ECE 375 Lab 8

Remotely operated vehicles

Lab Time: Friday 4-6

Aaron Vaughan

Bradley Heenk

TA Signature

1 Introduction

The purpose of this lab is to familiarize ourselves with data transmission and reception using USART protocol hardware to send and receive data through a medium. There are no additional includes needed for this as there are two different hardware implimentations of the USART modules within the atmega128 micro-controller. We use one of these modules to implement a remote control transmitter and robot receiver that communicate through infra red technology. Our task is to create an interactive toy that can drive remotely with commands sent by a remote as well as function independently of the remote with the basic bump-bot functionality. Additionally the robots can interact in a sort of freeze-tag game. The robots and remotes must have matched addresses in order to receive commands from the remote but all robots can receive the freeze command sent by the other robots in order to participate in the freeze-tag game.

The first part of the challenge code simply uses one of the 16-bit timer/counter modules to implement the wait functions. The second part of the challenge is done by literally copy-and-pasting the LAB7 code into our working receiver code and modifying the functionality of the Halt and Freeze action code reception. It changes the freeze code to speed up and halt to speed down. It also displays the current speed on the lowest 4 LED's via a 4-bit counter that will not overflow.

2 Program Overview

This program requires a ton of setup. A lot of reading and understanding of the hardware documentation is required in order to get the setup for the USART modules correct. Using the IR technology to send and receive requires us to use a slow enough rate of data transmission, called BAUD rate, that the IR technology can respond to. The typical protocol for the BAUD rate used in IR sensors is 2400 bps. The standard initialization of the stack and input/outputs are done after the USART module is set up via its three setup registers. Some special care was needed for the setting up the inputs as the USART1 module does interact with bit-2 and bit-3 of PORTD.The program overview must be explained as a transmitter and a receiver. To delineate these differences, I will itemize the separate programs from here on.

- The Transmitter
  We use polling to check for a button press. When polling, if a button is found to be activated, the main program jumps into a subroutine that deals with the transmission. Each subroutine calls the transmission subroutine, which waits until the sending buffer is empty and then loads it data to be transmitted. This transmission process is repeated twice; once for the address, and once for the command.

- The Receiver
  We use interrupts to implement the bump-bot behavior and the USART functionality for the receiver. This means that the Main program is empty and just loops forever waiting for something to do. When a command comes in, the bot will first check to see if the first byte is either the correct address or the "freeze signal". "Freeze signal" is a separate signal sent out by a bot that receives the freeze command from the remote. In this way, one can freeze other robots without freezing itself. For all other functionality, the receiver first verifies the address of the remote matches its own bot address. If the address matches, it will then check the operation sent by the remote and display on the LEDs an output that will match that command. The output for each command will follow the bump-bot behavior for turn right, turn left, go forward, go backward, and halt.

- The Wait Challenge
  This portion of the challenge just replaces the wait function with a 16-bit timer/counter wait function. We used interrupts to jump in and out of the wait function within the hit left and hit right interrupt routine. It works by setting the TCNT register to the correct value to cause a one second counting cycle and polling the TOV3 flag in a loop. When it is set, the interrupt handles clearing the flag and returns back to the wait function. After that, the wait function just returns back to the HitLeft or HitRight function to finish out the next task. It should be noted that although canvas has a submission for challenge code for a transmitter that our transmitter code is not changed for either challenge code, both were implemented within the receiver code.

- The PWM challenge code
  This portion of the challenge is just copy and paste from lab

7 to implement a functional remote operated speed control with 4-bit resolution (16 speeds). The counter will not overflow in either direction. The transmitter code is identical for challengecode and sourcecode. When a speed down is received from the transmitte the TekBot will simulate a decrease in speed. Similar functionality will follow for speed up. We use Fast PWM mode to strobe PINB7 and PINB4 using timer/counter0 and timer/counter2. The speed changes are done by changing the OCR0 and OCR2 values by 17 (increment for speed up and decrement for speed down). The 4-bit counter is done by polling the current state and incrementing a binary value and updating the output on the lower 4 LED outputs.

3 Internal Register Definitions and Constants

We use 5 registers to perform all of the operations. Less could have been used if we took advantage of push/pop instructions but meh, we went for easy to read register names instead of recycling mpr over and over again. Constants were selected for the counter values for the desired functionality and macros were assigned constant values to increase the readability of the code. These macros were used for the transmitter and receiver code for each command code and each action code.

4 Interrupt Vectors

- Transmitter
  There were no interrupt vectors used for the transmitter.
- Receiver
  The receiver uses interrupt vectors INT0 and INT1 (Address $0002, and $0004 respectively) for the bump-bot behavior that are tied to bit-0 and bit-1 of PORTD. INT0 and INT1 need to be set with external interrupt sense control on the falling edge since the buttons are active low. We then mask external interrupts INT3 through INT7. We also use interrupt vectors to implement the receiving and transmission of data at address numbers $0003C, and $0040 respectively. The receiver and transmitter interrupts

had to be enabled in the initialization of the USART module discussed below.

- Wait Challenge
  We used timer interrupts to implement the wait challenge. This was $003A. It is used to simply clear the TOV3 flag. It just returns from interrupt when called.

5 Program Initialization

For bot transmitter and receiver we set up all of PORTB for output, and initialized the stack pointer. Both the receiver and transmitter utilized the USART1 so we had to fiddle with three registers to get them correct. The USART units have I/O on PORTD so care must be taken when setting up PORTD as an output, since one of these terminals is actually an input now. We referred to the atmega128 manual for this insight.

