

out DDRB, mpr ; for output

ldi mpr, $00 ; Initialize Port B Data Register

out PORTB, mpr ; so all Port B outputs are low

; Initialize Port D for input

ldi mpr, $00 ; Set Port D Data Direction Register

out DDRD, mpr ; for input

ldi mpr, $FF ; Initialize Port D Data Register

out PORTD, mpr ; so all Port D inputs are Tri-State

;USART1

;Set baudrate at 2400bps

ldi mpr, BAUD\_LOW

sts UBRR1L, mpr

ldi mpr, BAUD\_HIGH

sts UBRR1H, mpr

;Enable reciever

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

rcall BaudRecv

; Set robot state

ldi ZL, low(ROBOT\_STATE)

ldi ZH, high(ROBOT\_STATE)

ldi mpr, 0b00000000

st Z, mpr

; Set interrupt flag

sei

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

;\* Main Program

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

MAIN:

in mpr, PIND

andi mpr, 1

cpi mpr, 1

breq POLL1

rcall HitLeft

POLL1:

in mpr, PIND

andi mpr, 2

cpi mpr, 2

breq POLL5

rcall HitRight

POLL5:

in mpr, PIND

andi mpr, 32

cpi mpr, 32

breq POLL6

rcall FREEZE\_SELF

POLL6:

in mpr, PIND

andi mpr, 64

cpi mpr, 64

breq POLL7

rcall BROADCAST\_FREEZE\_DUMMY\_TURN

POLL7:

in mpr, PIND

andi mpr, 128

cpi mpr, 128

breq MAIN

rcall BROADCAST\_FREEZE

rjmp MAIN

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

;\* Functions and Subroutines

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

FREEZE\_SELF:

ldi last\_msg, FREEZE\_BROADCAST

rjmp RECV\_USART\_SKIP\_UDR

BROADCAST\_FREEZE\_DUMMY\_TURN:

; enter sf

push mpr

in mpr, sreg

push mpr

push ZL

push ZH

; end enter sf

ldi last\_msg, 0b10010000

ld mpr, Z ; mpr = \*Z

ori mpr, 0b10000000 ; write bit 8 to 1

st Z, mpr ; \*Z = mpr

rjmp BRANCH\_IF\_KNOWS\_ADDRESS

INFINITE\_LOOP:

rjmp INFINITE\_LOOP

RECV\_USART:

lds last\_msg, UDR1

RECV\_USART\_SKIP\_UDR:

; enter sf

push mpr

in mpr, sreg

push mpr

push ZL

push ZH

; end enter sf

ldi ZL, low(ROBOT\_STATE)

ldi ZH, high(ROBOT\_STATE)

; Freeze broadcast conditional

cpi last\_msg, FREEZE\_BROADCAST

brne BRANCH\_IF\_KNOWS\_ADDRESS

; Frozen

in mpr, PINB

push mpr

ldi mpr, 0b10010000

out PORTB, mpr

ld mpr, Z

andi mpr, 0b00000010

cpi mpr, 0b00000010

breq INFINITE\_LOOP ; term on 3rd freeze

ld mpr, Z

inc mpr ; increase number of freezes

st Z, mpr

rcall Wait ; 1 sec

rcall Wait ; 2 sec

rcall Wait ; 3 sec

rcall Wait ; 4 sec

rcall Wait ; 5 sec

pop mpr

out PORTB, mpr

rjmp CLEANUP

BRANCH\_IF\_KNOWS\_ADDRESS: ; Guaranteed not to be frozen

ld mpr, Z

andi mpr, 0b10000000 ; check if address already recieved

cpi mpr, 0b10000000

breq EXEC\_COMMAND

; Now actually check our address

cpi last\_msg, ADDRESS

brne CLEANUP ; was not meant for us

; we heard our address

ld mpr, Z

ori mpr, 0b10000000 ; write bit 8 to 1

st Z, mpr

rjmp CLEANUP

EXEC\_COMMAND: ; Do whatever the opcode tells

; Flush address from memory

ld mpr, Z

andi mpr, 0b01111111 ; write bit 8 to 0

st Z, mpr

cpi last\_msg, FREEZE\_CMD

brne NON\_FREEZE\_CMD

rcall BROADCAST\_FREEZE

rjmp CLEANUP

NON\_FREEZE\_CMD:

