ECE 375 Lab 8

Remotely Operated Vehicle

Lab Time: Friday 12-2

Ariel Meshorer
Jordan Brantner

TA Signature

1 Introduction

The purpose of this lab is to use our knowledge of assembly programming and the USART modules to create a proof-of-concept product for a consumer product. This lab is a cumulative product of our previous work in assembly, utilizing interrupts from multiple sources to have two AVR boards interact with each other. To achieve this, we correctly initialized the transmitter and receiver in an asynchronous fashion to facilitate communication between the IR transmitter (remote) and receiver (robot). Using the unique bot address, the robot responds only to its remote, where the remote transmits along with its selected action, sent as a 16-bit logical packet. We additionally implemented a freeze functionality in the second portion of this lab, where we each remote sense a "freeze" action which does not respond to any other commands for an allotted period of time. Working with two additional boards provided by the TA, we were able to test this freeze functionality against each other.

2 Program Overview

This program provides the basic behavior for a robot as controlled by a remote. The actions created are Move Forward and Backward, turn Left and Right, and Halt. These actions are similar to the TekBot whisker functionality; the subroutines were adapted from previous labs. Each of these subroutines are assigned particular pushbuttons, (S1: Move Forward, S2: Move Backward, S5: Turn Right, S6: Turn Left, S7: Halt, S8: Freeze) which the user selects on the remote. Once the robot receives an action from the remote, it performs the action repeatedly until prompted otherwise. The "freeze" action is an additional functionality which transmits a standalone freeze signal from one robot to all other robots (in this case we are using two robots). Here the robot that receives the freeze signal (what might be described as the opponent) halts for five seconds and does not respond to any other commands. After unfreezing, it resumes its previous action until prompted otherwise. We used the Wait routine from the previous labs to "freeze" for five seconds.

3 Internal Register Definitions and Constants

This section is divided into Transmitter and Receiver in an effort to organize the content more effectively.

3.1 Transmitter

Here we define two multi-purpose registers to hold and store the command to be transmitted.. We define three wait registers to be used within the wait subroutine. We define engine bits for enable and direction and our unique bot address. We define the unique 8-bit bot address to allow the transmitter/receiver to communicate. We define our action codes/control signals as detailed in the lab handout. We define a wait time of 10 ms and our baud value of 832 to achieve a baud rate of 2400.

3.2 Receiver

The receiver has similar definitions as the transmitter, with the addition of the freeze count register and address register. The address register checks if our unique bot address matches from the transmitter; if it does not match, no action will be taken. The freeze count register tracks the number of freeze signals; once it is incremented to three, the bot will permanently freeze.

4 Initialization Routine

This section is divided into Transmitter and Receiver in an effort to organize the content more effectively.

4.1 Transmitter

Here we initialize our stack pointer, initialize PORTD to receive user input on the push buttons and PORTB for output on the LEDs. We then initialize the USART module necessary to communicate between boards. Using the USCR1 register, we load USRC1C with the proper bits to initialize the frame; load 00 into UPM as we are not using parity; and write 1 to USBS1 as we are using two stop bits. We then load 1 into TXEN1, or the transmitter pin, to enable the transmitter functionality. We additionally declare our baud rate of 2400 bits per second by loading our baud value into UBRR1L(ow) and UBRR1H(igh).

4.2 Receiver

The receiver performs a similar initialization routine as the transmitter in terms of the stack pointer and I/O, however it enables the receiver pin RCEN1 and RXCIE to enable the receiver functionality. We additionally initialize our 16 bit timer in fast PWM mode as a counter, along with initializing the timer interrupts.

5 Main Routine

This section is divided into Transmitter and Receiver in an effort to organize the content more effectively.

5.1 Receiver

In our receiver, main infinitely loops until stopped by an interrupt.

5.2 Transmitter

The transmitter main routine is slightly more involved, where it polls using ANDI to check if the user is pushing down one button. Using ANDI with corresponding bit pattern we perform the AND operation which thus sets the Z flag. If the Z flag is true, then BREQ would branch and thus ignore if multiple buttons are pressed. This functionality polls for all of the action codes including the Freeze.

