ECE 375 LAB 8

Remotely Operated Vehicle

Lab Time: Wednesday 5-7pm

Rhea Mae Edwards

Jack Neff

INTRODUCTION

The purpose of this final lab is to provide an introduction to the USART modules and functionality on the mega128 boards. This is implemented with the idea of building the concept of a simple robot that is controlled be an infrared (IR) remote, by the use of a couple of mega128 boards: one used as the remote and the other used as the robot. The concept is even taken further, by having the ability to have two robots be able to play freeze tag with one another still with the control of a remote.

PROGRAM OVERVIEW

PART 1 – REMOTE CONTROL OVERVIEW:

The proof-of-concept robot is implemented with various types of actions. These actions include, moving forward, moving backwards, turning right, turning left, and halting, which are chosen by the user by pressing buttons on the remote that is controlling the actions of the robot. The concept is that the robot also continues performing that same action until a different action is received, without needing to receive the same action repeatedly from the remote. Each robot has its own distinct remote control, based on its own distinct address.

These actions are sent from the remote to the robot as a 16-bit logical packet, which consists of two 8-bit values, which are sent back-to-back by the remote's USART module. The first 8-bit value being the robot address byte (having a MSB of 0), that indicates the specific robot the packet is intended for, and the second 8-bit value being the action code byte (having a MSB of 1), that indicates the action the user wants the robot to take. These packets are also transmitted at a baud rate of 2400 bits per second.

Byte 1: Robot Address	Byte 2: Action Code
0 X X X X X X X	1 X X X X X X X

Table 1: Packet Structure for Remote-to-Robot Communication

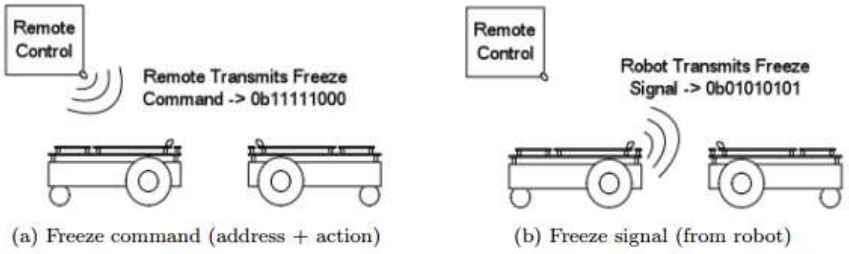
Robot Action	Action Code
Move Forward	0b10110000
Move Backward	0b10000000
Turn Right	0b10100000
Turn Left	0b10010000
Halt	0b11001000
Future Use	0b11111000

Table 2: Action Codes

The robot also performs the BumpBot behavior. Spceifically, when either of the whiskers are triggered, the robot reverses for 1 second, and then turns away from the point of contact for 1 second, all while ignoring any commands from the remote control. After the robot has finished reversing and turning, it goes back to whatever action is was doing before the impact, and then resumes listening for packets sent from its remote.

PART 2 – FREEZE TAG OVERVIEW:

In order to have two robots play freeze tag with one another, we implemented a sixth action to the remotes that freezes its robots. The freeze actions is defined as the binary number of 0b11111000, and is transmitted using the same packet structure as in the previous part, by sending the robot address first, and then the freeze action code (depicted in Figure 1a). Then the robot immediately transmits a standalone 8-bit freeze signal, defined as the binary number 0b01010101, which is sent directly without any address byte sent first. This is a transmission from one robot to all other nearby robots (depicted in Figure 1b).



(a) Freeze command (address + action) (b) Freeze signal (from robot)

Figure 1: Freeze command and freeze signal transmission

Any robot that receives an 8-bit freeze signal freezes for five seconds, except for the robot that just sent the freeze signal itself. Also, freezing means that the robot with halt, not responses to the whiskers, commands, or any other freeze signals. When a robot unfreezes, it resumes to what it was doing before it was frozen. But after being frozen for the third time, the robot stops working (meaning it stays frozen) until it is reset.

REMOTE CONTROL (TRANSMIT) SUBROUTINE EXPLANATIONS:

Besides the standard INIT and MAIN routines within the program, 11 additional routines were created and used. The functions CheckBack, Checkleft, Checkright, Checkfreeze, and Checktop are all polling function that read the input from PORTD and decide what function to run. They call the functions MoveForward, MoveBackward, Turnleft, Turnright, Stop, and Freeze.

