ECE 375 LAB 8

Introduction to AVR Development Tools

Lab Time: Tuesday 7-9

Aaron Rito and Robert Lockard

Introduction

Lab 8 uses the IR sensor and the USART feature to control the tekbot remotely. One board is programmed as a remote, the other as a receiver. This report details both version of code.

Program Overview

The receiver program sets the bot to halt, and waits for a command from the remote. When a command is received, the bot will perform the action corresponding to the received opcode. If the bot receives a freeze tag signal, it sends the freeze all signal out to all bots in the area.

The transmitter program uses the IR to send commands to a specific bot. The push buttons are assigned to send different commands. When a button is hit the remote will first send out the bot iD, and then the instruction code.

Initialization Routine

The program initializes the at128 chip , and then runs a routine to initialize the stack pointer and direct access registers. Then it defines the locations for the interrupt routines. Then after the typical tekbot initialization, the USART control registers are initialized. The baud rate is set to 2400, using the UBRR USART baud rate register. Then the parity, stop bits, and character size are set using the USART control and status register. In the receiver code, the RX enable and the RX complete interrupt bits are set. A tag counter is set to 3. Then the bot is set to halt. In the transmitter code the TX enable bit is set instead of RX.

Main Routine

The receiver main routine is only 2 lines long. It updates PORTB with the current saved action and loops. The transmitter code polls PORTB for a button press, and then calls the appropriate function to send the action code.

Receiver Functions:

Receive_Command

The receive command function is called when the Rx complete interrupt bit is set. It compares the data in UDR1 register to the list of instruction codes, and calls the appropriate action function. If no match is found, it returns to the main loop.

Save_action

The save action function simply saves the last received instruction in the mpr2 register. It's purpose is so the bot can return to its last command after being frozen or interrupted by the whisker bits.

Move_forward, Move_back, Halt, Hit_Left, Hit_right

These are the action functions for the bot. After calling the save action function, all they do is set the motors to perform the desired action. The Hit_left and Hit_right functions are identical to previous labs, to maintain whisker functionality. They are called with intermediate functions that save the action, so the whisker hits don't cause undefined behavior.

Freeze Function

The freeze function first disables all interrupts, and then sets the bot to freeze for 5 seconds. It decrements a tag counter (set to 3 in initialization) and then compares it to 0. If the tag counter has reached 0, the bot goes into a death loop.

Msg_out Function

The message out function is called when the bot is acting as a middle man for tag. It disables the RX pins, and enables the TX transmitter. Then it sends out a freeze code. After the code is sent and a short delay, the RX bits are re-enabled and the TX bit disabled. This avoids any race conditions or the bot receiving its own freeze code.

Death_loop Function

The death loop function is called when the tag counter has reached 0. It's sends the bot into an infinite loop with all interrupts disabled. This loop is terminal.

Transmitter Functions:

Send_bot_id Function

This function sends the bot ID out through the UDR1 register. Then it has a short delay. This gets called every time a new action is sent out.

ForwardTX, BackwardTX, turnleftTX, turnrightTX, haltTX

These functions are all identical except for the opcode they send out. They first call the send bit id function and then send the desired opcode. A delay “waitTX” is called that waits until the transmission is finished before returning to the main loop.

WaitTx

Wait TX simply waits for the transmission to finish using SBRC, then returns to the main loop.

freezeTX

The freezeTX signal is set on an interrupt but acts the same way as the other action functions. It sends the bot id and then the opcode, then a waits for the transmission to finish. This freezes the bot.

tag_sig

The tag_sig fucnton acts the same way as the other action functions. It sends the bot id and then the opcode, then a waits for the transmission to finish. This tells the bot to freeze other bots but not itself.

Study Questions

There are no study questions for this lab.

Difficulties

I ran out of coding space with the interrupt vectors. (Relative branch out of range error). I solved the issue by embedding the function calls using rcall within smaller branch call functions.

Conclusion

The IR hardware on the tekbot can send and receive data. The USART feature on the 128 chip allows for sending and receiving up to 9bit data. The baud rate is set in the UBBR register, the parity, stopbits, and character size are set in the UCSR1C register. The interrupts and enable bits for USART are in register UCSR1B. When sending and receiving data with the same unit, care must be taken that the bot does not receive it's own transmissions. This is accomplished by disabling the TX when receiving and disabling the RX when sending.