6 Subroutines

6.1 Transmitter

6.2 DeploySignal

As a precondition, the deploy signal requires the correct code you want to send out is in the MPR and pushes it to the stack. It repeatedly loads the control and status (USCR1A) register until the USART data register empty bit is set. If the bit is not empty, it goes the jumps back to the start of the routine, otherwise it skips the jump instruction and loads the bot address into UDR1 which then transmits the bot address. It repeats the same process until the data register is empty, when it is it loads the code into the UDR and gets sent out.

6.3 Receiver

6.4 USARTreceive

This function is quite large and should've been separated into smaller functions. It has four branching pathes; it first reads the first bit of the code. If the first bit is a 0, it assumes it is an address and branches to the check address. If it is a 1, it branches to perform the operation. Within check address, it checks if it is 01010101, it branches to the sub-block of code of the freeze function, otherwise it falls through and compares the code to the hardcoded control address. In the perform operation, it checks if it 1111000, if so it branches to TRANSMIT FREEZE, otherwise it falls through and shifts left one and displays on the LEDs. Within the freeze function, it turns off the reciever and disables interrupts, and stores the value of PORTB into an mpr (mpr2) and displays halt and waits for 5 seconds. It then increments the freeze counter register; once it reaches 3 it infinitely loops until it is reset. Once the wait is finished it restores the state of mpr2 onto the LEDs and turns back on the interrupts and receiver. TRANSMIT FREEZE turns off the reciever and turns on the transmitter and repeatedly checks and waits via polling UDRE1 until it is empty and once it is it sends 01010101. It then checks if TXC1 is set and then returns to main, otherwise it infinitely loops. After the four branches are complete, it returns to main.

6.5 Hit Left and Right

These are similar to the previous labs however they are implemented with interrupts.

7 Difficulties

Our main difficulties were regarding testing and the receiver code. The receiver kept freezing itself until we had it start waiting. The receiver would read what it was transmitting to itself. We additionally did not have the jumper on the board– this greatly affected testing and had us under the impression our functionality was off when it was a hardware problem in actuality.

8 Conclusion

The objectives of this lab were met and thus I can conclude this lab to be successful. From the board, we can see the correct functionality of the robot and remote. We can additionally observe the correct communication between the boards using the USART module. Therefore, I have demonstrated understanding of using USART to implement a proof-of-concept product that uses interrupts, timers, I/O ports and so on as a culmination of all of the labs.

