ECE 375 LAB 7

USART Remotely Communicated Rock Paper Scissors

Lab Time: Wednesday 10:00-11:50

Kenneth Tang

Travis Fredrickson

INTRODUCTION

This program allows two players to play Rock Paper Scissors on two ATmega32U4 boards. The two boards communicate using USART and must have wires connecting pins between boards so they can communicate.

PROGRAM OVERVIEW

The program has six stages: (0) Idle, (1) Ready Up, (2) Choose Hand, (3) Transfer Data, (4) Reveal Hands, and (5) Result. On reset, the boards default to Stage (0) Idle where a welcome message is displayed on the LCD telling the player to press PD7 to ready up. Pressing PD7 will advance to Stage (1) Ready Up. Stage (1) Ready Up displays a message to the LCD indicating that the opponent must also ready up, and a message is sent via USART to the other board to tell it that the player is ready. Once both players are ready, the program will advance to Stage (2) Choose Hand. In Stage (2) Choose Hand, the six second timer will start, and pressing PD4 will cycle hand choices. Once the timer is done, it will advance to Stage (3) Transfer Data. In Stage (3) Transfer Data, each board sends data to the other to indicate what each players hand is, after which it will advance to Stage (4) Reveal Hands. In Stage (4) Reveal Hands, the six second timer is started, and the oppoent's hand is displayed to each player. After the timer is done, it will advance to Stage (5) Result. In Stage (5) Result, it will start the six second timer, calculate if the player won, lost, or drawed, and display the result. After the timer is done, it will reset back to Stage (0) Idle.

The game flow diagram below illustrates what is described in the paragraph above.

Game Flow:

- Stage 0: Idle
  - LCD: Welcome message
  - Advance to next stage: PD7
- Stage 1: Ready Up
  - LCD: Ready up message
  - USART
  - Advance to next stage: When both players have pressed PD7
- Stage 2: Choose Hand
  - LCD top: Choose hand
  - LCD bottom: Current User hand
  - Timer
  - Cycle Hand on PD4 press
  - Advance to next stage: When Timer ends
- Stage 3: Transfer Data
  - USART
  - Advance to next stage: When data has been transferred and received
- Stage 4: Reveal Hands
  - LCD top: Opponent hand
  - LCD bottom: User hand
  - Timer
  - Advance to next stage: When Timer ends
- Stage 5: Result
  - LCD top: Result
  - LCD bottom: User hand
  - Timer
  - Advance to next stage: When Timer ends

TIMER AND LED FUNCTIONALITY

The 6 second Timer operates by using Timer/Counter1 (TCNT1) in Normal mode. TCNT1 has a delay of 1.5 seconds, so by running it four times we can create a 6 second Timer. Every time TCNT1 reaches its max value, an interrupt (TOV1) is triggered which calls the Timer function. Within the Timer function, LEDs 7:4 are updated to indicate to the players how many seconds are left: 1111 = 6 seconds, 0111 = 4.5 seconds, 0011 = 3 seconds, 0001 = 1.5 seconds.

LCD FUNCTIONALITY

The LCD works by loading string addresses stored in Program Memory into the Z register, then calling LCD functions to print to both rows, top row, or bottom row. It also relies on a LCD driver file provided by the course intructors, which must be included in the program code.

CALCULATIONS

1. USART Baud Rate Resister (UBBR)

   We want a Baud rate of 2400bps with the USART transmission speed doubled. The equation below is for calculating the UBBR value necessary to allow that.

   $$UBBR = \frac{clk}{8 \cdot Baud} - 1$$

   We know clk = 8MHz and Baud = 2400bps. The result is **UBBR = 415.666 = 416 = $01A0**.

2. Timer/Counter1 Prescale and Value

   We want to use Timer/Counter1 to create a delay of 1.5s. The equation below is for calculating the Value necessary to that.

   $$Value = Max + 1 - \frac{Delay \cdot clk}{Prescale}$$

   We know Max = $FFFF = 65535, Delay = 1.5s, and clk = 8MHz, but we don't know Prescale. Prescale can be 1, 8, 64, 256, or 1024. The max value for Value is 65535 (16 bits). By plugging in each Prescale value, we can solve for Value until it is smaller that 65535. The result is **Prescale = 256** and **Value = 18661 = $48E5**.

INITIALIZATION ROUTINE

Initialize the Stack Pointer to RAMEND. Initialize Port D Pin 3 for output, the rest for input, and enable pull up resistors. Initialize Port B for output. Initialize LCD and turn on backlight. Initialize Data Memory variables TIMER_STAGE to 4, GAME_STAGE to 0, user and opponent hands to Rock, and user and opponent ready flags to not ready. Initialize USART1 for double data rate, set recieve and transmit complete interrupts enable, transmitter and reciever enable, 8 data bits, 2 stop bits, asnych, no parity, and Baud Rate to 2400. Initialize Timer/Counter1 for normal mode and prescale to 256. Initialize interrupts INT1 and INT0 to trigger on falling edge, and disable INT1 and INT0 for now. Call NEXT_GAME_STAGE to get game started and to print welcome message to LCD. Enable global interrupts.

MAIN ROUTINE

Does nothing. Program operates on interrupts to call functions.