Citations

USART ATmega328p with Assembly code to Control LED and Servo. (n.d.). Retrieved March 14, 2017, from http://www.avrfreaks.net/forum/usart-atmega328p-assembly-code-control-led-and-servo

(2015, December 3). Retrieved March 14, 2017, from https://github.com/iankronquist?tab=repositories

Receiver Source Code

```
; lab8.asm

; RX code

; Created: 3/10/2017 12:22:06 PM

; Authors: Aaron Rito, Robert Lockard

.include "m128def.inc"               ; Include definition file

; Variable and Constant Declarations

.def    mpr = r16                            ; Multi-Purpose Register

.def    waitcnt = r17                ; Wait Loop Counter

.def    ilcnt = r18                          ; Inner Loop Counter

.def    olcnt = r19                          ; Outer Loop Counter

.def     current_action = r23  ; Holds the current operation.

.def    death_counter = r24          ; keeps track of freeze tags

.def     mpr2 = r21                          ;  holds  the  commands  to  avoid  writing  over  the
opcodes



.equ    WTime = 250                          ; Time to wait in wait loop

.equ     TurnTime = 100              ; A shorter 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

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

.equ    MovFwd = (1<<EngDirR|1<<EngDirL)        ; Move Forward Command
.equ    MovBck = $00                                    ; Move Backward Command
.equ    TurnR = (1<<EngDirL)                            ; Turn Right Command
.equ    TurnL = (1<<EngDirR)                            ; Turn Left Command
.equ    Halt = (1<<EngEnR|1<<EngEnL)            ; Halt Command

.equ    BotAddress = 1                                  ;(Enter your robot's address here (8
bits))

;These macros are the values to make the TekBot Move. The action codes have been set to match
them.

.equ    MovFwdR =   (1<<EngDirR|1<<EngDirL)     ;0b01100000 Move Forward Action Code
.equ    MovBckR =   $00                                 ;0b00000000 Move Backward Action Code
.equ    TurnRR =    (1<<EngDirL)                        ;0b01000000 Turn Right Action Code
.equ    TurnLR =    (1<<EngDirR)                        ;0b00100000 Turn Left Action Code
.equ    HaltR =     (1<<EngEnR|1<<EngEnL)       ;0b10010000 Halt Action Code
.equ    freeze_msg =    0b11111000                      ;send the freeze out message code
.equ     freeze_sig=    0b01010101                      ;freeze from any source
.equ    ser_buffer = $0100                              ;a buffer for the UDR1 register

.cseg
.org    $0000                                   ; Beginning of IVs
                rjmp    INIT                    ; Reset interrupt

.org INT0addr                                   ; setting up int0
```

```
                rcall HitRight
                reti

.org INT1addr                               ; setting up int1
                rcall HitLeft
                reti

.org $003C                                          ; USART signal recieved
        rcall recieve_command
        reti
.org    $0046                               ; End of Interrupt Vectors

INIT:
        cli
        ; Stack Pointer
        ldi             mpr, low(RAMEND)
        out             SPL, mpr
        ldi             mpr, high(RAMEND)
        out             SPH, mpr

        ;USART1 Set baudrate at 2400bps
    ldi mpr, $A0
    sts UBRR1L, mpr
    ldi mpr, $01
    sts UBRR1H, mpr
    ldi mpr, (0<<U2X1)
    sts UCSR1A, mpr

    ; Enable receive interrupt
    ldi mpr, (1<<RXCIE1)|(1<<RXEN1)
        sts UCSR1B, mpr
```

```
    ; Set frame format: 8 data bits, 2 stop bits
    ; 2 stop bits is USBS1, data bits are UCCSZ*
ldi mpr, (1<<UCSZ10)|(1<<UCSZ11)|(1<<USBS1)|(1<<UPM01)
sts UCSR1C, mpr

; Initialize Port B for output
    ldi             mpr, $FF            ; Set Port B Data Direction Register
    out             DDRB, mpr           ; for output
    ldi             mpr, $00            ; Initialize Port B Data Register
    out             PORTB, mpr          ; so all Port B outputs are low

    ; Initialize Port D for input
    ldi             mpr, $00            ; Set Port D Data Direction Register
    out             DDRD, mpr           ; for input
    ldi             mpr, $FF            ; Initialize Port D Data Register
    out             PORTD, mpr          ; so all Port D inputs are Tri-State

    ; Initialize TekBot Forward Movement
    ldi             mpr, MovFwd         ; Load Move Forward Command
    out             PORTB, mpr          ; Send command to motor

    ; set the Interrupt control state in EIRCA to falling
    ldi mpr, (1 << ISC11) | (1 << ISC01)
    sts EICRA, mpr

    ; Set the External Interrupt Mask for int0 and int1
    in mpr, EIMSK
    ori mpr, (1<<INT0) | (1<<INT1)
    out EIMSK, mpr
    ldi mpr, $03
    MOV death_counter, mpr
    ldi current_action, 0b10010000
```

```
        sei


;main loop

MAIN:


        out PORTB, current_action     ; sets PORTB to the saved action in case of freeze or
whisker hit

        rjmp MAIN                                   ; loop back, interrupts do the rest


;USART recieve opcode command

recieve_command:


        ldi XL, low(ser_buffer)             ;set X pointer to point at UDR1 (not sure I need
this)

        ldi XH, high(ser_buffer)

        lds mpr2, UDR1                      ;load the opcode


        ;the following compares the recieved opcode with the motor fucntions and calls the