9 Source Code

```
;***********************************************************
;*
;* This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;*  Author: Jordan Brantner and Ariel Meshorer
;*    Date: 3/8/22
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def addressReg = r17 ;stores if the code is correct or not
.def waitcnt = r18 ;wait count reg
.def ilcnt = r19 ;inner loop count reg
.def olcnt = r20 ;outer loop count reg
.def mpr2 = r21 ;second mpr
.def freezeCount = r22 ;count times it has frozen
.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 WaitTime = 100
.equ BotAddress = $64 ;(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 BaudVal = 832

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

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

.org $0002
rcall HitRight
reti
.org $0004
rcall HitLeft
reti
.org $003C
rcall USARTRecieve
reti
;Should have Interrupt vectors for:
;- Left whisker
;- Right whisker
;- USART receive

.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, $00
out DDRD, mpr ;initialize port D for input
ldi mpr, $FF
out PORTD, mpr ;enable pull up resistors
out DDRB, mpr ;initialize port B for output
;USART1
ldi mpr, (0<<UMSEL1 | 0<<UPM10 | 0<<UPM11 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C, mpr
ldi mpr, (1<<RXCIE1|1<<RXEN1|0<<UCSZ12)
sts UCSR1B, mpr
ldi mpr, (1<<U2X1) ;enable double data rate
sts UCSR1A, mpr
ldi mpr, low(BaudVal)
sts UBRR1L, mpr
ldi mpr, high(BaudVal)
sts UBRR1H, mpr
;Set baudrate at 2400bps
;Enable receiver and enable receive interrupts
;Set frame format: 8 data bits, 2 stop bits
;External Interrupts
ldi mpr, 0b00001010
sts EICRA, mpr
ldi mpr, 0b00000011
out EIMSK, mpr
ldi freezeCount, 0
sei
;Set the External Interrupt Mask
;Set the Interrupt Sense Control to falling edge detection

;Other

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
;TODO: ???
rjmp MAIN

;***********************************************************
;* Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------
; Sub: USARTRecieve
; Desc: checks if the USART signal is an address or a command. If it was
; an address, it check if it matches the hard coded address.
; if it was a command, it displays it on the board if the last
; address was the matching address
;-----------------------------------------------------------
USARTRecieve:
push mpr
```

```
lds mpr, UDR1
sbrs mpr, 7 ;if it starts with a 0, then check if valid address
rjmp CHECK_ADDRESS ;else, perform an operation
rjmp PERFORM_OP

PERFORM_OP:
cpi addressReg, 1 ;LSL if the signal left, then display on lights
brne SKIP1
cpi mpr, 0b11111000
breq TRANSMIT_FREEZE
lsl mpr
out PORTB, mpr
rjmp SKIP

CHECK_ADDRESS:
cpi mpr, 0b01010101
breq FREEZE
cpi mpr, BotAddress ;check if address matches
ldi addressReg, 0 ;set boolean addressreg to 0 if it doesnt
brne SKIP
ldi addressReg, 1
SKIP1:
rjmp SKIP

FREEZE:
lds mpr, UCSR1B
andi  mpr, 0b11101111 ;disable reception
sts UCSR1B, mpr
ldi mpr, 0b00000000 ;disable interrupts
out EIMSK, mpr

in mpr2, PORTB

ldi mpr, 0b11110000 ;display halt signal
out PORTB, mpr
ldi waitcnt, WaitTime ;wait for 5 seconds
rcall Lab1Wait
rcall Lab1Wait
rcall Lab1Wait
rcall Lab1Wait
rcall Lab1Wait

inc freezeCount ;check if frozen 3 times
FULLSTOP:
cpi freezeCount, 3
```

```
breq FULLSTOP ;if frozen 3 times, infinite loop until reset

out PORTB, mpr2

ldi mpr, 0b00000011 ;reenable interrupts
out EIMSK, mpr
lds mpr, UCSR1B
ori  mpr, 0b00010000 ;reenable reception
sts UCSR1B, mpr
rjmp SKIP

TRANSMIT_FREEZE:
;set to transmitter, turn off reciever
ldi mpr, (0<<TXCIE1|1<<TXEN1|0<<UDRIE1|0<<UCSZ12|0<<RXEN1)
sts UCSR1B, mpr

DeploySignal:
lds mpr, UCSR1A ;check if transmit is available
sbrs mpr, UDRE1
rjmp DeploySignal ;if not, go back

;transmit bot code
ldi mpr, 0b01010101 ;send freeze code to usart
sts UDR1, mpr

DoneSending:
lds mpr, UCSR1A ;check if transmit is available
sbrs mpr, TXC1
rjmp DoneSending ;if not, go back

ldi waitcnt, 10
rcall Lab1Wait

;set back to recieve
ldi mpr, (1<<RXCIE1|1<<RXEN1|0<<UCSZ12|0<<TXEN1)
sts UCSR1B, mpr
rjmp SKIP
SKIP:
pop mpr
ret

;----------------------------------------------------------------
; Sub: Lab1Wait
; 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
;-----------------------------------------------------------------
Lab1Wait:
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

;-----------------------------------------------------------------
; Sub: HitLeft
; Desc: moves back for 1 second, turns right for 1 second, then resume
; previous motion all while ignoring usart interrupts
;-----------------------------------------------------------------
HitLeft:
;get previous state
push mpr
push mpr2

lds mpr, UCSR1B
andi  mpr, 0b11101111 ;disable reception
sts UCSR1B, mpr
in mpr2, PORTB
; Move Backwards for a second
ldi mpr, 0b00000000 ; Move Backward
out PORTB, mpr
ldi waitcnt, WaitTime ; Wait for 1 second
rcall Lab1Wait ; Call wait function

; Turn left for a second
ldi mpr, 0b01000000 ; Turn Right
out PORTB, mpr
ldi waitcnt, WaitTime ; Wait for 1 second
```

```
rcall Lab1Wait ; Call wait function

; Resume previous motion
out PORTB, mpr2
ldi mpr, $FF
out EIFR, mpr ;clear latched interupts

lds mpr, UCSR1B
ori  mpr, 0b00010000 ;reenable reception
sts UCSR1B, mpr

pop mpr2
pop mpr
ret

;-----------------------------------------------------------------
; Sub: HitRight
; Desc: moves back for 1 second, turns left for 1 second, then resume
; previous motion all while ignoring usart interrupts
;-----------------------------------------------------------------
HitRight:

push mpr
push mpr2

lds mpr, UCSR1B
andi  mpr, 0b11101111 ;disable reception
sts UCSR1B, mpr
in mpr2, PORTB ;get previous state

; Move Backwards for a second
ldi mpr, 0b00000000 ; Move Backward
out PORTB, mpr
ldi waitcnt, WaitTime ; Wait for 1 second
rcall Lab1Wait ; Call wait function

; Turn left for a second
ldi mpr, 0b00100000 ; Turn Left
out PORTB, mpr
ldi waitcnt, WaitTime ; Wait for 1 second
rcall Lab1Wait ; Call wait function

; Resume previous motion
out PORTB, mpr2
ldi mpr, $FF
out EIFR, mpr ;clear latched interupts
```

```
lds mpr, UCSR1B
ori  mpr, 0b00010000 ;reenable reception
sts UCSR1B, mpr

pop mpr2
pop mpr
ret

;***********************************************************
;*
;* This is the TRANSMIT skeleton file for Lab 8 of ECE 375
;*
;*  Author: Jordan Brantner and Ariel Meshorer
;*    Date: 3/8/22
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def mpr2 = r17
.def waitcnt = r18 ;wait count reg
.def ilcnt = r19 ;inner loop count reg
.def olcnt = r20 ;outer loop count reg

.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 = $64;(Enter your robot's address here (8 bits))
; 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 WaitTime = 1
.equ BaudVal = 832
;***********************************************************
```