SUBROUTINES

1. LCD ALL
    a. Prints a string to the entire LCD and assumes that Z is already pointing to the string we want to print.
2. LCD TOP
    a. Prints a string to the top row of the LCD and assumes that Z is already pointing to the string we want to print.
3. LCD BOTTOM
    a. Prints a string to the bottom row of the LCD and assumes that Z is already pointing to the string we want to print.
4. MESSAGE RECEIVE
    a. After receiving data, this function decides what to do with it. It performs checks on it to see if the start message was sent or a hand signal then branches appropriately.
5. READY CHECK
    a. Does a status check after a message has transmitted or received.
6. SEND READY
    a. Sends the ready message via USART1 and sets the user's ready flag to be equal to SIGNAL_READY.
7. RECEIVE START
    a. Called when USART1 receive interrupt determines it has received a start message. Changes the opponents ready flag to be SIGNAL_READY and calls READY_CHECK to see if the game stage should advance (ie start the game).
8. SEND HAD
    a. Sends the user's current hand via USART1
9. NEXT GAME STAGE
    a. This function contains the core functions of the game. It calls functions based on the stage of the game which is always incremented when this function is called.
10. GAME STAGE 0
    a. This function prints the welcome message to the LCD.

11. GAME STAGE 1
    a. Prints the ready message to the LCD and sends the ready message via USART1.
12. GAME STAGE 2
    a. Starts the 6 second timer and displays the default choice of what the user's hand is to the bottom row of the LCD.
13. GAME STAGE 4
    a. Starts the 6 second timer and displays the opponent's hand to the top row of the LCD.
14. GAME STAGE 5
    a. Starts the 6 second timer and determines the winner based on the user's hand and opponent's hand. Displays the result on the top line of the LCD.
15. STORE HAND
    a. Stores the incoming opponent's hand to HAND_OPNT. Assumes that the mpr is holding the intended message. Should be called directly from MESSAGE_RECEIVE.
16. PRINT HAND
    a. Prints the opponent's hand on the top row of the LCD.
17. CYCLE HAND
    a. Cycles the user's hand through the three available options and updates the LCD accordingly.
18. TIMER
    a. Starts the timer for a delay of 1.5 seconds and updates PORTB's upper 4 LEDs to show the time remaining out of 6 seconds. Once the 6 seconds are completed (4 subsequent calls of TIMER) the TOV01 flag is disabled.
19. BUSY WAIT
    a. A wait loop that loops enough times to cause a delay of 15ms. The allowed limit for busy waiting in this lab is 150ms. This is used only to clear the interrupt queue and prevent duplicate calls.

STUDY QUESTIONS

None.

DIFFICULTIES

One difficult we had was a double call to a function. It was causing one board to advance through the game faster than the other. After much confusion, simply deleting that one line fixed the issue and the game finally worked perfectly.

CONCLUSION

The program works as expected. Challenge functionality of having two hands and picking one was not implemented.

