ECE 375 Lab 7

Remotely communicated Rock Paper Scissors

Lab Time: Friday 2 PM - 3:50 PM

Student 1: Winnie Woo
Student 2: Joseph Borisch

TA Signature

1 Introduction

The purpose of this lab is to use two separate AVR boards to play rock, paper, scissors by implementing knowledge of the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) module on the microcontroller. As well as instead of using the Wait function in the BumpBot program, create our own 16-bit timer/counter to create a 1.5 second time delay.

2 Program Overview

This program will play rock, paper, scissors between two AVR boards. The LCD display will be used to prompt messages to initiate the game. The 4 LEDs are used a countdown timer, with each LED being 1.5 seconds. PD7 will be used at the ready button to start the game while PD4 will be used to cycle through the gestures in the order of rock, paper, scissor.
Besides the standard INIT and MAIN routines within the program, there are additional subroutines to handle the entire functionality of the game such as transmitting and receiving the signal to and from the AVR boards.

3 Initialization Routine

The INIT routine initializes the stack pointer, PORT B and PORT D for output. We are also initializing the LCD display and the interrupt sense control (EICRA) trigger state to falling edge and configuring the external interrupt mask (EIMSK) to enable INT0, INT1 and INT3 and turn on the global interrupt. USART1 is also initialized to set the baud rate to 2400 bits per second with double data rate, as well as TCCR1A and TCCR1B for timer and counter.

4 Main Routine

The Main routine prints the welcome message on the LCD display.

5 GAME READY

This function triggers the interrupt when PD7 is pressed and updates the game status to display a welcome message on the LCD.

6 TRANSMIT SIGNAL

This function handles transmitting data to the other board

7 RECEIVE SIGNAL

This function handles receiving data from the other board

8 START GAME

This function tracks the amount of time passed for the game and the results of the game.

9 END GAME

This function handles the post game operations like resetting the registers and clearing the LCD display.

10 GAME RESULTS

This function handles the results of the game by comparing which gesture each user has selected.

11 WIN GAME

This function handles the procedures for the player winning the game and prints the subsequent win message on the LCD display.

12 LOST GAME

This function handles the procedures for the player losing the game and print the subsequent lose message on the LCD display.

13 TIE GAME

This function handles the procedures for both players tying the game and prints the subsequent tie message on the LCD display.

14 CYCLE HAND

This function handles the interrupt to change the selected gesture for each player.

15 FULL GAME WAIT

This function handles the LED time management for the game, by setting the number of loops to 3 and offsetting from max to get the right delay. Then within the check loop, determine if it is done counting and check if we need to break out of the loop.

16 Difficulties

One of the main difficulties was having to use two AVR boards between the both of us because it made it difficult to test the code separately. We also don't live on campus, so it was not possible to meet up due to the distance and most communication took place online.

17 Conclusion

In conclusion, this lab was by far the most difficult to implement due to having to share our AVR boards but it was a great way to learn about how to configure the USART1 module on the boards.