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

;***********************************************************
;* Interrupt Vectors
;***********************************************************
.org $0000 ; Beginning of IVs
rjmp  INIT ; Reset interrupt
.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Stack Pointer (VERY IMPORTANT!!!!)
ldi mpr, low(RAMEND)
out SPL, mpr
ldi mpr, high(RAMEND)
out SPH, mpr
;I/O Ports
ldi mpr, $00
out DDRD, mpr ;initialize port D for input
ldi mpr, $FF
out PORTD, mpr ;enable pull up resistors
out DDRB, mpr ;initialize port B for output
;USART1
ldi mpr, (0<<UMSEL1 | 0<<UPM10 | 0<<UPM11 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C, mpr
ldi mpr, (0<<TXCIE1|1<<TXEN1|0<<UDRIE1|0<<UCSZ12)
sts UCSR1B, mpr
ldi mpr, (1<<U2X1) ;enable double data rate
sts UCSR1A, mpr
ldi mpr, low(BaudVal)
sts UBRR1L, mpr
ldi mpr, high(BaudVal)
sts UBRR1H, mpr
;Set baudrate at 2400bps
;Enable receiver and enable receive interrupts
;Set frame format: 8 data bits, 2 stop bits
;External Interrupts
;Set the External Interrupt Mask
;Set the Interrupt Sense Control to falling edge detection