SOURCE CODE

```
;***********************************************************
;*   ECE375 Lab7: Rock Paper Scissors
;*
;*   Author: Kenneth Tang
;*           Travis Fredrickson
;*     Date: 11/19/2022
;*
;***********************************************************

.include "m32U4def.inc"         ; Include definition file

;***********************************************************
;*  Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16               ; Multi-Purpose Register
.def    waitcnt = r17           ; Wait Loop Counter
.def    ilcnt = r18
.def    olcnt = r19

.equ    WTime = 15              ; Time to wait in wait loop

.equ    SIGNAL_READY        = 0b1111_1111   ; Signal for ready to start game
.equ    SIGNAL_NOT_READY    = 0b0000_0000   ; Signal for not ready to start game
.equ    SIGNAL_ROCK         = 0b0000_0001   ; Signal for Rock
.equ    SIGNAL_PAPER        = 0b0000_0010   ; Signal for Paper
.equ    SIGNAL_SCISSORS     = 0b0000_0011   ; Signal for Scissors

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

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


.org    $0002                   ; INT0  Cycle through selection
        rcall CYCLE_HAND
        reti

.org    $0004                   ; INT1 Send Start Msg
        rcall NEXT_GAME_STAGE
        reti

.org    $0028                   ; Timer/Counter 1 Overflow
        rcall TIMER
        reti

.org    $0032                   ; USART1 Rx Complete
        rcall MESSAGE_RECEIVE
        reti

.org    $0034                   ; USART Data Register Empty
        reti

.org    $0036                   ; USART1 Tx Complete
        rcall READY_CHECK
        reti
```

```
.org    $0056                   ; End of Interrupt Vectors

;***********************************************************
;*  Program Initialization
;***********************************************************
INIT:
    ; Initialize the Stack Pointer
    ldi     mpr, high(RAMEND)
    out     SPH, mpr
    ldi     mpr, low(RAMEND)
    out     SPL, mpr

    ; I/O Ports
    ldi     mpr, (1<<PD3 | 0b0000_0000) ; Set Port D pin 3 (TXD1) for output
    out     DDRD, mpr                   ; Set Port D pin 2 (RXD1) for input
    ldi     mpr, $FF                    ; Enable pull up resistors
    out     PORTD, mpr

    ; Configure PORTB for output
    ldi     mpr, $FF
    out     DDRB, mpr
    ldi     mpr, $00
    out     PORTB, mpr

    ; USART1 Config
    ; Set double data rate
    ldi     mpr, (1<<U2X1)
    sts     UCSR1A, mpr
    ; Set recieve & transmit complete interrupts, transmitter & reciever enable, 8 data bits
frame formate
    ldi     mpr, (1<<RXCIE1 | 1<<TXCIE1 | 0<<UDRIE1 | 1<<RXEN1 | 1<<TXEN1 | 0<<UCSZ12)
    sts     UCSR1B, mpr
    ; Set frame formate: 8 data bits, 2 stop bits, asnych, no parity
    ldi     mpr, (0<<UMSEL11 | 0<<UMSEL10 | 0<<UPM11 | 0<<UPM10 | 1<<USBS1 | 1<<UCSZ11 |
1<<UCSZ10 | 0<<UCPOL1)
    sts     UCSR1C, mpr
    ; Baud to 2400 @ double data rate
    ldi     mpr, high(416)
    sts     UBRR1H, mpr
    ldi     mpr, low(416)
    sts     UBRR1L, mpr

    ; Timer/Counter 1
    ; Setup for normal mode WGM 0000
    ; COM disconnected 00
    ; Use OCR1A for top value
    ; CS TBD, using 100
    ldi     mpr, (0<<COM1A1 | 0<<COM1A0 | 0<<COM1B1 | 0<<COM1B0 | 0<<WGM11 | 0<<WGM10)
    sts     TCCR1A, mpr
    ldi     mpr, (0<<WGM13 | 0<<WGM12 | 1<<CS12 | 0<<CS11 | 0<<CS10)
    sts     TCCR1B, mpr

    ; LED Initialization
    call    LCDInit                         ; Initialize LCD
    call    LCDBacklightOn
    ldi     ZH, high(STRING_IDLE<<1)        ; Point Z to the welcome string
    ldi     ZL, low(STRING_IDLE<<1)
    call    LCD_ALL                         ; Print welcome message

    ; Data Memory Variables
        ; TIMER_STAGE
    ldi     mpr, 4
    ldi     XH, high(TIMER_STAGE)
    ldi     XL, low(TIMER_STAGE)
    st      X, mpr

        ; GAME_STAGE
    ldi     mpr, 0
    ldi     XH, high(GAME_STAGE)
    ldi     XL, low(GAME_STAGE)
    st      X, mpr
```

```
        ; HANDs
    ldi     mpr, SIGNAL_ROCK            ; Default hand
    ldi     XH, high(HAND_OPNT)
    ldi     XL, low(HAND_OPNT)
    st      X, mpr
    ldi     XH, high(HAND_USER)
    ldi     XL, low(HAND_USER)
    st      X, mpr

        ; READY Flags
    ldi     mpr, SIGNAL_NOT_READY
    ldi     XH, high(READY_OPNT)
    ldi     XL, low(READY_OPNT)
    st      X, mpr
    ldi     XH, high(READY_USER)
    ldi     XL, low(READY_USER)
    st      X, mpr



    ; External Interrupts
    ; Initialize external interrupts
    ldi     mpr, 0b0000_1010            ; Set INT1, INT0 to trigger on
    sts     EICRA, mpr                  ; falling edge

    ; Configure the External Interrupt Mask
    ldi     mpr, (0<<INT1 | 0<<INT0)    ; Disable INT1 and INT0 for now
    out     EIMSK, mpr

    rcall   NEXT_GAME_STAGE


    ; Enable global interrupts
    sei


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

        rjmp    MAIN


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

; Printing functions ----------------
LCD_ALL:
    ;-----------------------------------------------------------
    ; Func: LCD All
    ; Desc: Prints a string to the entire LCD
    ;       Assumes Z already points to string.
    ;-----------------------------------------------------------
    ; Save variables
    push    mpr
    push    ilcnt
    push    XH
    push    XL

    ; Set parameters
    ldi     XH, $01                     ; Point X to LCD top line