The USART1 initialization for both modules were set up to use double data rate with 2400 bps BAUD rate, 8-bit data frames, with two stop bits, and no error detection. The BAUD rate was set by loading values into UBRR1L and UBRR1H. The value was found in a table in the atmega128 data sheet. This configuration utilizes all three UCSR1-A,B,C registers.

The transmitter setup used the above configuration as a general setup. Then we set the TXEN1 (Transmitter enable) bit in the UCSR1B register. The receiver bot setup also set the TXEN1 bit with the addition of the RXEN1 (Receiver enable) bit and the RXIEN1 (USART1 Rx complete interrupt) bit.

The Wait challenge initialization required us to initialize one of the 16-bit counters. We chose to set it up in normal mode with interrupts enabled.

6 Main Program

- Transmitter
  Main just sits in a loop polling the input buttons with the SBRS mnemonic. If the SBRS mnemonic finds a cleared bit that corresponds

to that button input from PORTD then it will call a subroutine
to send the action command.

- Receiver
  Main is empty and just loops forever waiting for an interrupt
  to trigger some action.
- Challenge code
  The challenge codes were implemented within the receiver code.
  Nothing changes with the transmitter code for either challenge.
  Nothing is changed with the main program.

7 Subroutines

* Transmitter Subroutines

- Function Name: USART_Transmit_instruction
  Description: This code uses polling of the UDRE flag to determine
  when to send the data
- Function Name: GoRight
  Description: Sends the GoRight command signal with address
- Function Name: GoLeft
  Description: Sends the GoLeft command signal with address
- Function Name: GoFwd
  Description: Sends the Gofwd command signal with address
- Function Name: GoBck
  Description: Sends the GoBck command signal with address
- Function Name: Stop
  Description: Sends the Stop command signal with address

- Function Name: Freeze_All
  Description: Sends the Freeze_All command signal with address

- Funciton Name: Wait
  Description: 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

*Receiver Subroutines

- Function Name: USART Receive
  Description: Checks the first USART packet from UDR1 and calls two different functions to either be fronzen from the universal

- Function Name: USART_Transmit_instruction
  Description: This code uses polling of the UDRE flag to determine when to send the data

- Function Name: RX Freeze
  Description: Gets called when we reveice the universal freeze signal takes care of saving the previous PORTB state and waiting 5 seconds before inputs can be used again

- Function Name: Op Code
  Description: Gets called to check the address of the first USART and calls the various other functions to receive the correct signal from the transmitter

- Function Name: HitLeft
  Description: Handles functionality of the TekBot when the right whisker is triggered.

- Function Name: HitRight
  Description:Handles functionality of the TekBot when the right

whisker is triggered.

- Funciton Name: Wait
  Description: 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

- Function Name: Queue Fix Function
  Description: This fucntions counts to around 600 micro seconds this is used to help avoid queue delays since we know our Atmega128 chip runs at 16 mhz we know each clock cycle will be 1 / 16 mhz and convert it to microseconds. Now we can take 600 value and divide by out result which was used to determine how many loops of 255 clock cycles we would need to stack hence the inner loop and outer loops

*Wait Challenge

- Function Name: TimerWait
  Description: This implements a one second wait cycle driven by the timer /counter3

8 Stored Program Data

There is not any stored program data for this lab.

9 Additional Program Includes

There were no additional program includes for this lab.

10 Additional Questions

1. There are no additional questions for this lab.

   This is my favorite part of this lab. :)

11 Difficulties

The hardest part was getting the setup correct. We struggled with the USART module interfering with PORTD and with external interrupt sense control. The rest of the lab was actually not that hard. There was a ton of reading over and over through the text to understand how this transmission protocol functioned but once we understood it the code flowed right out and our modules just worked. Debugging with interrupts was a bit tricky too but we found that by placing function calls into main we could simulate interrupt sequences one at a time.

12 Conclusion

This lab took a lot of debugging but it was a ton of fun. Bradley has a sweet oscilloscope in his room, well its really a whole lab. We tested the output from our transmitter and had the ability to visualize the output from our transmitter to help us debug the freeze command operation. We gained a ton of useful skills in this course and this lab really puts the icing on the cake. Being able to implement a working remote control and receiver module will definitely be of use in future courses and possibly in the workforce when I graduate.

13 Transmitter Source Code

```
;***********************************************************
;*
;* Aaron_Vaughan_and_Bradley_Heenk_Lab8_Tx_sourcecode.asm
;*
;* Enter the description of the program here
```

```
;*
;* This is the TRANSMIT skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*  Author: Aaron Vaughan and Bradley Heenk
;*    Date: 11/26/2019
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def dataIO = r17            ; Transmission data register
.def waitcnt = r18 ; Wait Loop Counter
.def ilcnt = r19 ; Inner Loop Counter
.def olcnt = r20 ; Outer Loop Counter

.equ WTime = 50 ; Time to wait in wait loop
.equ Address = $1A ; This is the Rx/Tx ID number

.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 Freeze =  (0b11111000) ;freeze action code

;***********************************************************
```

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

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

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
; Initialize the 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

    ; 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/output
ldi mpr, $0C ; Set Port D Data Direction Register
out DDRD, mpr ; for input/output
ldi mpr, $F3 ; Initialize Port D Data Register
out PORTD, mpr ; so all Port D inputs are Tri-State

ldi mpr, $F3
out PORTD, mpr
; USART1
; Set baudrate at 2400bps
ldi mpr, high(832)
sts UBRR1H, mpr
```