18 Source Code

```
;***********************************************************
;*
;* This is the TRANSMIT skeleton file for Lab 7 of ECE 375
;*
;*   Rock Paper Scissors
;*   Requirement:
;*   1. USART1 communication
;*   2. Timer/counter1 Normal mode to create a 1.5-sec delay
;***********************************************************
;*
;*  Author 1: JOSEPH BORISCH
;*  Author 2: WINNIE WOO
;*   Date: 11/26/2022
;*
;***********************************************************

.include "m32U4def.inc"      ; Include definition file

;***********************************************************
;*  Internal Register Definitions and Constants
;***********************************************************
.def   mpr = r16             ; Multi-Purpose Register
.def  oppReadyReg = r12  ; Register tracking ready state of opponent
.def  userReadyReg = r13    ; Register tracking ready state of user
.def  receiveReg = r18   ; Register used for receiving data from other board
.def  transmitReg = r19  ; Register used for transmitting data to other board
.def  oppHandSelect = r14   ; Register used to track the hand sleection of opponent
   (1=rock,2=paper,3=scissor)
.def  handSelect = r24   ; Register used to track the hand sleection of user
   (1=rock,2=paper,3=scissor)
.def  olcnt = r23        ; Register tracking the inner loop for the wait timer
.def  ilcnt = r17        ; Register tracking the outer loop for the wait timer

; Use this signal code between two boards for their game ready
```

```
.equ   SendReady = 0b11111111

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

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

.org $0002
     rcall GAME_READY      ; Interrupt to start the game
     reti

.org $0004
     rcall CYCLE_HAND      ; Interrupt to cycle through hand selection
     reti

.org $0032
     rcall RECEIVE_SIGNAL  ; interrupt to indicate opponent is transmitting data
     reti

.org   $0056                  ; End of Interrupt Vectors

;***********************************************************
;*  Program Initialization
;***********************************************************
INIT:
   ;Stack Pointer (VERY IMPORTANT!!!!)
   ldi mpr, low(RAMEND) ; Low byte of END SRAM addr
   out SPL, mpr       ; Write byte to SPL
   ldi mpr, high(RAMEND) ; high byte of END sram addr
   out SPH, mpr       ; Write byte to SPH
   ;I/O Ports
   ; 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
   ;USART1
      ;Set baudrate at 2400bps
```

```
   ;Enable receiver and transmitter
   ;Set frame format: 8 data bits, 2 stop bits
   ldi     mpr, 0b00100010    ; UDRE1, U2X1
   sts     UCSR1A, mpr      ; USART data register empty, double USART transmission speed

   ldi     mpr, 0b10011000    ; RXCIE, RXEN, TXEN, UCSZ
   sts     UCSR1B, mpr      ; RX complete interrupt enable, Receiver enable,
      transmitter enable, character size

   ldi     mpr, 0b00001110    ; UMSEL11:10, UPM11:10, USBS1, UCSZ11:10, UCPO
   sts     UCSR1C, mpr      ; USART mode select, parity mode, bit select, character
      size, clock polarity

   ; Baud rate 2400bps with double
   ldi     mpr, high(416)   ; calculated UBBR1 for given baudrate and clock frequency
      8MHz
   sts     UBRR1H, mpr
   ldi     mpr, low(416)    ; calculated UBBR1 for given baudrate and clock frequency
      8MHz
   sts     UBRR1L, mpr

      ;TIMER/COUNTER1
      ;Set Normal mode
      ; Configure 16-bit Timer/Counter 1A and 1B
      ldi     mpr, 0b11110000       ; Normal mode (WGM11:10)
      sts     TCCR1A, mpr        ; / inverting mode (COM1A1:COM1A0 and COM1B1:COM1B0)
      ldi     mpr, 0b00000101       ; Normal mode (WGM13:WGM12)
      sts     TCCR1B, mpr        ; / prescale=1024 (CS12:CS10)
      ;Other
      ; Initialize LCD Display
      rcall LCDInit
      rcall LCDClr
      rcall  LCDBacklightOn


;***********************************************************
;*  Main Program
;***********************************************************
MAIN:
      clr     mpr
      add     mpr, oppReadyReg         ; add two player ready registers
      add     mpr, userReadyReg
      cpi     mpr, $02              ; compare two player ready registers, if both
         equal to 1, then start game
      breq  START_GAME

      ldi     YL, 0x00              ; initialize address pointing to first line of low
         byte on LCD
```

```
      ldi     YH, 0x01              ; initialize address pointing to first line of
         high byte on LCD

      ldi     ZL, low(STRING_PREGAME_1<<1) ; load lo string into Z
      ldi     Zh, high(STRING_PREGAME_1<<1) ; load high string into Z
   PREGAME_LOOP:
         lpm mpr, Z+                ; mpr gets lo Z
         st Y+, mpr                 ; low Y gets lo Z