    ldi     XL, $00                     ; ^
    ldi     ilcnt, 32                   ; Loop 32 times for 32 characters

 LCD_ALL_LOOP:
    ; Load in characters
    lpm     mpr, Z+
    st      X+, mpr
```

```
    dec     ilcnt
    brne    LCD_ALL_LOOP

    ; Write to LCD
    call    LCDWrite

    ; Restore variables
    pop     XL
    pop     XH
    pop     ilcnt
    pop     mpr

    ; Return from function
    ret

LCD_TOP:
    ;-----------------------------------------------------------
    ; Func: LCD Top
    ; Desc: Prints a string to the top row of the LCD
    ;       Assumes Z already points to string.
    ;-----------------------------------------------------------
    ; Save variables
    push    mpr
    push    ilcnt
    push    XH
    push    XL

    ; Set parameters
    ldi     XH, $01                 ; Point X to LCD top line
    ldi     XL, $00                 ; ^
    ldi     ilcnt, 16               ; Loop 16 times for 16 characters

 LCD_TOP_LOOP:
    ; Load in characters
    lpm     mpr, Z+
    st      X+, mpr
    dec     ilcnt
    brne    LCD_TOP_LOOP

    ; Write to LCD
    call    LCDWrite

    ; Restore variables
    pop     XL
    pop     XH
    pop     ilcnt
    pop     mpr

    ; Return from function
    ret

LCD_BOTTOM:
    ;-----------------------------------------------------------
    ; Func: LCD Bottom
    ; Desc: Prints a string to the bottom row of the LCD
    ;       Assumes Z already points to string.
    ;-----------------------------------------------------------
    ; Save variables
    push    mpr
    push    ilcnt
    push    XH
    push    XL

    ; Set parameters
    ldi     XH, $01                 ; Point X to LCD bottom line
    ldi     XL, $10                 ; ^
    ldi     ilcnt, 16               ; Loop 16 times for 16 characters

 LCD_BOTTOM_LOOP:
    ; Load in characters
    lpm     mpr, Z+
```

```
    st      X+, mpr
    dec     ilcnt
    brne    LCD_BOTTOM_LOOP

    ; Write to LCD
    call    LCDWrite

    ; Restore variables
    pop     XL
    pop     XH
    pop     ilcnt
    pop     mpr

    ; Return from function
    ret

; USART ----------------------------
MESSAGE_RECEIVE:
    ;-------------------------------------------------------------------
    ; Sub:  Message Receive
    ; Desc: After receiving data, this function decides what to do with it
    ;       It performs checks on it to see what was sent in then branches
    ;       to the appropriate function.
    ;-------------------------------------------------------------------
    push mpr
    push ZH
    push ZL
    cli                             ; Turn interrupts off

    ;--------- Read message in UDR1 -----------;
    lds     mpr, UDR1               ; Read the incoming data
    cpi     mpr, SIGNAL_READY
    breq    MR_READY                ; If its the ready signal
    call    STORE_HAND              ; Else it must be a hand signal
    rjmp    MR_END


 MR_READY:
    call    RECEIVE_START           ; Set OPNT ready flag and advance game stage

 MR_END:
    sei                             ; Turn interrupts back on
    pop ZL
    pop ZH
    pop mpr
    ret

READY_CHECK:
    ;-------------------------------------------------------------------
    ; Sub:  Ready Check
    ; Desc: Does a status check after a message has been transmitted on USART1
    ;-------------------------------------------------------------------
    push mpr
    push ilcnt
    push ZH
    push ZL
    push XH
    push XL

    ;--------- Check to see if we should start the game ----------------;
    ldi     ZH, high(READY_USER)            ; Load both ready flags
    ldi     ZL, low(READY_USER)
    ld      mpr, Z
    ldi     XH, high(READY_OPNT)
    ldi     XL, low(READY_OPNT)
    ld      ilcnt, X
    cpi     mpr, SIGNAL_READY
    brne    TC_END                          ; If they aren't equal to ready jump to end
    cpi     ilcnt, SIGNAL_READY
    brne    TC_END                          ; If they aren't equal to ready jump to end
```

```
    ;--------- If both ready ----------------;
    ldi     mpr, SIGNAL_NOT_READY           ; Change ready flags
    st      Z, mpr
    st      X, mpr
    rcall   NEXT_GAME_STAGE                 ; If both flags are ready, advance game

 TC_END:
    ; Other checks
    pop XL
    pop XH
    pop ZL
    pop ZH
    pop ilcnt
    pop mpr
    ret

SEND_READY:
    ;-----------------------------------------------------------------
    ; Sub:  Send ready
    ; Desc: Sends the ready message via USART1 and sets the user's ready flag
    ;       to be SIGNAL_READY
    ;-----------------------------------------------------------------
    push mpr
    push waitcnt
    push ZH
    push ZL


    ldi     ZH, high(READY_USER)        ; Load the ready flag
    ldi     ZL, low(READY_USER)
    ldi     mpr, SIGNAL_READY
    st      Z, mpr                      ; Store a 1 to the ready flag

    ;----- Transmit via USART ----------;
 Ready_Transmit:
    lds     mpr, UCSR1A                 ; Load in USART status register
    sbrs    mpr, UDRE1                  ; Check the UDRE1 flag
    rjmp    Ready_Transmit              ; Loop back until data register is empty

    ldi     mpr, SIGNAL_READY           ; Send the start message to the other board
    sts     UDR1, mpr

    ;----- Clear the queue -------------;
    rcall   BUSY_WAIT                   ; Wait to clear queue
    ldi     mpr, 0b0000_0011            ; Clear interrupts