```
ldi mpr, low(832)
sts UBRR1L, mpr
; Clear set 2x
ldi mpr, ((1<<U2X1))
sts UCSR1A, mpr
; Enable transmitter
ldi mpr, (1<<TXEN1) ; 0b01000000
sts UCSR1B, mpr
; Set frame format: 8 data bits, 2 stop bits
ldi mpr, 0b00001110 ; 0b00001110
sts UCSR1C, mpr

;Other

;************************************************************
;* Main Program
;************************************************************
MAIN:
; Start polling PORTB to check what we are supposed
; to be doing.
in mpr, PIND
andi mpr, 0b11110011 ; Mask Tx/Rx pins


; If button 0 is pressed send GoRight command
sbrs mpr, PIND0
rcall GoRight
; If button 1 is pressed send GoLeft command
sbrs mpr, PIND1
rcall GoLeft
; If button 4 is pressed send MovFwd command
sbrs mpr, PIND4
rcall GoFwd
; If button 5 is pressed send MovBck command
sbrs mpr, PIND5
rcall GoBck
; If button 6 is pressed send Stop command
sbrs mpr, PIND6
rcall Stop
; If button 7 is pressed
```

```
sbrs mpr, PIND7
rcall Freeze_All

rjmp MAIN

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

;-----------------------------------------------------------
; Func: USART_Transmit_instruction
; Desc: This code uses polling of the UDRE flag to determine
; when to send the data
;-----------------------------------------------------------
USART_Transmit:
in mpr, SREG ; Save program state
push mpr ;

; Wait for the UDRE to be empty
lds mpr, UCSR1A
sbrs mpr, UDRE1
rjmp USART_Transmit
; Load dataIO into buffer and send it
sts UDR1, dataIO

pop mpr ; Restore program state
out SREG, mpr ;

ret

;----------------------------------------------------------------
; Sub: GoRight
; Desc: Sends the GoRight command signal with address
;----------------------------------------------------------------
GoRight:
push mpr ; Save state of processor
in mpr, SREG ;
push mpr ;
push dataIO
push waitcnt
```

```
ldi dataIO, Address ; Load Address
rcall USART_Transmit ; Transmit Address

ldi dataIO, turnR ; Load Command
rcall USART_Transmit ; Transmit Command

pop waitcnt ; Restore state of processor
pop dataIO ;
pop mpr ;
out SREG, mpr ;
pop mpr

ret ; Return from subroutine

;----------------------------------------------------------------
; Sub: GoLeft
; Desc: Sends the GoLeft command signal with address
;----------------------------------------------------------------
GoLeft:
push mpr ; Save state of processor
in mpr, SREG ;
push mpr ;
push dataIO
push waitcnt

ldi dataIO, Address ; Load Address
rcall USART_Transmit ; Transmit Address
;ldi waitcnt, Pause
;rcall Wait

ldi dataIO, TurnL ; Load Command
rcall USART_Transmit ; Transmit Command
;ldi waitcnt, Pause
;rcall Wait

pop waitcnt ; Restore processor state
pop dataIO ;
pop mpr ;
out SREG, mpr ;
```

```
pop mpr

ret ; Return from subroutine


;-----------------------------------------------------------------
; Sub: GoFwd
; Desc: Sends the Gofwd command signal with address
;-----------------------------------------------------------------
GoFwd:
push mpr ; Restore processor state
in mpr, SREG ;
push mpr ;
push dataIO
push waitcnt

ldi dataIO, Address ; Load Address
rcall USART_Transmit ; Transmit Address
;ldi waitcnt, Pause
;rcall Wait

ldi dataIO, MovFwd ; Load Command
rcall USART_Transmit ; Transmit Command
;ldi waitcnt, Pause
;rcall Wait

pop waitcnt ; Restore processor state
pop dataIO ;
pop mpr ;
out SREG, mpr ;
pop mpr ;

ret ; Return from subroutine


;-----------------------------------------------------------------
; Sub: GoBck
; Desc: Sends the GoBck command signal with address
;-----------------------------------------------------------------
GoBck:
```

```
push mpr ; Save processor state
in mpr, SREG ;
push mpr ;
push dataIO ;
push waitcnt ;

ldi dataIO, Address ; Load Address
rcall USART_Transmit ; Transmit Address
;ldi waitcnt, Pause
;rcall Wait

ldi dataIO, MovBck ; Load Command
rcall USART_Transmit ; Transmit Command
;ldi waitcnt, Pause
;rcall Wait

pop waitcnt ; Restore processor state
pop dataIO ;
pop mpr ;
out SREG, mpr ;
pop mpr

ret ; Return from subroutine


;-----------------------------------------------------------------
; Sub: Stop
; Desc: Sends the Stop command signal with address
;-----------------------------------------------------------------
Stop:
push mpr ; Save processor state
in mpr, SREG ;
push mpr ;
push dataIO
push waitcnt

ldi dataIO, Address ; Load Address
rcall USART_Transmit ; Transmit Address
ldi waitcnt, Pause
rcall Wait
```