         cpi ZL, low(STRING_PREGAME_2<<1) ; compare the value of the low_end after
            shifting one bit
         brne  PREGAME_LOOP
      rcall LCDWrite                ; Write pre game message to LCD
      rjmp  MAIN

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

;-----------------------------------------------------------
; Func: GAME READY
; Desc: Interrupt triggered when PD7 is pressed. Updates
;     game status and sends ready message to other board
;-----------------------------------------------------------
GAME_READY: ; Begin a function with a label

      ldi     mpr,$FF    ; clear queued interrupts
      out     EIFR, mpr

      ldi     YL, 0x00      ; initialize address pointing to first line of low byte on
         LCD
      ldi     YH, 0x01      ; initialize address pointing to first line of high byte on
         LCD

      ldi     ZL, low(STRING_READY_1<<1) ; load lo string into Z
      ldi     Zh, high(STRING_READY_1<<1) ; load high string into Z
   READY_LOOP:
         lpm mpr, Z+             ; mpr gets lo Z
         st Y+, mpr              ; low Y gets lo Z
         cpi ZL, low(STRING_READY_2<<1) ; compare the value of the low_end after
            shifting one bit
         brne  READY_LOOP
      rcall LCDWrite

      ldi     mpr, $01
      mov     userReadyReg, mpr

      ldi mpr, SendReady
      mov  transmitReg, mpr
      rcall TRANSMIT_SIGNAL
```

```
     ret

;-----------------------------------------------------------
; Func: TRANSMIT SIGNAL
; Desc: Handles transmitting data to other board
;-----------------------------------------------------------
TRANSMIT_SIGNAL: ; Begin a function with a label

     lds     mpr, UCSR1A      ; read the data from UDR1
     sbrs  mpr, UDRE1    ; loop if UDR1 still has data
     rjmp  TRANSMIT_SIGNAL

     sts     UDR1, transmitReg ; Send the data to the other board



     ldi     mpr, 0b00000010    ; enable PD4 for selecting hand, disable PD7
     out     EIMSK, mpr

     ret

;-----------------------------------------------------------
; Func: RECEIVE SIGNAL
; Desc: Handles receiving data from other board
;-----------------------------------------------------------
RECEIVE_SIGNAL: ; Begin a function with a label

     lds     receiveReg, UDR1

     cpi     receiveReg, $FF    ; check if data is to start game
     breq  OPP_READY

     jmp     RECEIVE_SIGNAL_END    ; if not game start up, finished receiving

   OPP_READY:
     ldi    mpr, $01
     mov     oppReadyReg, mpr
     sbrs  userReadyReg, $01  ; check if user is ready
     jmp     RECEIVE_SIGNAL_END    ; if user not ready, jump to end

     ldi     mpr, 0b00000010    ; enable PD4 for selecting hand, disable PD7
     out     EIMSK, mpr

   RECEIVE_SIGNAL_END:

     ret

;-----------------------------------------------------------
; Func: START GAME
```

```
; Desc: Main body for the game itself. Tracks the time
;    passed and handles results of game
;-----------------------------------------------------------
START_GAME:

     ldi     YL, 0x00      ; initialize address pointing to first line of low byte on
        LCD
     ldi     YH, 0x01      ; initialize address pointing to first line of high byte on
        LCD

     ldi     ZL, low(STRING_START_1<<1) ; load lo string into Z
     ldi     Zh, high(STRING_START_1<<1) ; load high string into Z
   START_GAME_LOOP_1:
        lpm mpr, Z+              ; mpr gets lo Z
        st Y+, mpr               ; low Y gets lo Z
        cpi ZL, low(STRING_START_2<<1) ; compare the value of the low_end after
           shifting one bit
        brne  START_GAME_LOOP_1
     ldi     ZL, low(STRING_ROCK_1<<1) ; load lo string into Z
     ldi     Zh, high(STRING_ROCK_1<<1) ; load high string into Z
     ldi   YL, $10               ;initialize address pointing to second line of low
        byte on LCD
     ldi   YH, $01               ;initialize address pointing to second line of high
        byte on LCD
   START_GAME_LOOP_2:
        lpm mpr, Z+
        st Y+, mpr
        cpi ZL, low(STRING_ROCK_2<<1) ; compare the value of the low_end after shift
           one bit, if it is reached that, the break
        brne  START_GAME_LOOP_2

     clr     userReadyReg
     clr     oppReadyReg

     rcall FULL_GAME_WAIT     ; call wait routine to count down time on LEDS

     lds     mpr, UCSR1A         ; enable transmit of data
     sbr     mpr, UDRE1
     sts     UCSR1A, mpr

     mov     transmitReg, handSelect  ; put user move in trasmit register and send to
        other board
     rcall TRANSMIT_SIGNAL

     ;
     mov     oppHandSelect, handSelect
     rcall TRANSMIT_SIGNAL
     ;display opponents move to second line of LCD
     mov     oppHandSelect, receiveReg
```

```
   ldi     mpr, $01
   cp    oppHandSelect, mpr