;Other
ldi mpr, 1
```

```
;***********************************************************
;* Main Program
;***********************************************************
MAIN:
in mpr, PIND

mov mpr2, mpr ;get PIND
andi mpr2, 0b00000001 ;check if forwards
breq MOVE_FORWARD ;if so, jump to forwards branch

mov mpr2, mpr
andi mpr2, 0b00000010 ;poll for backwards
breq MOVE_BACKWARDS

mov mpr2, mpr
andi mpr2, 0b01000000 ;poll for halt
breq STOP

mov mpr2, mpr
andi mpr2, 0b10000000 ;poll for freeze
breq FREEZE

mov mpr2, mpr
andi mpr2, 0b00100000 ;poll for turn left
breq TURN_LEFT

mov mpr2, mpr
andi mpr2, 0b00010000 ;poll for right
breq TURN_RIGHT

rjmp MAIN

MOVE_FORWARD:
ldi mpr, 0b10110000 ;load mpr with respective code
rcall DeploySignal ;call deploy signal to send code
rjmp MAIN ;resume polling
MOVE_BACKWARDS:
ldi mpr, 0b10000000 ;deploy back signal
rcall DeploySignal
rjmp MAIN
TURN_RIGHT:
ldi mpr, 0b10100000 ;deploy right signal
rcall DeploySignal
rjmp MAIN
TURN_LEFT:
```

```
ldi mpr, 0b10010000 ;deploy left signal
rcall DeploySignal
rjmp MAIN
FREEZE:
ldi mpr, 0b11111000 ;deploy freeze signal
rcall DeploySignal
rjmp MAIN
STOP:
ldi mpr, 0b11001000 ;deploy stop signal
rcall DeploySignal
rjmp MAIN



;***********************************************************
;* Functions and Subroutines
;***********************************************************
;Deploy Signal outputs the bot address out to USART, and then
;outputs the signal stored in mpr to the USARt to create a full
;16 bit packet
DeploySignal:
push mpr
push mpr2
Retry:
lds mpr2, UCSR1A ;check if transmit is available
sbrs mpr2, UDRE1
rjmp Retry ;if not, go back

;transmit bot code
ldi mpr2, BotAddress ;send Bot code to usard
sts UDR1, mpr2

ldi waitcnt, WaitTime ;wait 10 ms to separate bot address and data packets
rcall Lab1Wait

TRANSMIT_INSTRUCTION:
lds mpr2, UCSR1A ;check if bot code has been sent
sbrs mpr2, UDRE1
rjmp TRANSMIT_INSTRUCTION
sts UDR1, mpr ;send 8 bit command to usart

pop mpr2
pop mpr
ret

;-----------------------------------------------------------------
```

```
; Sub: Lab1Wait
; 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
;-----------------------------------------------------------------
Lab1Wait:
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
```

10 Challenge Code

```
    ;***********************************************************
;*
;* This is the TRANSMIT skeleton file for Lab 8 of ECE 375
;*
;*  Author: Jordan Brantner and Ariel Meshorer
;*    Date: 3/8/22
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def mpr2 = r17
.def waitcnt = r18 ;wait count reg
```

```
.def ilcnt = r19 ;inner loop count reg
.def olcnt = r20 ;outer loop count reg

.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 = $64;(Enter your robot's address here (8 bits))
; 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 WaitTime = 1
.equ BaudVal = 832
;***********************************************************
;* Start of Code Segment
;***********************************************************
.cseg ; Beginning of code segment

;***********************************************************
;* Interrupt Vectors
;***********************************************************
.org $0000 ; Beginning of IVs
rjmp  INIT ; Reset interrupt
.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Stack Pointer (VERY IMPORTANT!!!!)
ldi mpr, low(RAMEND)
out SPL, mpr
ldi mpr, high(RAMEND)
out SPH, mpr
;I/O Ports
ldi mpr, $00
out DDRD, mpr ;initialize port D for input
ldi mpr, $FF
out PORTD, mpr ;enable pull up resistors
out DDRB, mpr ;initialize port B for output
```

```
;USART1
ldi mpr, (0<<UMSEL1 | 0<<UPM10 | 0<<UPM11 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C, mpr
ldi mpr, (0<<TXCIE1|1<<TXEN1|0<<UDRIE1|0<<UCSZ12)
sts UCSR1B, mpr
ldi mpr, (1<<U2X1) ;enable double data rate
sts UCSR1A, mpr
ldi mpr, low(BaudVal)
sts UBRR1L, mpr
ldi mpr, high(BaudVal)
sts UBRR1H, mpr
;Set baudrate at 2400bps
;Enable receiver and enable receive interrupts
;Set frame format: 8 data bits, 2 stop bits
;External Interrupts
;Set the External Interrupt Mask
;Set the Interrupt Sense Control to falling edge detection