```
ldi dataIO, Halt ; Load Command
rcall USART_Transmit ; Transmit Command
ldi waitcnt, Pause
rcall Wait

pop waitcnt ; Restore processor state
pop dataIO ;
pop mpr ;
out SREG, mpr ;
pop mpr

ret ; Return from subroutine

;-----------------------------------------------------------------
; Sub: Freeze
; Desc: Sends the Freeze signal
;-----------------------------------------------------------------
Freeze_All:
push mpr ; Save processor state
in mpr, SREG ;
push mpr ;
push dataIO
push waitcnt

ldi dataIO, Address ; Load Address
rcall USART_Transmit ; Transmit Address


ldi dataIO, Freeze ; Load Command
rcall USART_Transmit ; Transmit Command

ldi waitcnt, WTime ; 500ms wait period to prevent spamming of the freeze command
rcall Wait
pop waitcnt ; Restore processor state
pop dataIO ;
pop mpr ;
out SREG, mpr ;
pop 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



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

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

14 Receiver Sourcecode

‘

```
;************************************************************
;*
;* Aaron_Vaughan_and_Bradley_Heenk_Lab8_Rx_sourcecode.asm
;*
;* This program is the USART receiver which takes in
;* various command from a transmitter and preforms
;* different functions based on what the transmitter sends.
;* Also takes use of the old TurnLeft and TurnRight fucntions
;* from previous labs.
;*
;* This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;************************************************************
;*
;*  Author: Aaron Vaughan and Bradley Heenk
;*    Date: 11/26/2019
;*
;************************************************************

.include "m128def.inc" ; Include definition file

;************************************************************
;* Internal Register Definitions and Constants
;************************************************************
.def mpr = r16 ; Multipurpose register
.def check = r17
.def freezecount = r18
.def dataIO = r19
.def rxcheck = r20
.def memory = r21
.def waitcnt = r23
.def ilcnt = r24
.def olcnt = r25

.equ LongWait = 250

.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 Address = $1A ;(Enter your robot's address here (8 bits))
.equ Address = $69 ;(Enter your robot's address here (8 bits))

;//////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
;//////////////////////////////////////////////////////////////
.equ MovFwd =  (1<<EngDirR|1<<EngDirL) ;0b01100000 Move Forward Action Code
.equ MovBck =  $00 ;0b00000000 Move Backward Action Code
.equ TurnR =   (1<<EngDirL) ;0b01000000 Turn Right Action Code
.equ TurnL =   (1<<EngDirR) ;0b00100000 Turn Left Action Code
.equ Halt =    (1<<EngEnR|1<<EngEnL) ;0b10010000 Halt Action Code

.equ MovFwdAct =  ($80|1<<(EngDirR-1)|1<<(EngDirL-1)) ;0b10110000 Move Forward Action Co
.equ MovBckAct =  ($80|$00) ;0b10000000 Move Backward Action Code
.equ TurnRAct =   ($80|1<<(EngDirL-1)) ;0b10100000 Turn Right Action Code
.equ TurnLAct =   ($80|1<<(EngDirR-1)) ;0b10010000 Turn Left Action Code
.equ HaltAct =    ($80|1<<(EngEnR-1)|1<<(EngEnL-1)) ;0b11001000 Halt Action Code
.equ FreezeAct =  (0b11111000) ;0b11111000 Freeze Action Code
.equ FreezeAll =  (0b01010101) ;0b01010101 Freeze All 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
.org $0002 ; Beginning of IVs
```

```
rcall  Hitright ; Reset interrupt
reti
;- Right whisker
.org $0004 ; Beginning of IVs
rcall  Hitleft ; Reset interrupt
reti
;- USART receive
.org $003C ; Beginning of IVs
rcall  USART_Receive ; Reset interrupt
reti

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Stack Pointer (VERY IMPORTANT!!!!)
ldi mpr, low(RAMEND)
out SPL, mpr
ldi mpr, high(RAMEND)
out SPH, mpr
;I/O Ports
ldi mpr, $FF
out DDRB, mpr
ldi mpr, $00
out PORTB, mpr
; Initialize I/O Ports
ldi mpr, 0b00001000
out DDRD, mpr
ldi mpr, 0b11110011
out PORTD, mpr
;USART1
;Set baudrate at 2400bps
ldi mpr, low(832)
sts UBRR1L, mpr
ldi mpr, high(832)
sts UBRR1H, mpr
;Enable receiver and enable receive interrupts
ldi mpr, (1<<U2X1)
```

```
sts UCSR1A, mpr
ldi mpr, (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1)
sts UCSR1B, mpr
;Set frame format: 8 data bits, 2 stop bits
ldi mpr, (1<<UCSZ10)|(1<<UCSZ11)|(1<<USBS1)|(1<<UPM01)
sts UCSR1C, mpr
;External Interrupts
; Initialize external interrupts
ldi mpr, 0b00001010 ; Set the Interrupt Sense Control to falling edge
sts EICRA, mpr
; Set the Interrupt Sense Control to falling edge
ldi mpr, 0b00000000
out EICRB, mpr
;Set the External Interrupt Mask
ldi mpr, 0b00000011 ; Set value to what we want to hide
out EIMSK, mpr

;Other
ldi check, 0 ; Set our to zero
ldi freezecount, 0 ; Set our freeze count to zero

ldi memory, 0b01100000 ; Setup our memory and setup for move forward
out PORTB, memory ; Set our LED's to move forward

sei

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

;***********************************************************
;* Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------
; Sub: USART Receive
; Desc: Checks the first USART packet from UDR1 and calls two
; different functions to either be fronzen from the
; universal
```

```
;-----------------------------------------------------------------
USART_Receive:

lds mpr, UDR1

; Check for the freeze only command
cpi mpr, FreezeAll
breq RX_FREEZE ; We got universal freeze go ahead and freeze

; Check the address
rjmp USART_OPCODE ; Normal case check address and opcode

;-----------------------------------------------------------
; Func: USART_Transmit_instruction
; Desc: This code uses polling of the UDRE flag to determine
; when to send the data
;-----------------------------------------------------------
USART_Transmit:

; Wait for the UDRE to be empty
lds mpr, UCSR1A
sbrs mpr, UDRE1
rjmp USART_Transmit
; Load dataIO into buffer and send it
sts UDR1, dataIO

ret

;-----------------------------------------------------------------
; Sub: RX Freeze
; Desc: Gets called when we reveice the universal freeze signal