INITIALIZATION ROUTINE

The initialization routine initializes the stack pointer, PORTD for input, PORTB for output. We also set the U2X1 flag on UCSR1A to double the baud rate, and place the calculated value of 832 in the Baud register. This will give us a Baud Rate of 2400 combined with the 2X flag. We also enable the TXEN transmit bit on UCSR1B and set up the correct frame format and mode in UCSR1C by enabling asynchronous mode (UMSEL1 = 0), giving ourselves 2 stop bits (USBS = 1), and 8 data bits (UCSZ10: UCSZ11 = 1).

MAIN ROUTINE

The Main routine combines with the check routines for a polling operation. It checks if the pin input is for forward motion, and if it is, moves the bot forward. If not, the program flow goes to CheckBack, and if it is not the input for backing up, goes to CheckLeft and so on until the input matches that of one of the checks. This is a polling method.

CHECKBACK ROUTINE

The CheckBack routine is the same thing as main, essentially, but looking for a different input bit pattern. If the input is for backwards motion (pin 1 pressed), then MoveBackward is executed, otherwise flow continues to the next check. If MoveBackward is called, program executes it and then returns to main.

CHECKLEFT ROUTINE

The Checkleft routine is similar to above functions. It checks for a match, if it gets one, calls Turnleft. Otherwise flow goes to Checkright.

CHECKRIGHT ROUTINE

The Checkright routine does more polling. It calls Turnright on the correct input.

CHECKFREEZE ROUTINE

The Checkfreeze routine does more polling. It can call Freeze on a match.

CHECKSTOP ROUTINE

The Checkstop routine does the same thing, calls stop on a match.

MOVEFORWARD ROUTINE

The MoveForward routine outputs to LEDs the pattern of the input, so we know what was sent. Then transmits the Bot Address and then a byte with appropriate bits to make the bot go forward.

MOVEBACKWARD ROUTINE

The MoveBackward routine also outputs the input to LEDs, then transmits bot address and then bits to make the bot move back.

TURNLEFT ROUTINE

The Turnleft routine does the same thing, but transmits a turn left signal as well as bot address.

TURNRIGHT ROUTINE

The Turnright routine transmits a turn right signal and the bot address.

STOP ROUTINE

The Stop routine transmits a stop signal with the bot address.

FREEZE ROUTINE

The Freeze routine transmits a freeze signal, does not send an address.

ROBOT (RECEIVE) SUBROUTINE EXPLANATIONS:

Besides the standard INIT and MAIN routines within the program, 20 additional routines were created and used, consisting of Receive, Testflag, Run, Setflag, Moveforward, Movebackward, Turnright, Turnleft, ReceiveFreeze, Frozen, Freeze, Dead, HitLeft, HitRight, Wait, Loop, OLoop, and ILoop.

INITIALIZATION ROUTINE

The initialization routine initializes the stack pointer, the I/O ports, and the USART registers. The Baud rate is matched with the transmit function at 2400 by loading 832 into UBBR and setting the U2X1 flag on UCSR1A. So that the bot can receive and responds to interrupts, we enable the RXEN1 and RXCIE1 on UCSR1B. We also enable TXEN1 so we can transmit a freeze signal. We set the data frame as before, and also set asynchronous mode in

USCR1C. We enable external interrupts on 0 and 1 pins in EIMSK, and set EICRA to detect falling edges and throw interrupts.

Interrupt Vectors

We have HitRight and HitLeft interrupts for bumpbot activity, and more importantly, an interrupts that calls the Receive subroutine with data is received in the receive buffer. This routine will handle the values received and call the appropriate functions.

Main Routine

The Main routine does nothing. This is because we are waiting for interrupts and choose to handle then with another function, other than main.

Receive Routine

The Receive routine gets the data from UDR1.

Testflag Routine

The Testflag routine checks if a flag, held in r21, is set. The flag is set if an address is sent and matches the address of the bot. ($2A). The flag prevents the bot that sends the freeze signal from being frozen.

Run Routine

The Run routine is basically a polling routine, and it only activates when the flag in r21 is set and an action signal is received. It compares the data received in UDR1 to several bit patterns that correspond to subroutines. At the end, it returns from interrupt, so it ends the processes that execute after receiving data.