;Other
ldi mpr, 1

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
in mpr, PIND

mov mpr2, mpr ;get PIND
andi mpr2, 0b00000001 ;check if forwards
breq MOVE_FORWARD ;if so, jump to forwards branch

mov mpr2, mpr
andi mpr2, 0b00000010 ;poll for backwards
breq MOVE_BACKWARDS

mov mpr2, mpr
andi mpr2, 0b01000000 ;poll for halt
breq SPEED_UP

mov mpr2, mpr
andi mpr2, 0b10000000 ;poll for freeze
breq SPEED_DOWN

mov mpr2, mpr
andi mpr2, 0b00100000 ;poll for turn left
breq TURN_LEFT
```

```
mov mpr2, mpr
andi mpr2, 0b00010000 ;poll for right
breq TURN_RIGHT

rjmp MAIN

MOVE_FORWARD:
ldi mpr, 0b10110000 ;load mpr with respective code
rcall DeploySignal ;call deploy signal to send code
rjmp MAIN ;resume polling
MOVE_BACKWARDS:
ldi mpr, 0b10000000 ;deploy back signal
rcall DeploySignal
rjmp MAIN
TURN_RIGHT:
ldi mpr, 0b10100000 ;deploy right signal
rcall DeploySignal
rjmp MAIN
TURN_LEFT:
ldi mpr, 0b10010000 ;deploy left signal
rcall DeploySignal
rjmp MAIN
SPEED_DOWN:
ldi mpr, 0b11111000 ;deploy freeze signal
rcall DeploySignal
ldi waitcnt, 25
rcall Lab1Wait
rjmp MAIN
SPEED_UP:
ldi mpr, 0b11001000 ;deploy stop signal
rcall DeploySignal
ldi waitcnt, 25
rcall Lab1Wait
rjmp MAIN



;***********************************************************
;* Functions and Subroutines
;***********************************************************
;Deploy Signal outputs the bot address out to USART, and then
;outputs the signal stored in mpr to the USARt to create a full
;16 bit packet
DeploySignal:
push mpr
push mpr2

lds mpr2, UCSR1A ;check if transmit is available
sbrs mpr2, UDRE1
rjmp DeploySignal ;if not, go back

;transmit bot code
ldi mpr2, BotAddress ;send Bot code to usard
sts UDR1, mpr2

ldi waitcnt, 1
rcall Lab1Wait

TRANSMIT_INSTRUCTION:
lds mpr2, UCSR1A ;check if bot code has been sent
sbrs mpr2, UDRE1
rjmp TRANSMIT_INSTRUCTION
sts UDR1, mpr ;send 8 bit command to usart



pop mpr2
pop mpr
ret

;----------------------------------------------------------------
; Sub: Lab1Wait
; 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
;----------------------------------------------------------------
Lab1Wait:
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

;***********************************************************
;*
;* This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;*  Author: Jordan Brantner and Ariel Meshorer
;*    Date: 3/8/22
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def addressReg = r17 ;stores if the code is correct or not
.def waitcnt = r18 ;wait count reg
.def ilcnt = r19 ;inner loop count reg
.def olcnt = r20 ;outer loop count reg
.def mpr2 = r21 ;second mpr
.def freezeCount = r22 ;count times it has frozen
.def speed = r23 ; holds speed
.def doneWaiting = r24 ; store if waiting is done
.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 WaitTime = 100
.equ BotAddress = $64 ;(Enter your robot's address here (8 bits))
.equ T1Val = $0BDD ;timer value to wait 1 second

;////////////////////////////////////////////////////////////
;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 BaudVal = 832

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

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

.org $003C
rcall USARTRecieve
reti
.org $001C
rcall secondElapsed
reti
;Should have Interrupt vectors for:
;- Left whisker
;- Right whisker
;- USART receive