; takes care of saving the previous PORTB state and waiting
; 5 seconds before inputs can be used again
;-----------------------------------------------------------------
RX_FREEZE:

in mpr, PORTB ; Save old PORTB state
push mpr ; Push mpr on the stack to preverse PORTB state

ldi mpr, Halt ; Load the HALT command into mpr
```

```
out PORTB, mpr ; Tell our LED's to be halted

ldi waitcnt, LongWait ; Load 2.5 seconds into wantcnt
rcall waitfunc ; Wait 2.5 seconds
rcall waitfunc ; Wait 2.5 seconds

pop mpr ; Pop mpr on the stack to get previous PORTB state
out PORTB, mpr ; Restore old PORTB state

cpi freezecount, 2 ; Check to see if we've been frozen 3 times
breq DIEINFIRE ; If so go ahead and loop forever and take no commands
inc freezecount ; Otherwise increment that we've been frozen

ret

DIEINFIRE:
rjmp DIEINFIRE

ret


;-----------------------------------------------------------------
; Sub: Op Code
; Desc: Gets called to check the address of the first USART
; and calls the various other functions to reveice the
; correct signal from the transmitter
;-----------------------------------------------------------------
USART_OPCODE:

sbrs check, 0 ; Checks lsb in check to see if its set and skip the next function
rjmp Address_Check ; If its not set go ahead and compare the address

ldi check, 0 ; Setup check to zero since we know the next set of DATA

cpi mpr, FreezeAct ; Load action into mpr
breq FreezeFunc ; If true call FreezeFunc
cpi mpr, MovFwdAct ; Load action into mpr
breq MovFwdFunc ; If true call MovFwdFunc
cpi mpr, MovBckAct ; Load action into mpr
breq MovBckFunc ; If true call MovBckFunc
```

```
cpi mpr, TurnRAct ; Load action into mpr
breq TurnRFunc ; If true call TurnRFunc
cpi mpr, TurnLAct ; Load action into mpr
breq TurnLFunc ; If true call TurnLFunc
cpi mpr, HaltAct ; Load action into mpr
breq HaltFunc ; If true call HaltFunc

rjmp OPCODEEXIT ; Exit if nothing compares

FreezeFunc:
; Enable the receiver
ldi mpr, (1<<RXCIE1)|(1<<TXCIE1)|(1<<RXEN1)|(1<<TXEN1)
sts UCSR1B, mpr

ldi dataIO, FreezeAll ; Load the transmit dataIO with FreezeAll
rcall USART_Transmit ; Call our USART_Transmit to send out the freeze

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

rjmp OPCODEEXIT

MovFwdFunc:
ldi mpr, MovFwd ; Setup mpr with MovFwdFunc Command
out PORTB, mpr ; Load the command into PORTB
rjmp OPCODEEXIT

MovBckFunc:
ldi mpr, MovBck ; Setup mpr with MovBckFunc Command
out PORTB, mpr ; Load the command into PORTB
rjmp OPCODEEXIT

TurnRFunc:
ldi mpr, TurnR ; Setup mpr with TurnRFunc Command
out PORTB, mpr ; Load the command into PORTB
rjmp OPCODEEXIT

TurnLFunc:
ldi mpr, TurnL ; Setup mpr with TurnLFunc Command
```

```
out PORTB, mpr ; Load the command into PORTB
rjmp OPCODEEXIT

HaltFunc:
ldi mpr, Halt ; Setup mpr with HaltFunc Command
out PORTB, mpr ; Load the command into PORTB
rjmp OPCODEEXIT

Address_Check:
cpi mpr, Address ; Check to see if we are the correct address
breq Okay_Address ; If true setup setup our check register
rjmp OPCODEEXIT

Okay_Address:
ldi check, 1 ; Load a one into check allow fucntion to bypass check next time
rjmp OPCODEEXIT

OPCODEEXIT:

ret

;----------------------------------------------------------------
; Sub: HitLeft
; Desc: Handles functionality of the TekBot when the right whisker
; is triggered.
;----------------------------------------------------------------
HitLeft: ; Begin a function with a label

; Save variable by pushing them to the stack
push mpr
push waitcnt
in   mpr, SREG
push mpr

in memory, PORTB ; Save old PORTB state

; Execute the function here
; Preform reverse command wait 100 ms
ldi mpr,$00
out PORTB,mpr
```

```
ldi waitcnt, 100
rcall WaitFunc

; Preform left command wait 100 ms
ldi mpr,$20 ; 0010 0000
out PORTB,mpr
ldi waitcnt, 100
rcall WaitFunc

rcall QueueFix
ldi mpr, $03
out EIFR, mpr

out PORTB, memory ; Restore old PORTB state

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

ret ; End a function with RET

;-----------------------------------------------------------------
; Sub: HitRight
; Desc: Handles functionality of the TekBot when the right whisker
; is triggered.
;-----------------------------------------------------------------
HitRight: ; Begin a function with a label

; Save variable by pushing them to the stack
push mpr
push waitcnt
in   mpr, SREG
push mpr

in memory, PORTB ; Save old PORTB state

; Execute the function here
; Preform reverse command wait 100 ms
```

```
ldi mpr,0x0
out PORTB,mpr
ldi waitcnt, 100
rcall WaitFunc

; Preform right command wait 100 ms
ldi mpr,0x40
out PORTB,mpr
ldi waitcnt, 100
rcall WaitFunc

rcall QueueFix
ldi mpr, $03
out EIFR, mpr

out PORTB, memory ; Restore old PORTB state

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

ret ; End a function with RET

;-----------------------------------------------------------
; Func: Queue Fix Function
; Desc: This fucntions counts to around 600 micro seconds
; this is used to help avoid queue delays since we
; know our Atmega128 chip runs at 16 mhz we know each
; clock cycle will be 1 / 16 mhz and convert it to
; microseconds. Now we can take 600 value and divide
; by out result which was used to determine how many
; loops of 255 clock cycles we would need to stack
; hence the inner loop and outer loops
;-----------------------------------------------------------
QueueFix:
push ilcnt ; Push registers onto the stack
push olcnt
ldi ilcnt, 255 ; Load 255 into ilcnt
ldi olcnt, 30 ; Load 30 into olcnt
ILOOPQUEUE:
dec ilcnt ; Decrement ilcnt
brne ILOOPQUEUE ; Branch if not equal to zero to ILOOPQUEUE
OLOOPQUEUE:
ldi ilcnt, 255 ; Load 255 into ilcnt
dec olcnt ; Decrement olcnt
brne ILOOPQUEUE ; Branch if not equal to zero to ILOOPQUEUE

pop olcnt ; Pop registers off the stack
pop ilcnt
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
;----------------------------------------------------------------
WaitFunc:
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
;***********************************************************
```