    out     EIFR, mpr

    pop ZL
    pop ZH
    pop waitcnt
    pop mpr
    ret

RECEIVE_START:
    ;-----------------------------------------------------------------
    ; Sub:  Receive start
    ; Desc: Called when USART1 receive interrupt determines it has
    ;       received the start message. Changes the opponents ready
    ;       flag to be SIGNAL_READY and calls READY_CHECK to see if
    ;       the game stage should advance (ie start the game)
    ;-----------------------------------------------------------------
    push mpr
    push ZH
    push ZL

    ldi     mpr, SIGNAL_READY       ; Change opponents ready flag to 1
    ldi     ZH, high(READY_OPNT)
    ldi     ZL, low(READY_OPNT)
    st      Z, mpr
    call    READY_CHECK             ; Check to see if we should start
```

```
    pop ZL
    pop ZH
    pop mpr
    ret

SEND_HAND:
    ;-----------------------------------------------------------------
    ; Sub:  Send hand
    ; Desc: Sends the user's current hand via USART1.
    ;-----------------------------------------------------------------

    push mpr
    push ZH
    push ZL

 Hand_Transmit:
    ; See if the USART data register is empty
    lds     mpr, UCSR1A             ; UDRE1 will be 1 when buffer is empty
    sbrs    mpr, UDRE1              ; Test only the 5th bit
    rjmp    Hand_Transmit

    ldi     ZH, high(HAND_USER)     ; Load the user's hand
    ldi     ZL, low(HAND_USER)
    ld      mpr, Z
    sts     UDR1, mpr               ; Send user's hand via USART1

    pop ZL
    pop ZH
    pop mpr
    ret

; Core game -------------------------
NEXT_GAME_STAGE:
    ;-------------------------------------------------------------
    ; Func: Next game stage
    ; Desc: This is essentially the core function of the game.
    ;       Game stages change as follows:
    ;           0 -> 1      0 = IDLE
    ;           1 -> 2      1 = READY UP
    ;           2 -> 3      2 = SELECT HAND
    ;           3 -> 4      3 = SEND & RECEIVE HANDS
    ;           4 -> 5      4 = REVEAL HANDS
    ;           5 -> 0      5 = RESULT
    ;-------------------------------------------------------------
    ; Save variables
    push    mpr
    push    XH
    push    XL

    ; Branch based on current Game Stage
    ldi     XH, high(GAME_STAGE)
    ldi     XL, low(GAME_STAGE)
    ld      mpr, X

    cpi     mpr, 0
    breq    NEXT_GAME_STAGE_0
    cpi     mpr, 1
    breq    NEXT_GAME_STAGE_1
    cpi     mpr, 2
    breq    NEXT_GAME_STAGE_2
    cpi     mpr, 3
    breq    NEXT_GAME_STAGE_3
    cpi     mpr, 4
    breq    NEXT_GAME_STAGE_4
    cpi     mpr, 5
    breq    NEXT_GAME_STAGE_5

    ; If no compare match, branch to end
    rjmp    NEXT_GAME_STAGE_END

 NEXT_GAME_STAGE_0:                     ; IDLE
```

```
    ; Print the welcome message
    ; Enable PD7
    rcall   GAME_STAGE_0
    ldi     mpr, 1                      ; Update GAME_STAGE
    st      X, mpr
    ldi     mpr, (1<<INT1)              ; Enable INT1 (PD7) so it can start the game again
    out     EIMSK, mpr
    rjmp    NEXT_GAME_STAGE_END         ; Jump to end

 NEXT_GAME_STAGE_1:                     ; READY UP
    ; Disable PD7
    ; Print ready message
    ; Send ready message
    ldi     mpr, (0<<INT1)              ; Disable INT1 (PD7) because it's only use was to start
the game
    out     EIMSK, mpr
    rcall   GAME_STAGE_1                ; Do stuff for this stage
    ldi     mpr, 2                      ; Update GAME_STAGE
    st      X, mpr
    rjmp    NEXT_GAME_STAGE_END         ; Jump to end

 NEXT_GAME_STAGE_2:                     ; CHOOSE HAND
    ; Display user hand on bottom row
    ; Starts the timer
    ; Enables PD4
    rcall   GAME_STAGE_2
    ldi     mpr, 3                      ; Update GAME_STAGE
    st      X, mpr
    ldi     mpr, (1<<INT0)              ; Enable INT0 so hand can be changed
    out     EIMSK, mpr
    rjmp    NEXT_GAME_STAGE_END         ; Jump to end

 NEXT_GAME_STAGE_3:
    ; Disable PD4
    ; Send user's hand
    ; wait for recieve
    ldi     mpr, (0<<INT0)              ; Disable INT0 so hand cannot be changed
    out     EIMSK, mpr
    call    SEND_HAND                   ; Send user hand to USART
                                        ; Recieve hand USART
                                        ; Taken care of by receive complete interrupt
    ldi     mpr, 4                      ; Update GAME_STAGE
    st      X, mpr

 NEXT_GAME_STAGE_4:                     ; REVEAL HANDS
    ; Start 6 second timer
    ; Display opnt hand on LCD top row
    rcall   GAME_STAGE_4                ; Do stuff for this stage
    ldi     mpr, 5                      ; Update GAME_STAGE
    st      X, mpr                      ; ^
    rjmp    NEXT_GAME_STAGE_END         ; Jump to end

 NEXT_GAME_STAGE_5:                     ; RESULT
    ; Start 6 second timer
    ; Determine the winner
    ; Print win/loss/tie msg on the LCD
    rcall   GAME_STAGE_5                ; Do stuff for this stage
    ldi     mpr, 0                      ; Update GAME_STAGE, so it wraps around and next time it
begins at the start
    st      X, mpr                      ; ^
    rjmp    NEXT_GAME_STAGE_END         ; Jump to end

 NEXT_GAME_STAGE_END:
    ; Clear interrupt queue
    rcall   BUSY_WAIT
    ldi     mpr, 0b1111_1111
    out     EIFR, mpr

    ; Restore variables
    pop     XL
    pop     XH
```

```
    pop     mpr

    ; Return from function
    ret

GAME_STAGE_0:
    ;-------------------------------------------------------------
    ; Func: Game stage 0
    ; Desc: Prints the welcome message to the LCD
    ;-------------------------------------------------------------
    ; Save variables
    push    ZH
    push    ZL

    ; Print to LCD
    ldi     ZH, high(STRING_IDLE<<1)
    ldi     ZL, low(STRING_IDLE<<1)
    rcall   LCD_ALL

    ; Restore variables
    pop     ZL
    pop     ZH

    ; Return from function
    ret

GAME_STAGE_1:
    ;-------------------------------------------------------------
    ; Func: Game stage 1
    ; Desc: Prints the ready message to the LCD and sends the ready
    ;       message via USART1
    ;-------------------------------------------------------------
    ; Save variables
    push    ZH
    push    ZL

    ; Print to LCD
    ldi     ZH, high(STRING_READY_UP<<1)
    ldi     ZL, low(STRING_READY_UP<<1)
    rcall   LCD_ALL

    ; Send ready message to other board
    rcall   SEND_READY

    ; Restore variables
    pop     ZL
    pop     ZH

    ; Return from function
    ret

GAME_STAGE_2:
    ;-------------------------------------------------------------
    ; Func: Game stage 2
    ; Desc: Starts the 6 second timer and displays the default
    ;       choice of what the user's hand is to the bottom row
    ;       of the LCD
    ;-------------------------------------------------------------
    ; Save variables
    push    mpr
    push    XH
    push    XL
    push    ZH
    push    ZL

    ; Start 6 second timer
    rcall   TIMER

    ; Print to LCD
    ldi     ZH, high(STRING_CHOOSE_HAND<<1)
    ldi     ZL, low(STRING_CHOOSE_HAND<<1)
```

```
    rcall   LCD_TOP

    ; Load in HAND_USER
    ldi     XH, high(HAND_USER)
    ldi     XL, low(HAND_USER)
    ld      mpr, X

    ; Display default hand
    cpi     mpr, SIGNAL_ROCK                ; These compares are to print
    breq    GAME_STAGE_2_ROCK               ; the correct screen to the LCD
    cpi     mpr, SIGNAL_PAPER
    breq    GAME_STAGE_2_PAPER
    cpi     mpr, SIGNAL_SCISSORS
    breq    GAME_STAGE_2_SCISSORS

    ; If no compare match, jump to end
    rjmp    GAME_STAGE_2_END

 GAME_STAGE_2_ROCK:                         ; Change to ROCK
    ; Change Data Memory variable HAND_USER
    ldi     mpr, SIGNAL_ROCK
    st      X, mpr

    ; Print to LCD
    ldi     ZH, high(STRING_ROCK<<1)        ; Point Z to string
    ldi     ZL, low(STRING_ROCK<<1)         ; ^
    rcall   LCD_BOTTOM

    ; Jump to end
    rjmp    GAME_STAGE_2_END

 GAME_STAGE_2_PAPER:                            ; Change to PAPER
    ; Change Data Memory variable HAND_USER
    ldi     mpr, SIGNAL_PAPER
    st      X, mpr

    ; Print to LCD
    ldi     ZH, high(STRING_PAPER<<1)       ; Point Z to string
    ldi     ZL, low(STRING_PAPER<<1)        ; ^
    rcall   LCD_BOTTOM

    ; Jump to end
    rjmp    GAME_STAGE_2_END

 GAME_STAGE_2_SCISSORS:                     ; Change to SCISSORS
    ; Change Data Memory variable HAND_USER
    ldi     mpr, SIGNAL_SCISSORS
    st      X, mpr

    ; Print to LCD
    ldi     ZH, high(STRING_SCISSORS<<1)    ; Point Z to string
    ldi     ZL, low(STRING_SCISSORS<<1)     ; ^
    rcall   LCD_BOTTOM

    ; Jump to end
    rjmp    GAME_STAGE_2_END

 GAME_STAGE_2_END:
    ; Restore variables
    pop     ZL
    pop     ZH
    pop     XL
    pop     XH
    pop     mpr

    ; Return from function
    ret

GAME_STAGE_4:
    ;-----------------------------------------------------------
    ; Func: Game stage 4
```

```
    ; Desc: Starts the 6 second timer and displays the opponent's hand
    ;       to the top row of the LCD
    ;-----------------------------------------------------------
    ; Save variables
    push    mpr
    push    XH
    push    XL
    push    ZH
    push    ZL

    ; Start 6 second timer
    rcall   TIMER

    ; Branch based on Opponent Hand
    ldi     XH, high(HAND_OPNT)
    ldi     XL, low(HAND_OPNT)
    ld      mpr, X

    cpi     mpr, 1
    breq    GAME_STAGE_4_ROCK
    cpi     mpr, 2
    breq    GAME_STAGE_4_PAPER
    cpi     mpr, 3
    breq    GAME_STAGE_4_SCISSORS

    ; If no compare match, branch to end
    rjmp    GAME_STAGE_4_END

 GAME_STAGE_4_ROCK:
    ; Print to LCD
    ldi     ZH, high(STRING_ROCK<<1)
    ldi     ZL, low(STRING_ROCK<<1)
    rcall   LCD_TOP

    ; Jump to end
    rjmp    GAME_STAGE_4_END

 GAME_STAGE_4_PAPER:
    ; Print to LCD
    ldi     ZH, high(STRING_PAPER<<1)
    ldi     ZL, low(STRING_PAPER<<1)
    rcall   LCD_TOP

    ; Jump to end
    rjmp    GAME_STAGE_4_END

 GAME_STAGE_4_SCISSORS:
    ; Print to LCD
    ldi     ZH, high(STRING_SCISSORS<<1)
    ldi     ZL, low(STRING_SCISSORS<<1)
    rcall   LCD_TOP

    ; Jump to end
    rjmp    GAME_STAGE_4_END

 GAME_STAGE_4_END:
    ; Restore variables
    pop     ZL
    pop     ZH
    pop     XL
    pop     XH
    pop     mpr

    ; Return from function
    ret

GAME_STAGE_5:
    ;-----------------------------------------------------------
    ; Func: Game stage 5
    ; Desc: Starts the 6 second timer and determines the winner
    ;       based on the user's hand and opponent's hand. Displays
```

```
   ;       the result on the top line of the LCD
   ;-------------------------------------------------------------
   ; Save variables
   push    mpr
   push    ilcnt
   push    XH
   push    XL
   push    ZH
   push    ZL

   ; Start 6 second timer
   rcall   TIMER

   ; Decide Won/Lost/Draw