; last\_msg is set

mov mpr, last\_msg ; mpr = last\_msg

lsl mpr

out PORTB, mpr

rjmp CLEANUP

CLEANUP:

; exit sf

pop ZH

pop ZL

pop mpr

out sreg, mpr

pop mpr

ret

; end exit sf

BROADCAST\_FREEZE:

; enter sf

push mpr

in mpr, sreg

push mpr

; end enter sf

rcall BaudTransmit

ldi mpr, FREEZE\_BROADCAST

sts UDR1, mpr ; Send the actual command

; Mask next Interrupts (Only listen for TXC)

ldi mpr, TX\_B | (1<<TXCIE1)

sts UCSR1B, mpr

;Cleanup

ldi mpr, TX\_A | (1<<TXC1) ; Clear transmission complete interrupt

sts UCSR1A, mpr

; exit sf

pop mpr

out sreg, mpr

pop mpr

ret

; end exit sf

END\_BROADCAST: ; TXC1 -> 1

; enter sf

push mpr

in mpr, sreg

push mpr

; end enter sf

ldi mpr, TX\_A | (1<<TXC1) ; Clear transmission complete interrupt

rcall BaudRecv

; we don't care about udre, only txc

; exit sf

pop mpr

out sreg, mpr

pop mpr

ret

; end exit sf

; LAB 1 FUNCTIONS

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

; Sub: HitRight

; Desc: Handles functionality of the TekBot when the right whisker

; is triggered.

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

HitRight:

push mpr ; Save mpr register

push waitcnt ; Save wait register

in mpr, SREG ; Save program state

push 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

pop mpr ; Restore program state

out SREG, mpr ;

pop waitcnt ; Restore wait register

pop mpr ; Restore mpr

ret ; Return from subroutine

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

; Sub: HitLeft

; Desc: Handles functionality of the TekBot when the left whisker

; is triggered.

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

HitLeft:

push mpr ; Save mpr register

push waitcnt ; Save wait register

in mpr, SREG ; Save program state

push 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

pop mpr ; Restore program state

out SREG, mpr ;

pop waitcnt ; Restore wait register

pop mpr ; Restore mpr

ret ; Return from subroutine

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

; 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

ret ; Return from subroutine

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

;\* Shared functions

;\*

;\* These functions are exactly the same in all files

;\*

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

BaudTransmit:

; Begin stack frame

push mpr

in mpr, sreg

push mpr

; End begin stack frame

; Configure Port D for output

in mpr, DDRD

ori mpr, 0b00001000

out DDRD, mpr

; Set baud control registers

ldi mpr, TX\_A

sts UCSR1A, mpr

ldi mpr, TX\_B

sts UCSR1B, mpr

ldi mpr, TX\_C

sts UCSR1C, mpr

; Cleanup

; Exit stack frame

pop mpr

out sreg, mpr

pop mpr

ret

; End exit stack frame

BaudRecv:

; Begin stack frame

push mpr

in mpr, sreg

push mpr

; End begin stack frame

; Configure Port D for input

in mpr, DDRD

andi mpr, 0b11111011

out DDRD, mpr

; Set baud control registers

ldi mpr, RX\_A

sts UCSR1A, mpr

ldi mpr, RX\_B | (1<<RXCIE1)

sts UCSR1B, mpr

ldi mpr, RX\_C

sts UCSR1C, mpr

; Cleanup

; Exit stack frame

pop mpr

out sreg, mpr

pop mpr

ret

; End exit stack frame

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

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

;\* Stored Program Data

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

.dseg

.org $0100

ROBOT\_STATE:

.byte 1

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

;\* Additional Program Includes

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