appropriate action

        cpi mpr2, MovFwdR

        breq move_forward_call


        cpi mpr2, MovBckR

        breq move_back


        cpi mpr2, TurnRR

        breq HitRight_call


        cpi mpr2, TurnLR

        breq HitLeft_call


        cpi mpr2, Halt

        breq Halt_now
```

```
        cpi mpr2, freeze_sig
        breq freeze

        cpi mpr2, freeze_msg
        breq msg_out

        ret

;ran out of space on the interrupt vector space, need calls for calls
HitLeft_call:
        rcall   save_action
        rcall HitLeft
        ret
HitRight_call:
        rcall   save_action
        rcall HitRight
        ret
move_forward_call:
        rcall move_forward
        ret
;death loop after 3 freeze tags
death_loop:
        ldi mpr, $FF   ;setPORTB to all on death signal
        out PORTB, mpr
        rjmp death_loop

;Halt action
Halt_now:
        rcall   save_action                 ; save the action so we can return to it
        ldi             mpr, HaltR
        out             PORTB, mpr
```

```
        ldi             waitcnt, Wtime
        rcall   wait
        ret

;freeze as a slave
freeze:
        cli
        ldi mpr, $00            ;disable the interrupts
        out EIMSK, mpr
        out EIFR, mpr
        ldi mpr, $0F
        out PORTB, mpr          ; show the freeze signal on PORTB
        ldi     waitcnt, WTime          ; Wait for 1 second
        rcall wait
        dec death_counter       ; decrement the tag counter
        cpi death_counter, $00
        breq death_loop                 ; die if taqgged 3 times
        in mpr, EIMSK           ;re-enable interrpus
        ori mpr, (1<<INT0) | (1<<INT1)
        out EIMSK, mpr
        out EIFR, mpr
        sei
        ret

;freeze as a middle man
msg_out:

        ldi mpr, (0<<RXCIE1)|(0<<RXEN1) | (1<<TXEN1) ;disable the reciever, enable the transmitter
        sts UCSR1B, mpr
        ldi mpr, 0b01010101                                                     ; send out the
freeze tag signal
        sts UDR1, mpr
```

```
        out PORTB, mpr

        ldi     waitcnt, TurnTime                                       ;  Wait  for  1
second for the transmit to finish

        rcall Wait

        ldi mpr, (1<<RXCIE1)|(1<<RXEN1) | (0<<TXEN1) ;disable   transmitter,   enable   reciever,
avoids race condition

        sts UCSR1B, mpr

        ret


;move backwards

move_back:

        rcall   save_action           ; save the action command

        ldi             mpr, MovBck             ; Load Move back Command

        out             PORTB, mpr              ; Send command to motors

        ldi             waitcnt, TurnTime       ; Wait for 1 second

        rcall   Wait                    ; Call wait function

        ret

;move forward

move_forward:

        rcall   save_action           ;save the action command

        ldi             mpr, MovFwd             ; Load Move Forward Command

        out             PORTB, mpr              ; Send command to motors

        ldi             waitcnt, TurnTime       ; Wait for 1 second

        rcall   Wait                    ; Call wait function

        ret

;turn right, the action command is saved in the action call in case the whiskers are using the
fucntion

HitRight:

        cli                             ;disable interrrupts

        ldi mpr, $00

        out EIMSK, mpr

        out EIFR, mpr
```

```
        ; Move Backwards for a second
        ldi             mpr, MovBck     ; Load Move Backward command
        out             PORTB, mpr      ; Send command to port
        ldi             waitcnt, TurnTime       ; Wait for 1 second
        rcall   Wait                    ; Call wait function

        ; Turn left for a second
        ldi             mpr, TurnRR     ; Load Turn Left Command
        out             PORTB, mpr      ; Send command to port
        ldi             waitcnt, TurnTime       ; Wait for 1 second
        rcall   Wait                    ; Call wait function

        in mpr, EIMSK                               ;re-enable interrupts
        ori mpr, (1<<INT0) | (1<<INT1)
        out EIMSK, mpr
        out EIFR, mpr
        sei
        ret                             ; Return from subroutine

;turn left, the action command is saved in the action call in case the whiskers are using the
fucntion
HitLeft:
        cli
        ldi mpr, $00
        out EIMSK, mpr
        out EIFR, mpr

        ; Move Backwards for a second
        ldi             mpr, MovBck     ; Load Move Backward command
        out             PORTB, mpr      ; Send command to port
        ldi             waitcnt,TurnTime        ; Wait for 1 second
        rcall   Wait                    ; Call wait function
```

```
        ; Turn right for a second
        ldi             mpr, TurnLR     ; Load Turn Left Command
        out             PORTB, mpr      ; Send command to port
        ldi             waitcnt, TurnTime       ; Wait for 1 second
        rcall   Wait                    ; Call wait function


        in mpr, EIMSK                                   ;re-enable interrupts
        ori mpr, (1<<INT0) | (1<<INT1)
        out EIMSK, mpr
        out EIFR, mpr
        sei
        ret


;the same wait function from the other labs
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

; saves the action codes for certain actions.
save_action:
	MOV current_action, mpr2
	ret
```