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

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

15 BOTH Challenge codes

;*** ;*
;* Aaron$_{V}aughan_{a}nd_{B}radley_{H}eenk_{L}ab8_{R}x_{c}hallenegecode.asm; *; *ThisprogramistheUSARTreceiverwh$
****
$*; *; *Author\ :\ AaronVaughanandBradleyHeenk; *Date\ :\ 11/26/2019; *; * *$
****

.include "m128def.inc" ; Include definition file

;*** ;*
Internal Register Definitions and Constants ;**
.def mpr = r16 ; Multipurpose register .def check = r17 .def freezecount
= r18 .def dataIO = r19 .def rxcheck = r20 .def memory = r21 .def
waitcnt = r23 .def ilcnt = r24 .def olcnt = r25

.equ LongWait = 250

.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 Address = $1A; (Enteryourrobot'saddresshere(8bits))$

;//
;These macros are the values to make the TekBot Move. ;//////////////////////////////////
.equ MovFwd = (1<<EngDirR|1<<EngDirL) ;0b01100000 Move Forward Action
Code .equ MovBck = $00; 0b00000000 MoveBackwardActionCode.equTurnR =$
$(1 << EngDirL); 0b01000000 TurnRightActionCode.equTurnL = (1 << EngDirR); 0b00100000 TurnLe$
$(1 << EngEnR|1 << EngEnL); 0b10010000 HaltActionCode$

.equ MovFwdAct = $(80|1 << (EngDirR-1)|1 << (EngDirL-1)); 0b10110000 MoveForwardActio$
$(80|00); 0b10000000 MoveBackwardActionCode.equTurnRAct = (80|$1<<(EngDirL-1))

;0b10100000 Turn Right Action Code .equ TurnLAct = $(80|1 << (EngDirR-$
$1)); 0b10010000 TurnLeftActionCode.equHaltAct =$ (80|1<<(EngEnR-1)|1<<(EngEnL-1))
;0b11001000 Halt Action Code .equ FreezeAct = (0b11111000) ;0b11111000
Freeze Action Code .equ FreezeAll = (0b01010101) ;0b01010101 Freeze
All Code .equ Step = $11; EachStepisdecimal17$

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

;*** ;*
Interrupt Vectors ;***
.org $0000; BeginningofIVsrjmpINIT; Resetinterrupt$

;Should have Interrupt vectors for: ;- Left whisker .org $0002; BeginningofIVsrcallHi$
; Beginning of IVs rcall Hitleft ; Reset interrupt reti ;- USART receive
.org $003C; BeginningofIVsrcallUSART_Receive; Resetinterruptreti; -Timerinterrupt.org$003A
reti

.org $0046; EndofInterruptVectors$

;*** ;*
Program Initialization ;***
INIT: ;Stack Pointer (VERY IMPORTANT!!!!) ldi mpr, low(RAMEND) out
SPL, mpr ldi mpr, high(RAMEND) out SPH, mpr ;I/O Ports ldi mpr, $FFoutDDRB, mprldimpr,$0
out PORTB, mpr ; Initialize I/O Ports ldi mpr, 0b00001000 out DDRD,
mpr ldi mpr, 0b11110011 out PORTD, mpr ;USART1 ;Set baudrate at 2400bps
ldi mpr, low(832) sts UBRR1L, mpr ldi mpr, high(832) sts UBRR1H, mpr
;Enable receiver and enable receive interrupts ldi mpr, (1<<U2X1)
sts UCSR1A, mpr ldi mpr, (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1) sts UCSR1B,
mpr ;Set frame format: 8 data bits, 2 stop bits ldi mpr, (1<<UCSZ10)|(1<<UCSZ11)|(1<<US
sts UCSR1C, mpr ;External Interrupts ; Initialize external interrupts
ldi mpr, 0b00001010 ; Set the Interrupt Sense Control to falling edge
sts EICRA, mpr ; Set the Interrupt Sense Control to falling edge ldi
mpr, 0b00000000 out EICRB, mpr ;Set the External Interrupt Mask ldi
mpr, 0b00000011 ; Set value to what we want to hide out EIMSK, mpr