   ; Decision is based on subtracting the opnt's hand from the user's hand
   ; Result = user - opnt
       ; Calculate result value
           ; Won   = -2, 1
           ; Lost  = -1, 2
           ; Draw  = 0
   ldi     XH, high(HAND_USER)
   ldi     XL, low(HAND_USER)
   ld      mpr, X
   ldi     XH, high(HAND_OPNT)
   ldi     XL, low(HAND_OPNT)
   ld      ilcnt, X
   sub     mpr, ilcnt              ; Result value stored in mpr

   ; Branch based on result
   cpi     mpr, -2
   breq    GAME_STAGE_5_WON
   cpi     mpr, 1
   breq    GAME_STAGE_5_WON
   cpi     mpr, -1
   breq    GAME_STAGE_5_LOST
   cpi     mpr, 2
   breq    GAME_STAGE_5_LOST
   cpi     mpr, 0
   breq    GAME_STAGE_5_DRAW

   ; If no compare match, jump to end
   rjmp    GAME_STAGE_5_END

GAME_STAGE_5_WON:
   ; Print to LCD
   ldi     ZH, high(STRING_WON<<1)
   ldi     ZL, low(STRING_WON<<1)
   rcall   LCD_TOP

   ; Jump to end
   rjmp    GAME_STAGE_5_END

GAME_STAGE_5_LOST:
   ; Print to LCD
   ldi     ZH, high(STRING_LOST<<1)
   ldi     ZL, low(STRING_LOST<<1)
   rcall   LCD_TOP

   ; Jump to end
   rjmp    GAME_STAGE_5_END

GAME_STAGE_5_DRAW:
   ; Print to LCD
   ldi     ZH, high(STRING_DRAW<<1)
   ldi     ZL, low(STRING_DRAW<<1)
   rcall   LCD_TOP

   ; Jump to end
   rjmp    GAME_STAGE_5_END

GAME_STAGE_5_END:
```

```
    ; Restore variables
    pop     ZL
    pop     ZH
    pop     XL
    pop     XH
    pop     ilcnt
    pop     mpr

    ; Return from function
    ret

STORE_HAND:
    ;-----------------------------------------------------------
    ; Func: Store hand
    ; Desc: Stores the incoming opponents hand to HAND_OPNT
    ;       Assumes that the mpr is holding the intended message
    ;       Should be called directly from MESSAGE_RECEIVE
    ;-----------------------------------------------------------
    push mpr
    push ZH
    push ZL

    ldi     ZH, high(HAND_OPNT)     ; mpr currently holds OPNT hand
    ldi     ZL, low(HAND_OPNT)
    st      Z, mpr                  ; Store the hand received
    call    PRINT_HAND

    pop ZL
    pop ZH
    pop mpr
    ret

PRINT_HAND:
    ;-----------------------------------------------------------
    ; Func: Print hand
    ; Desc: Prints the opponent's hand on the top row of the LCD
    ;-----------------------------------------------------------
    push mpr
    push XH
    push XL
    push ZH
    push ZL

    ; Branch based on Opponent Hand
    ldi     XH, high(HAND_OPNT)
    ldi     XL, low(HAND_OPNT)
    ld      mpr, X

    cpi     mpr, 1
    breq    PRINT_HAND_ROCK
    cpi     mpr, 2
    breq    PRINT_HAND_PAPER
    cpi     mpr, 3
    breq    PRINT_HAND_SCISSORS

    ; If no compare match, branch to end
    rjmp    PRINT_HAND_END

 PRINT_HAND_ROCK:
    ; Print to LCD
    ldi     ZH, high(STRING_ROCK<<1)
    ldi     ZL, low(STRING_ROCK<<1)
    rcall   LCD_TOP

    ; Jump to end
    rjmp    PRINT_HAND_END

 PRINT_HAND_PAPER:
    ; Print to LCD
    ldi     ZH, high(STRING_PAPER<<1)
    ldi     ZL, low(STRING_PAPER<<1)
```

```
    rcall   LCD_TOP

    ; Jump to end
    rjmp    PRINT_HAND_END

 PRINT_HAND_SCISSORS:
    ; Print to LCD
    ldi     ZH, high(STRING_SCISSORS<<1)
    ldi     ZL, low(STRING_SCISSORS<<1)
    rcall   LCD_TOP

    ; Jump to end
    rjmp    PRINT_HAND_END

 PRINT_HAND_END:
    ; Restore variables
    pop     ZL
    pop     ZH
    pop     XL
    pop     XH
    pop     mpr
    ret

CYCLE_HAND:
    ;-----------------------------------------------------------
    ; Func: Cycle hand
    ; Desc: Cycles the user's hand through the three available options
    ;       and updates the LCD accordingly
    ;-----------------------------------------------------------
    ; Save variables
    push    mpr
    push    XH
    push    XL
    push    ZH
    push    ZL

    ; Load in HAND_USER
    ldi     XH, high(HAND_USER)
    ldi     XL, low(HAND_USER)
    ld      mpr, X

    ; Change hand based on current hand
    cpi     mpr, SIGNAL_ROCK
    breq    CYCLE_HAND_PAPER
    cpi     mpr, SIGNAL_PAPER
    breq    CYCLE_HAND_SCISSORS
    cpi     mpr, SIGNAL_SCISSORS
    breq    CYCLE_HAND_ROCK

    ; If no compare match, jump to end
    rjmp    CYCLE_HAND_END

 CYCLE_HAND_ROCK:                           ; Change to ROCK
    ; Change Data Memory variable HAND_USER
    ldi     mpr, SIGNAL_ROCK
    st      X, mpr

    ; Print to LCD
    ldi     ZH, high(STRING_ROCK<<1)        ; Point Z to string
    ldi     ZL, low(STRING_ROCK<<1)         ; ^
    rcall   LCD_BOTTOM

    ; Jump to end
    rjmp    CYCLE_HAND_END

 CYCLE_HAND_PAPER:                          ; Change to PAPER
    ; Change Data Memory variable HAND_USER
    ldi     mpr, SIGNAL_PAPER
    st      X, mpr

    ; Print to LCD
```

```
    ldi     ZH, high(STRING_PAPER<<1)       ; Point Z to string
    ldi     ZL, low(STRING_PAPER<<1)        ; ^
    rcall   LCD_BOTTOM

    ; Jump to end
    rjmp    CYCLE_HAND_END

 CYCLE_HAND_SCISSORS:                       ; Change to SCISSORS
    ; Change Data Memory variable HAND_USER
    ldi     mpr, SIGNAL_SCISSORS
    st      X, mpr

    ; Print to LCD
    ldi     ZH, high(STRING_SCISSORS<<1)    ; Point Z to string
    ldi     ZL, low(STRING_SCISSORS<<1)     ; ^
    rcall   LCD_BOTTOM

    ; Jump to end
    rjmp    CYCLE_HAND_END

 CYCLE_HAND_END:
    ; Clear interrupt queue
    rcall   BUSY_WAIT
    ldi     mpr, 0b1111_1111
    out     EIFR, mpr

    ; Restore variables
    pop     ZL
    pop     ZH
    pop     XL
    pop     XH
    pop     mpr