.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, $00
out DDRD, mpr ;initialize port D for input
ldi mpr, $FF
out PORTD, mpr ;enable pull up resistors
out DDRB, mpr ;initialize port B for output
;USART1
ldi mpr, (0<<UMSEL1 | 0<<UPM10 | 0<<UPM11 | 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10)
sts UCSR1C, mpr
```

```
ldi mpr, (1<<RXCIE1|1<<RXEN1|0<<UCSZ12)
sts UCSR1B, mpr
ldi mpr, (1<<U2X1) ;enable double data rate
sts UCSR1A, mpr
ldi mpr, low(BaudVal)
sts UBRR1L, mpr
ldi mpr, high(BaudVal)
sts UBRR1H, mpr
;Set baudrate at 2400bps
;Enable receiver and enable receive interrupts
;Set frame format: 8 data bits, 2 stop bits
;timer
ldi mpr, (1<<WGM00 | 1<<WGM01 | 1<<COM01| 1<<COM00 | 1<<CS02 | 0<<CS01 | 0<<CS00)
    out TCCR0, mpr    ;fast PWM, inverse mode, 64 prescale
    ldi mpr, (1<<WGM20 | 1<<WGM21 | 1<<COM21| 1<<COM20 | 1<<CS22 | 0<<CS21 | 0<<CS20)
    out TCCR2, mpr
    ldi mpr, 0
out OCR2, mpr
    out OCR0, mpr
;enable 16 bit counter
    ldi mpr, $00
    out TCCR1A, mpr    ;normal operation on all 3 channels
    ldi mpr, 0b00000100    ;256 prescale, normal mode
    out TCCR1B, mpr
    ldi mpr, 0b00000000    ;disable TOIE interrupt initially
    out TIMSK, mpr
ldi freezeCount, 0
ldi speed, 0
sei
;Set the External Interrupt Mask
;Set the Interrupt Sense Control to falling edge detection

;Other

;************************************************************
;* Main Program
;************************************************************
MAIN:
;write speed to display
ldi mpr, 17
mul mpr, speed ;get waveform for corresponding speed
out OCR2, r0 ;set to duty cycle
out OCR0, r0

in mpr, PORTB ;get current value of OCR and state
cbr mpr, $0F ;clear bottom nibble, wrtie speed
```

```
or mpr, speed
out PORTB, mpr ;reoutput

in mpr, PIND

sbrs mpr, 0 ;if hit right whisker is not cleared, call hit right
rcall HitRight ;if so, call it

sbrs mpr, 1 ;if hit left whisker is not cleared, call hit right
rcall HitLeft ;if so, call it

rjmp MAIN

;***********************************************************
;* Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------------
; Sub: USARTRecieve
; Desc: checks if the USART signal is an address or a command. If it was
; an address, it check if it matches the hard coded address.
; if it was a command, it displays it on the board if the last
; address was the matching address
;-----------------------------------------------------------------
USARTRecieve:
push mpr

lds mpr, UDR1
sbrs mpr, 7 ;if it starts with a 0, then check if valid address
rjmp CHECK_ADDRESS ;else, perform an operation
rjmp PERFORM_OP

PERFORM_OP:
cpi addressReg, 1 ;LSL if the signal left, then display on lights
brne SKIP1
cpi mpr, 0b11001000 ;if speed up command, execute it
breq SpeedUp
cpi mpr, 0b11111000 ;if speed down, execute it
breq SpeedDown
lsl mpr ;otherwise, lsl signal and display it
out PORTB, mpr
rjmp SKIP

SpeedUp:
cpi speed, 15 ;if speed isnt at max, increment
breq SKIP
inc speed
```

```
rjmp SKIP

SpeedDown:
cpi speed, 0 ;if speed isnt at min, decrement
breq SKIP
dec speed
rjmp SKIP

CHECK_ADDRESS:
cpi mpr, 0b01010101
breq FREEZE
cpi mpr, BotAddress ;check if address matches
ldi addressReg, 0 ;set boolean addressreg to 0 if it doesnt
brne SKIP
ldi addressReg, 1
SKIP1:
rjmp SKIP