; Configure 8-bit Timer/Counters ldi mpr, $79; Noprescaling, FastPWM, InvertedCompareo$
; No prescaling, Fast PWM, Inverted Compare out TCCR2, mpr

; init the counter/timer3 normal, 256, interrupt enable ldi mpr,
high(3036) ; Set the starting value to 3036 sts TCNT3H, mpr ldi mpr,
low(3036) sts TCNT3L, mpr

ldi mpr, 0 ; Set normal mode sts TCCR3A, mpr ldi mpr, (1<<CS32)

; 256 prescaler sts TCCR3B, mpr ldi mpr, (1<<TOIE3) ; Enable the TOV interrupt sts ETIMSK, mpr

;Other ldi memory, 0b01100000 ; Setup our memory and setup for move forward out PORTB, memory ; Set our LED's to move forward

ldi olcnt, 0 ; Start at min speed ldi ilcnt, 0 ; Start at min speed ldi check, 0 ; Set our to zero ldi freezecount, 0 ; Set our freeze count to zero

sei

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

;*** ;* Functions and Subroutines ;***
;---
; Sub: USART Receive ; Desc: Checks the first USART packet from UDR1 and calls two ; different functions to either be fronzen from the ; universal ;---
USART$_Receive:$

lds mpr, UDR1

; Check the address rjmp USART$_OPCODE; Normalcasecheckaddressandopcode$

;---
; Sub: Op Code ; Desc: Gets called to check the address of the first USART ; and calls the various other functions to reveice the ; correct signal from the transmitter ;---
USART$_OPCODE:$

sbrs check, 0 ; Checks lsb in check to see if its set and skip the next function rjmp Address$_Check; Ifitsnotsetgoaheadandcomparetheaddress$

ldi check, 0 ; Setup check to zero since we know the next set of DATA

cpi mpr, FreezeAct ; Load action into mpr breq FreezeFunc ; If true call FreezeFunc cpi mpr, MovFwdAct ; Load action into mpr breq MovFwdFunc ; If true call MovFwdFunc cpi mpr, MovBckAct ; Load action into mpr breq MovBckFunc ; If true call MovBckFunc cpi mpr, TurnRAct ; Load action into mpr breq TurnRFunc ; If true call TurnRFunc cpi mpr, TurnLAct ; Load action into mpr breq TurnLFunc ; If true call TurnLFunc cpi mpr, HaltAct ; Load action into mpr breq HaltFunc ;

If true call HaltFunc

rjmp OPCODEEXIT ; Exit if nothing compares

FreezeFunc: rcall IncSpd rjmp OPCODEEXIT

MovFwdFunc: ldi mpr, MovFwd ; Setup mpr with MovFwdFunc Command
out PORTB, mpr ; Load the command into PORTB rjmp OPCODEEXIT

MovBckFunc: ldi mpr, MovBck ; Setup mpr with MovBckFunc Command
out PORTB, mpr ; Load the command into PORTB rjmp OPCODEEXIT

TurnRFunc: ldi mpr, TurnR ; Setup mpr with TurnRFunc Command
out PORTB, mpr ; Load the command into PORTB rjmp OPCODEEXIT

TurnLFunc: ldi mpr, TurnL ; Setup mpr with TurnLFunc Command
out PORTB, mpr ; Load the command into PORTB rjmp OPCODEEXIT

HaltFunc: rcall DecSpd rjmp OPCODEEXIT

Address$_{C}heck : cpimpr, Address; Checktoseeifwearethecorrectaddressbreq Okay_{A}ddress; Iftrueset$

Okay$_{A}ddress : ldicheck, 1; Loadaoneintocheckallowfucntiontobypasschecknexttimerjmp OPCODE$

OPCODEEXIT:

ret

;---
; Sub: HitLeft ; Desc: Handles functionality of the TekBot when
the right whisker ; is triggered. ;--
HitLeft: ; Begin a function with a label

; Save variable by pushing them to the stack push mpr push waitcnt
in mpr, SREG push mpr

in memory, PORTB ; Save old PORTB state

; Execute the function here ; Preform reverse command wait 100
ms in ilcnt, PORTB ; Get the current state ldi mpr, 0x0 andi ilcnt,
$0F; Maskthemotorstateandimpr,$F0 ; Mask the counter state or ilcnt,
mpr out PORTB, ilcnt rcall TimerWait

; Preform left command wait 100 ms in ilcnt, PORTB ; Get the current
state ldi mpr, 0x20 andi ilcnt, $0F; Maskthemotorstateandimpr,$F0 ; Mask
the counter state or ilcnt, mpr out PORTB, ilcnt rcall TimerWait

rcall QueueFix ldi mpr, $03 outEIFR, mpr$

out PORTB, memory ; Restore old PORTB state

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

ret ; End a function with RET

;-- ; Sub: HitRight ; Desc: Handles functionality of the TekBot when the right whisker ; is triggered. ;-- HitRight: ; Begin a function with a label