    ; Return from function
    ret

TIMER:
    ;-----------------------------------------------------------
    ; Func: Timer
    ; Desc: Starts the timer for a delay of 1.5 seconds and updates
    ;       PORTB's upper 4 LEDs to show the time remaining. Once the
    ;       6 seconds are completed (4 subsequent calls TIMER) the
    ;       TOV01 flag is disabled.
    ;-----------------------------------------------------------
    ; Save variables
    push    mpr
    push    XH
    push    XL

    ldi     mpr, $48        ; Write to high byte first
    sts     TCNT1H, mpr     ; ^
    ldi     mpr, $E5        ; Write to low byte second
    sts     TCNT1L, mpr     ; ^

    ; Load in TIMER_STAGE
    ldi     XH, high(TIMER_STAGE)
    ldi     XL, low(TIMER_STAGE)
    ld      mpr, X

    ; Branch based on current TIMER_STAGE
    cpi     mpr, 4
    breq    TIMER_4
    cpi     mpr, 3
    breq    TIMER_3
    cpi     mpr, 2
    breq    TIMER_2
    cpi     mpr, 1
    breq    TIMER_1
    cpi     mpr, 0
    breq    TIMER_0
```

```
    ; If no compare match, branch to end
    rjmp    TIMER_END

 TIMER_4:                          ; Start timer
    ldi     mpr, (1<<TOIE1)     ; TOIE1 = 1 = Overflow Interrupt Enabled
    sts     TIMSK1, mpr         ; ^
    ldi     mpr, 3              ; Update TIMER_STAGE
    st      X, mpr              ; ^
    in      mpr, PINB           ; Update LEDs
    andi    mpr, 0b0000_1111    ; ^
    ori     mpr, 0b1111_0000    ; ^
    out     PORTB, mpr          ; ^
    rjmp    TIMER_END           ; Jump to end

 TIMER_3:
    ldi     mpr, 2              ; Update TIMER_STAGE
    st      X, mpr              ; ^
    in      mpr, PINB           ; Update LEDs
    andi    mpr, 0b0000_1111    ; ^
    ori     mpr, 0b0111_0000    ; ^
    out     PORTB, mpr          ; ^
    rjmp    TIMER_END           ; Jump to end

 TIMER_2:
    ldi     mpr, 1              ; Update TIMER_STAGE
    st      X, mpr              ; ^
    in      mpr, PINB           ; Update LEDs
    andi    mpr, 0b0000_1111    ; ^
    ori     mpr, 0b0011_0000    ; ^
    out     PORTB, mpr          ; ^
    rjmp    TIMER_END           ; Jump to end

 TIMER_1:
    ldi     mpr, 0              ; Update TIMER_STAGE
    st      X, mpr              ; ^
    in      mpr, PINB           ; Update LEDs
    andi    mpr, 0b0000_1111    ; ^
    ori     mpr, 0b0001_0000    ; ^
    out     PORTB, mpr          ; ^
    rjmp    TIMER_END           ; Jump to end

 TIMER_0:                          ; End timer
    ldi     mpr, (0<<TOIE1)     ; TOIE1 = 0 = Overflow Interrupt Disabled
    sts     TIMSK1, mpr         ; ^
    ldi     mpr, 4              ; Update TIMER_STAGE, so it wraps around and next time it begins
at the start
    st      X, mpr              ; ^
    in      mpr, PINB           ; Update LEDs
    andi    mpr, 0b0000_1111    ; ^
    ori     mpr, 0b0000_0000    ; ^
    out     PORTB, mpr          ; ^
    rcall   NEXT_GAME_STAGE     ; Update GAME_STAGE
    rjmp    TIMER_END           ; Jump to end

 TIMER_END:
    ; Restore variables
    pop     XL
    pop     XH
    pop     mpr

    ; Return from function
    ret

BUSY_WAIT:
    ;-----------------------------------------------------------------
    ; Func: BUSY_WAIT
    ; Desc: A  wait loop that loops enough times to cause a delay
    ;       of 15ms. The allowed limit for busy waiting in this lab
    ;       is 150ms. This is used only to clear the interrupt queue
    ;       and prevent duplicate calls.
    ;-----------------------------------------------------------------
```

```
    ; Save variables
    push    mpr
    push    ilcnt
    push    olcnt

    ldi     mpr, 15
 BUSY_WAIT_LOOP:
    ldi     olcnt, 224      ; Load olcnt register
 BUSY_WAIT_OLOOP:
    ldi     ilcnt, 237      ; Load ilcnt register
 BUSY_WAIT_ILOOP:
    dec     ilcnt           ; Decrement ilcnt
    brne    BUSY_WAIT_ILOOP ; Continue Inner Loop
    dec     olcnt           ; Decrement olcnt
    brne    BUSY_WAIT_OLOOP ; Continue Outer Loop
    dec     mpr
    brne    BUSY_WAIT_LOOP

    ; Restore variables
    pop     olcnt
    pop     ilcnt
    pop     mpr

    ; Return from function
    ret

;***********************************************************
;*  Stored Program Data
;***********************************************************
STRING_IDLE:
        .DB "Welcome!        Please press PD7"
STRING_READY_UP:
        .DB "Ready. Waiting  for the opponent"
STRING_CHOOSE_HAND:
        .DB "Game start      "
STRING_WON:
        .DB "You Win!        "
STRING_LOST:
        .DB "You Lose!       "
STRING_DRAW:
        .DB "Draw            "
STRING_ROCK:
        .DB "Rock            "
STRING_PAPER:
        .DB "Paper           "
STRING_SCISSORS:
        .DB "Scissor         "

;***********************************************************
;*  Data Memory Allocation
;***********************************************************
.dseg
.org    $0200
TIMER_STAGE:        ; TIMER_STAGE value for timer loop and LED display
    .byte 1
GAME_STAGE:         ; Indicates the current stage the game is in
    .byte 1
HAND_OPNT:          ; Opponent choice: Rock / Paper / Scissors
    .byte 1
HAND_USER:          ; User choice: Rock / Paper / Scissors
    .byte 1
READY_OPNT:         ; Opponent ready
    .byte 1
READY_USER:         ; User ready
    .byte 1

;***********************************************************
;*  Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"
```