Setflag Routine

The Setflag routine sets a flag if the data sent is the correct address. Since the address is sent first, the flag is set by the correct address and then the next transmission can be executed.

Moveforward Routine

The Moveforward routine is called from run. Sets the bot to move forward, outputs this to PORTB, and then clears the flag in r21.

Movebackward Routine

The Movebackward routine is also called from run. Makes the bot indicate backward motion and then clears flag.

Turnright Routine

The Turnright routine is the same as the above two, but makes the bot turn right.

Turnleft Routine

The Turnleft routine is also the same, but makes the bot turn left.

ReceiveFreeze Routine

The ReceiveFreeze routine is called from run. Rather than outputting to the LEDs, this function loads a freeze signal into UDR1 and transmits it.

Frozen Routine

The Frozen routine freezes the robot, as long as the r21 flag is not set. So if the correct address has not been provided beforehand (which it won't be), the robot freezes.

Freeze Routine

The Freeze routine first backs up the EIMSK and UCSR1B registers, then writes them to 0 to prevent interrupts or data sending/receiving when frozen. The bot then waits for a while and restores EIMSK and UCSR1B. If the robot has been frozen 3 times (freeze has a counter), then it branches to Dead, where EIMSK and UCSR1B and permanently zeroed out. (Until reset).

Dead Routine

The Dead routine zeroes the EIMSK and UCSR1B registers, effectively shutting down the bot.

HitLeft and HitRight Routines

The HitLeft and HitRight routines implement the bumpbot behavior from previous labs. They are run on interrupts triggered by pin0 and pin1 on PORTD.

Wait Routine

The Wait routine is just a loop to make the robot wait for a few seconds, which is executed when the robot is frozen.

Loop, OLoop, and ILoop Routines

The Loop, OLoop, and ILoop routines are for the wait function. They look a large number of times, eating up clock cycles and making the robot wait.

Difficulties

We had quite a bit of difficulties in this lab, including getting the bot to transmit, receive, and display the appropriate output. We didn't know how to set the Baud rate at first, until we read the datasheet and saw that clock speed is 16 Mhz, then we calculated it with formulas on the datasheet. The hardest part was the freeze function, and getting the robot to only respond when the addresses matched.

Conclusion

We learned a lot from this lab about USART and Baud Rate. This pushed the limits of what we thought tekbots were capable of doing. Now I think we both appreciate the import and utility of embedded systems a lot more.