TRANSMITTER SOURCE CODE

```
;***********************************************************
;*
;*	Tek_Bot Transmitter
;*
;*	This Assembly code transmits the required operational
;*	commands to an identical Tek_Bot with an reciever assembly
;*	flash. This program has the ability through its IR
;*	transmitter to send out six differnt commands including:
;*	move foward, move backward, turn right, turn left, halt,
;*	and freeze.
;*
;*	This is the TRANSMIT  file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*	 Author: RJ Lockard & Aron Rito
;*	   Date: 3/13/2017
;*
;***********************************************************

.include "m128def.inc"		; Include definition file
```

```
;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                               ; Multi-Purpose Register

.def    ilcnt = r18                             ; Inner Loop Counter
.def    olcnt = r19                             ; Outer Loop Counter
.def    waitcnt = r20                ; Wait Loop Counter

                                                ;  Use  these  commands  between  the
remote and TekBot
                                                ; MSB = 1 thus:
                                                ; commands are shifted right by one
and ORed with 0b10000000 = $80

.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    BotID = 00000001             ;  8  bit  BotID  specifies  the  id  in  which  in  which  the
specific remote will connect to the specific reciever

.equ    Wtime = 50                              ; intialized wait time

.equ    MovFwd =  (1<<(EngDirR)|1<<(EngDirL))        ; Move Forwards Command
.equ    MovBck =  ($00)                                                 ;          Move
Backwards Command
.equ    TurnR =   (1<<(EngDirL))                                  ; Turn Right Command
.equ    TurnL =   (1<<(EngDirR))                                  ; Turn Left Command
.equ    Halt =    (1<<(EngEnR)|1<<(EngEnL))                  ; Halt Command
```

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


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


.org    $0002
               rcall   FreezeTX
               reti


.org    $0004
               rcall   HaltTX
               reti


.org    $0006
               rcall   tag_sig
               reti


.org    $0046                                   ; End of Interrupt Vectors


;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:


                                                                ; 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




                                                            ; I/O Ports
        ldi             mpr, $00                    ; Initialize Port B for outputs
        out             PORTB, mpr
        ldi             mpr, $FF
        out             DDRB, mpr

        ldi             mpr, $FF                    ; Initialize Port D for inputs
        out             PORTD, mpr
        ldi             mpr, $00
        out             DDRD, mpr


                                                            ; USART1
                                                            ; Set baudrate at 2400bps
                                                            ; Set the baud rate and enable double
data rate
    ldi mpr, $A0
    sts UBRR1L, mpr
    ldi mpr, $01
    sts UBRR1H, mpr
    ldi mpr, (0<<U2X1)
    sts UCSR1A, mpr


                                                            ; Enable transmitter
    ldi mpr, (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
                                                        ; Set the Interrupt to sense for
falling edge
                                                        ; Set the External Interrupt Mask
        ldi     mpr, 0x0
        out     EICRB, mpr
        ldi     mpr, 0xa
        sts     EICRA, mpr
        ldi     mpr, 0x03
        out     EIMSK, mpr


        clr mpr                                 ; Clear mpr
        out PORTB, mpr


        sei                                             ;Set the External Interrupt Mask


;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:


        sbis PIND, 7                        ; Set move foward command to PIND 7 (button 7)
        rjmp forwardTX
        sbis PIND, 6                        ; Set move backward command to PIND 6 (button 6)
        rjmp backwardTX
        sbis PIND, 5                        ; Set turn left command to PIND 5 (button 5)
        rjmp turnLeftTX
```