FREEZE:
lds mpr, UCSR1B
andi  mpr, 0b11101111 ;disable reception
sts UCSR1B, mpr
ldi mpr, 0b00000000 ;disable interrupts
out EIMSK, mpr

in mpr2, PORTB

ldi mpr, 0b10010000 ;display halt signal
out PORTB, mpr
ldi waitcnt, WaitTime ;wait for 5 seconds
rcall Lab1Wait
rcall Lab1Wait
rcall Lab1Wait
rcall Lab1Wait
rcall Lab1Wait

inc freezeCount ;check if frozen 3 times
FULLSTOP:
cpi freezeCount, 3 ;if frozen 3 times, infinite loop until reset
breq FULLSTOP

out PORTB, mpr2

ldi mpr, 0b00000011 ;reenable interrupts
out EIMSK, mpr
lds mpr, UCSR1B
```

```
ori  mpr, 0b00010000 ;reenable reception
sts UCSR1B, mpr
rjmp SKIP

SKIP:
pop mpr
ret

;-----------------------------------------------------------------
; Sub: secondElapsed
; Desc: sets the done waiting register to 1
;-----------------------------------------------------------------
secondElapsed:
ldi doneWaiting, 1
ldi mpr, $FF ;clear latched interrupts
out TIFR, mpr
ret

;-----------------------------------------------------------------
; Sub: HitLeft
; Desc: moves back for 1 second, turns right for 1 second, then resume
; previous motion all while ignoring usart interrupts
;-----------------------------------------------------------------
HitLeft:

;get previous state
push mpr
push mpr2

lds mpr, UCSR1B
andi  mpr, 0b11101111 ;disable reception
sts UCSR1B, mpr

; Move Backwards for a second
in mpr2, PORTB
out PORTB, speed
rcall IntWait

; Turn left for a second
ldi mpr, 0b01000000 ; Turn Right
or mpr, speed
out PORTB, mpr
rcall IntWait

; Resume previous motion
out PORTB, mpr2
```

```
lds mpr, UCSR1B
ori  mpr, 0b00010000 ;reenable reception
sts UCSR1B, mpr

pop mpr2
pop mpr
ret

;-----------------------------------------------------------------
; Sub: HitRight
; Desc: moves back for 1 second, turns left for 1 second, then resume
; previous motion all while ignoring usart interrupts
;-----------------------------------------------------------------
HitRight:
push mpr
push mpr2

lds mpr, UCSR1B
andi  mpr, 0b11101111 ;disable reception
sts UCSR1B, mpr
in mpr2, PORTB ;get previous state

; Move Backwards for a second Move Backward
out PORTB, speed
rcall IntWait

; Turn left for a second
ldi mpr, 0b00100000 ; Turn Left
or mpr, speed
out PORTB, mpr
rcall IntWait

; Resume previous motion
out PORTB, mpr2

lds mpr, UCSR1B
ori  mpr, 0b00010000 ;reenable reception
sts UCSR1B, mpr

pop mpr2
pop mpr
ret

;-----------------------------------------------------------------
; Sub: IntWait
; Desc: uses interrupts to wait 1 second
;-----------------------------------------------------------------
IntWait:
push mpr

ldi mpr, 0b00000000 ;disable timer
out TIMSK, mpr
ldi mpr, $FF ;clear latched interrupts
out TIFR, mpr
cli

ldi mpr, high(T1Val)    ;load 1 sec delay into TCNT
        out TCNT1H, mpr
        ldi mpr, low(T1Val)
        out TCNT1L, mpr

clr doneWaiting ;set done waiting flag to 0
sei
ldi mpr, 0b00000100 ;enable timer interrupt
out TIMSK, mpr
WAIT:
cpi doneWaiting, 1 ;wait until interrupt triggers
brne WAIT

ldi mpr, 0b00000000 ;disable timer again
out TIMSK, mpr

pop mpr
ret
;-----------------------------------------------------------------
; Sub: Lab1Wait
; 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
;-----------------------------------------------------------------
Lab1Wait:
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
```