SOURCE CODE

REMOTE CONTROL (TRANSMIT) PROGRAM:

```
;***********************************************************
;*
;*      AssemblerApplication8Transmitbackup.asm
;*
;*      This is the TRANSMIT file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*       Author: Jack Neff, Rhea Mae Edwards
;*         Date: 3/14/2017
;*
;***********************************************************

.include "m128def.inc"               ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                               ; Multi-Purpose 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
.equ    Frze = 0b11111000
.equ    BotAddress = $2A

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

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

.org    $0046                                   ; End of Interrupt Vectors

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

        ;I/O Ports
        ldi mpr, 0b00000000
        out DDRD, mpr
        ldi mpr, 0b11111111
        out PORTD, mpr
```

```
        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

        ;USART1
        ;Set baudrate at 2400bps

        ldi mpr, (1<<U2X1)
        sts UCSR1A, mpr
        ldi mpr, high(832)
        sts UBRR1H, mpr
        ldi mpr, low(832)
        sts UBRR1L, mpr

        ;Enable transmitter
        ldi mpr, (1<<TXEN1)
        sts UCSR1B, mpr

        ;Set frame format: 8 data bits, 2 stop bits
        ldi mpr, (0<<UMSEL1|1<<USBS1|1<<UCSZ11|1<<UCSZ10)
        sts UCSR1C, mpr

;***********************************************************
;*      Main Program
;***********************************************************
MAIN:                                   ;Main subroutine which polls PORTD for input
                in mpr, PIND            ;Reads input from PORTD and compares to each possible
                                        ;command and calls the subroutine accordingly
                cpi mpr, 0b11111110
                brne Checkback
                rcall MoveForward
                jmp MAIN

CheckBack:
                cpi mpr, 0b11111101     ;All these subroutines do the same thing which is basically
                                        ;call their respective routine when
                brne Checkleft          ;Their command is read
                rcall MoveBackward
                rjmp MAIN

Checkleft:
                cpi mpr, 0b11101111
                brne Checkright
                rcall Turnleft
                rjmp MAIN

Checkright:
                cpi mpr, 0b11011111
                brne Checkfreeze
                rcall Turnright
                rjmp MAIN

Checkfreeze:
                cpi mpr, 0b01111111
                brne Checkstop
                rcall Freeze
                rjmp MAIN

Checkstop:
                cpi mpr, 0b10111111
                brne MAIN
                rcall Stop
                rjmp MAIN

;***********************************************************
;*      Functions and Subroutines
;***********************************************************
MoveForward:                            ;Move Forward subroutine
        out PORTB, mpr                  ;Outputs to LED to display button press
```

```
        ldi mpr, BotAddress           ;Loads Address to transmit
        sts UDR1, mpr                 ;Transmits
        ldi mpr, MovFwd               ;Loads MovFwd
        sts UDR1, mpr                 ;Transmits
        ret

MoveBackward:
        out PORTB, mpr                ;The rest of the subroutines are basically the same as
MoveForward
        ldi mpr, BotAddress           ;except they load their own commands into UDR1
        sts UDR1, mpr
        ldi mpr, MovBck
        sts UDR1, mpr
        ret

Turnleft:
        out PORTB, mpr
        ldi mpr, BotAddress
        sts UDR1, mpr
        ldi mpr, TurnL
        sts UDR1, mpr
        ret

Turnright:
        out PORTB, mpr
        ldi mpr, BotAddress
        sts UDR1, mpr
        ldi mpr, TurnR
        sts UDR1, mpr
        ret

Stop:
        out PORTB, mpr
        ldi mpr, BotAddress
        sts UDR1, mpr
        ldi mpr, Halt
        sts UDR1, mpr
        ret

Freeze:
        out PORTB, mpr
        ldi mpr, BotAddress
        sts UDR1, mpr
        ldi mpr, Frze
        sts UDR1, mpr
        ret

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

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

ROBOT (RECEIVE) PROGRAM:

```
;***********************************************************
;*
;*      AssemblerApplication8.asm
;*
;*      This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*       Author: Jack Neff, Rhea Mae Edwards
;*         Date: 3/14/2017
;*
;***********************************************************

.include "m128def.inc"               ; Include definition file

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

.equ    WTime = 100                             ; Wait time
.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 = $2A                        ;(Enter your robot's address here (8 bits))

;/////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
;/////////////////////////////////////////////////////////////

.equ    MovFwd =  (1<<EngDirR|1<<EngDirL)            ;0b01100000 Move Forward Action Code
.equ    MovBck =  $00                                ;0b00000000 Move Backward Action Code
.equ    TurnR =   (1<<EngDirL)                       ;0b01000000 Turn Right Action Code
.equ    TurnL =   (1<<EngDirR)                       ;0b00100000 Turn Left Action Code
.equ    Halt =    (1<<EngEnR|1<<EngEnL)              ;0b10010000 Halt Action Code

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

;***********************************************************
;*      Interrupt Vectors
;***********************************************************
.org    $0000                           ; Beginning of IVs
              rjmp    INIT              ; Reset interrupt
.org    $0002
              rcall HitRight            ; Interrupt to trigger HitRight routine
              reti
.org    $0004
              rcall HitLeft             ; Interrupt to trigger HitLeft Routine
              reti
.org    $003C
              rcall Receive             ;Interrupt that triggers when receives a command from USART
              reti

.org    $0046                           ; End of Interrupt Vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:
       ;Stack Pointer (VERY IMPORTANT!!!!)
```

```
        ldi mpr, high(RAMEND)
        out sph, mpr
        ldi mpr, low(RAMEND)
        out spl, mpr
        ;I/O Ports
        ldi mpr, $ff
        out DDRB, mpr
        ldi mpr, $00
        out PORTB, mpr

        ldi mpr, $00
        out DDRD, mpr

        ldi mpr, $00
        out DDRE, mpr
        ldi r24, 0
        ldi r26, 0

        ;USART1
        ;Set baudrate at 2400bps

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

        ldi mpr, high(832)
        sts UBRR1H, mpr
        ldi mpr, low(832)
        sts UBRR1L, mpr

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

        ;Set frame format: 8 data bits, 2 stop bits
        ldi mpr, (0<<UMSEL1|1<<USBS1|1<<UCSZ11|1<<UCSZ10)
        sts UCSR1C, mpr