```
        sbis PIND, 4                    ; Set turn right command to PIND 4 (button 4)

        rjmp turnRightTX

        sbis PIND, 2                    ; Set test RX bot transmittsion to test freeze and
death

        rjmp tag_sig

        rjmp MAIN


;***********************************************************

;*      Functions and Subroutines

;***********************************************************


sendBotId:

        ldi mpr, BotID                  ; Send Bot ID command

        sts UDR1, mpr

        rcall waitTX                    ; rcall to the wait transmition command to ensure no
double button press

        ret                                             ; return from subrountine


tag_sig:                                        ; test command to freeze rx, triggered by
Pin (button 3)

        ldi waitcnt, Wtime

        rcall wait                              ; rcall to the wait transmition command to
ensure no double button press

        rcall sendBotId

        ldi mpr, 0b01010101             ; Loading coresponding action opcode

        sts UDR1, mpr

        out PORTB, mpr                  ; Port the corresponding command to be transmitted

        rcall waitTX                    ; rcall to the wait transmition command

        rjmp MAIN                               ; rjumps back to the main routine


forwardTX:                                      ; Move foward command, triggered by Pin
(button 7)

        ldi waitcnt, Wtime

        rcall wait                              ; rcall to the wait transmition command to
ensure no double button press
```

```
        rcall sendBotId
        ldi mpr, MovFwd                         ; Loading coresponding action opcode
        sts UDR1, mpr
        out PORTB, mpr                  ; Port the corresponding command to be transmitted
        rcall waitTX                    ; rcall to the wait transmition command
        rjmp MAIN                               ; rjumps back to the main routine


backwardTX:                                     ; Move backward command, triggered by Pin 6
(button 6)
        ldi waitcnt, Wtime
        rcall wait                              ; rcall to the wait transmition command to
ensure no double button press
        rcall sendBotId
        ldi mpr, MovBck                         ; Loading coresponding action opcode
        sts UDR1, mpr
        out PORTB, mpr                  ; Port the corresponding command to be transmitted
        rcall waitTX                    ; rcall to the wait transmition command
        rjmp MAIN                               ; rjumps back to the main routine


turnLeftTX:                                     ; Move trunLeft command, triggered by Pin 5
(button 5)
        ldi waitcnt, Wtime
        rcall wait                              ; rcall to the wait transmition command to
ensure no double button press
        rcall sendBotId
        ldi mpr, TurnL                  ; Loading coresponding action opcode
        sts UDR1, mpr
        out PORTB, mpr                  ; Port the corresponding command to be transmitted
        rcall waitTX                    ; rcall to the wait transmition command
        rjmp MAIN                               ; rjumps back to the main routine


turnRightTX:                                    ; Move turnRight command, triggered by Pin 4
(button 4)
        ldi waitcnt, Wtime
```

```
        rcall wait                              ; rcall to the wait transmition command to
ensure no double button press

        rcall sendBotId

        ldi mpr, TurnR                      ; Loading coresponding action opcode

        sts UDR1, mpr

        out PORTB, mpr                      ; Port the corresponding command to be transmitted

        rcall waitTX                        ; rcall to the wait transmition command

        rjmp MAIN                               ; rjumps back to the main routine


HaltTX:                                     ; Halt command, triggered by interrupt (button 1)

        ldi waitcnt, Wtime

        rcall wait                          ; rcall to the wait transmition command to ensure no
double button press

        rcall sendBotId

        ldi mpr, Halt                       ; Loading coresponding action opcode

        sts UDR1, mpr

        out PORTB, mpr                      ; Port the corresponding command to be transmitted

        rcall waitTX                        ; rcall to the wait transmition command

        ret


FreezeTX:                                       ; FreezeTX command, triggered by interrupt
(button 0)

        ldi waitcnt, Wtime

        rcall wait                              ; rcall to the wait transmition command to
ensure no double button press

        rcall sendBotId

        ldi mpr, 0b11111000                 ; Loading coresponding action opcode

        sts UDR1, mpr

        out PORTB, mpr                      ; Port the corresponding command to be transmitted

        rcall waitTX                        ; rcall to the wait transmition command

        ret


waitTX:                                         ; Wait subroutine

        lds mpr, UCSR1A
```

```
        sbrs mpr, TXC1
        ret     `


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
```