; Save variable by pushing them to the stack push mpr push waitcnt in mpr, SREG push mpr

in memory, PORTB ; Save old PORTB state

; Execute the function here ; Preform reverse command wait 100 ms in ilcnt, PORTB ; Get the current state ldi mpr, 0x0 andi ilcnt, $0F; Maskthemotorstateandimpr,$F0 ; Mask the counter state or ilcnt, mpr out PORTB, ilcnt rcall TimerWait

; Preform right command wait 100 ms in ilcnt, PORTB ; Get the current state ldi mpr, 0x40 andi ilcnt, $0F; Maskthemotorstateandimpr,$F0 ; Mask the counter state or ilcnt, mpr out PORTB, ilcnt rcall TimerWait

rcall QueueFix ldi mpr, $03outEIFR, mpr$

out PORTB, memory ; Restore old PORTB state

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

ret ; End a function with RET

;--- ; Func: Queue Fix Function ; Desc: This fucntions counts to around 600 micro seconds ; this is used to help avoid queue delays since we ; know our Atmega128 chip runs at 16 mhz we know each ; clock cycle will be 1 / 16 mhz and convert it to ; microseconds. Now we can take 600 value and divide ; by out result which was used to determine how many ; loops of 255 clock cycles we would need to stack ; hence the inner loop and outer loops ;--- QueueFix: push ilcnt ; Push registers onto the stack push olcnt ldi ilcnt, 255 ; Load 255 into ilcnt ldi olcnt, 30 ; Load 30 into olcnt ILOOPQUEUE: dec ilcnt ; Decrement ilcnt brne ILOOPQUEUE ; Branch if not equal to zero to ILOOPQUEUE OLOOPQUEUE: ldi ilcnt, 255 ; Load 255 into ilcnt dec olcnt ; Decrement olcnt brne ILOOPQUEUE ; Branch if not equal to zero to ILOOPQUEUE pop olcnt ; Pop registers off the stack pop ilcnt ret

;--
; Sub: TimerWait ; Desc: Handles functionality of the TekBot when the right whisker ; is triggered. ;---
TimerWait:

ldi mpr, high(3036) ; Set the starting value to 3036 sts TCNT3H, mpr ldi mpr, low(3036) sts TCNT3L, mpr ldi mpr, (1<<TOV3) ;ldi mpr, (1<<TOIE3) ; Enable the TOV interrupt ;sts ETIMSK, mpr TIMERLOOP: lds mpr, ETIFR sbrs mpr, TOV3 rjmp TIMERLOOP ldi mpr, (1<<TOV3) sts ETIFR, mpr

ret

;-- ; Func: Increment Speed ; Desc: This funciton should increment the binary output ; on PORTB(0..3) and increment OCCR0 by "step" ;--------------------------
IncSpd: ; Begin a function with a label

; If needed, save variables by pushing to the stack push mpr push ilcnt in mpr, SREG push mpr push olcnt

; Execute the function here

in ilcnt, PORTB ; Get the current state mov mpr, ilcnt ; Get a copy of current state andi ilcnt, $0F; Maskthemotorstateandimpr,$F0 ; Mask the counter state cpi ilcnt, $0F; MakesurewearentatmaxspeedbreqSKIP_ADD; Skiptheupda$

; Step the motor speed by 1/16 speed increments push mpr ; Save mpr ldi mpr, step ; Load in the step value (decimal 17) in olcnt, OCR0 ; Get the current speed add olcnt, mpr ; Increment the speed by the step value out OCR0, olcnt ; Set the new speed out OCR2, olcnt ; Set the new speed pop mpr ; Increment the speed display inc ilcnt ; Increment the Speed counter to update the led count or ilcnt, mpr out PORTB, ilcnt ; Update the LEDs

SKIP$_ADD : rcallWAITFUNCldimpr,$0F out EIFR, mpr ; Restore any saved variables by popping from stack pop olcnt pop mpr out SREG, mpr pop ilcnt pop mpr

ret ; End a function with RET

;-- ; Func: Decrement Speed ; Desc: Cut and paste this and fill in the info at the ; beginning of your functions ;--
DecSpd: ; Begin a function with a label

; If needed, save variables by pushing to the stack push mpr push ilcnt in mpr, SREG push mpr push olcnt

; Execute the function here

in ilcnt, PORTB ; Get the current Speed count mov mpr, ilcnt ; Get a copy of the current state of the output andi ilcnt, $0F; Maskthemotorstateandimpr$,FC
; Mask the counter state cpi ilcnt, $00; Makesurewearen'tatminspeedalreadybreqSKIP_ADD2; S$

; Step the motor speed by 1/16 speed increments push mpr ; Save mpr ldi mpr, step ; Load in the step value (decimal 17) in olcnt, OCR0 ; Get the current speed sub olcnt, mpr ; decrement the speed by the step value out OCR0, olcnt ; Set the new speed out OCR2, olcnt ; Set the new speed pop mpr ; Decrement the speed count display dec ilcnt ; Increment the Speed counter to update the led count or ilcnt, mpr ; Get the new state of our motors and count out PORTB, ilcnt ; Update the LEDs to display new state

SKIP$_ADD2 :; Thisisourdebouncesequence, waitandflag...rcallWAITFUNCldimpr$,0F
out EIFR, mpr

; Restore any saved variables by popping from stack pop olcnt pop mpr out SREG, mpr pop ilcnt pop mpr

ret ; End a function with 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 ;---
WAITFUNC: 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, 30 ; 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

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

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