        ;External Interrupts

        ;Set the External Interrupt Mask
        ldi mpr, (1<<INT0)|(1<<INT1)
        out EIMSK, mpr

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

        sei

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

;***********************************************************
;*      Functions and Subroutines
;***********************************************************
Receive:                                    ;Subroutine to recieve USART
                lds r17, UDR1               ; Loads USART into register 17
                mov mpr, r17                ; Move it to MPR
                andi mpr, 0b10000000        ;AND it to mask out the other bits
                tst mpr                     ; If mpr is 0
                brne Testflag               ; Branch to Testflag if it is not
                cpi r17, BotAddress         ; Compare r17 to BotAddress
                breq Setflag                ; If it is then set flag
                cpi r17, 0b01010101         ; Compare r17 to freeze command, if it is then go to
freeze
                breq Frozen

Testflag:                   ;Subroutine to test if r27 is set
```

```
               tst r21        ;If r17 is set go to Run
               brne Run
               ret
Run:                          ;Subroutine that actually runs the robot
       cpi r17, 0b10110000    ;Runs through every posibility of commands received from the remote
       BREQ Moveforward       ;Then breaks to the subroutine that corresponds to that command
       cpi r17, 0b10000000
       BREQ Movebackward
       cpi r17, 0b10100000
       BREQ Turnright
       cpi     r17, 0b10010000
       BREQ Turnleft
       cpi r17, 0b11001000
       BREQ Stop
       cpi r17, 0b11111000
       BREQ ReceiveFreeze
       reti

Setflag:                      ;Subroutine to set flag
       ldi r21, 1             ;Loads 1 into r21
       ret

Moveforward:                  ;Subroutine to move forward
       ldi mpr, MovFwd        ; Loads MovFwd into r25 and outputs to PORTB
       ldi r25, MovFwd        ;Backs up MovFwd to r25 for later
       out PORTB, mpr
       clr r21                ;Clear the flag
       ret

Movebackward:                 ;The other subroutines are identical to Moveforward except they
have their own
       ldi mpr, MovBck        ;corresponding commands
       ldi r25, MovBck
       out PORTB, mpr
       clr r21
       ret

Turnright:
       ldi mpr, TurnR
       out PORTB, mpr
       ldi r25, TurnR
       clr r21
       ret

Turnleft:
       ldi mpr, TurnL
       out PORTB, mpr
       ldi r25, TurnL
       clr r21
       ret

ReceiveFreeze:         ;Receives freeze command and outputs the command to freeze other robots
       ldi mpr, 0b01010101
       sts UDR1, mpr  ;Outputs freeze robot command to UDR1
       ret

Frozen:
       sbrs r21, 0    ;Freeze the robot when register 21 is not set and it receives a freeze
signal
       rcall Freeze
       ret

Freeze:                       ;The subrountine happens when the robot is frozen
       ldi mpr, 0b00000001    ;Loads LED display into mpr
       out PORTB, mpr         ;Outputs that display to PORTB
       ldi mpr, EIMSK         ;Back up EIMSK
       ldi r22, UCSR1B        ;Back up UCSR1B
       out EIMSK, r26         ;Outputs 0 to EIMSK to prevent interrupts
       sts UCSR1B, r26        ;Outputs 0 to USCR1B to prevent USART signals
       ldi waitcnt, WTime     ;Loads WTime into Waitcnt
       rcall Wait             ;Wait for 5 seconds
```

```
        rcall Wait
        rcall Wait
        rcall Wait
        rcall Wait
        out EIMSK, mpr          ;Restore EIMSK
        sts UCSR1B, r22         ;Restore UCSR1B
        inc r24                 ;increment counter to see how many times robot has been frozen
        cpi r24, 3              ;If counter reaches 3, go to Dead
        breq Dead
        out PORTB, r25          ;Output original content of r25 before it was frozen
        ret

Dead:
        out EIMSK, r26          ;Write 0s to EIMSK
        sts UCSR1B, r26         ;Write 0s to UCSR1B
        ret

Stop:                           ;Subroutine for stop
        ldi mpr, Halt           ;Loads halt into mpr
        out PORTB, mpr          ;Output to PORTB
        clr r21                 ;Clears the flag

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

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

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