   breq  OPP_ROCK_SELECT

   ldi     mpr, $02
   cp    oppHandSelect, mpr
   breq  OPP_PAPER_SELECT

   ldi     mpr, $03
   cp    oppHandSelect, mpr
   breq  OPP_SCISSOR_SELECT

OPP_ROCK_SELECT:
     ldi     YL, 0x10      ; initialize address pointing to first line of low byte
        on LCD
     ldi     YH, 0x01      ; initialize address pointing to first line of high byte
        on LCD

     ldi     ZL, low(STRING_ROCK_1<<1) ; load lo string into Z
     ldi     Zh, high(STRING_ROCK_1<<1) ; load high string into Z
  OPP_ROCK_MSG:
       lpm mpr, Z+             ; mpr gets lo Z
       st Y+, mpr              ; low Y gets lo Z
       cpi ZL, low(STRING_ROCK_2<<1) ; compare the value of the low_end after
          shifting one bit
       brne  OPP_ROCK_MSG
     rcall LCDWrLn2

OPP_PAPER_SELECT:
     ldi     YL, 0x10      ; initialize address pointing to first line of low byte
        on LCD
     ldi     YH, 0x01      ; initialize address pointing to first line of high byte
        on LCD

     ldi     ZL, low(STRING_PAPER_1<<1) ; load lo string into Z
     ldi     Zh, high(STRING_PAPER_1<<1) ; load high string into Z
  OPP_PAPER_MSG:
       lpm mpr, Z+             ; mpr gets lo Z
       st Y+, mpr              ; low Y gets lo Z
       cpi ZL, low(STRING_PAPER_2<<1) ; compare the value of the low_end after
          shifting one bit
       brne  OPP_PAPER_MSG
     rcall LCDWrLn2

OPP_SCISSOR_SELECT:
     ldi     YL, 0x10      ; initialize address pointing to first line of low byte
        on LCD
```

```
        ldi     YH, 0x01      ; initialize address pointing to first line of high byte
           on LCD

        ldi     ZL, low(STRING_SCISSOR_1<<1) ; load lo string into Z
        ldi     Zh, high(STRING_SCISSOR_1<<1) ; load high string into Z
     OPP_SCISSOR_MSG:
          lpm mpr, Z+             ; mpr gets lo Z
          st Y+, mpr              ; low Y gets lo Z
          cpi ZL, low(STRING_SCISSOR_2<<1) ; compare the value of the low_end after
             shifting one bit
          brne  OPP_SCISSOR_MSG
        rcall LCDWrLn2
     jmp     END_GAME   ; No return, go straight to the end game procedure
;-----------------------------------------------------------
; Func: END GAME
; Desc: Handles post game operation such as resetting
;    registers and clearing the LCD
;-----------------------------------------------------------
END_GAME:

   rcall GAME_RESULT  ; display game results to LCD

   ldi     mpr, 0b00000001    ; Configure interrupt for PD7
   out     EIMSK, mpr

   rcall LCDClr          ; clr LCD

   clr     handSelect      ; reset hand selectrion register

   clr     userReadyReg
   clr     oppReadyReg

   ret

;-----------------------------------------------------------
; Func: GAME RESULTS
; Desc: Handles functionality of the results of the game
;-----------------------------------------------------------
GAME_RESULT:

     cpi     handSelect, $01       ; compare user hand to see what is chosen
     breq  USER_SELECT_ROCK

     cpi     handSelect, $02       ; compare user hand to see what is chosen
     breq  USER_SELECT_PAPER

     cpi     handSelect, $03       ; compare user hand to see what is chosen
     breq  USER_SELECT_SCISSOR
   USER_SELECT_ROCK:
```

```
        ldi     mpr, $01
        cp    oppHandSelect, mpr
        breq  TIE_GAME
        ldi     mpr, $02
        cp    oppHandSelect, mpr
        breq  LOST_GAME
        ldi     mpr, $03
        cp    oppHandSelect, mpr
        breq  WIN_GAME
        ret
   USER_SELECT_PAPER:
        ldi     mpr, $01
        cp    oppHandSelect, mpr
        breq  WIN_GAME
        ldi     mpr, $02
        cp    oppHandSelect, mpr
        breq  TIE_GAME
        ldi     mpr, $03
        cp    oppHandSelect, mpr
        breq  LOST_GAME
        ret
   USER_SELECT_SCISSOR:
        ldi     mpr, $01
        cp    oppHandSelect, mpr
        breq  LOST_GAME
        ldi     mpr, $02
        cp    oppHandSelect, mpr
        breq  WIN_GAME
        ldi     mpr, $03
        cp    oppHandSelect, mpr
        breq  TIE_GAME
        ret
      ret

;-----------------------------------------------------------
; Func: WIN GAME
; Desc: Handles procedure for player winning game.
;     Prints win message
;-----------------------------------------------------------
WIN_GAME:
     ldi     YL, 0x00      ; initialize address pointing to first line of low byte on
        LCD
     ldi     YH, 0x01      ; initialize address pointing to first line of high byte on
        LCD

     ldi     ZL, low(STRING_WINMESSAGE_1<<1) ; load lo string into Z
     ldi     Zh, high(STRING_WINMESSAGE_1<<1) ; load high string into Z
   WIN_GAME_LOOP:
        lpm mpr, Z+             ; mpr gets lo Z
```

```
        st Y+, mpr              ; low Y gets lo Z
        cpi ZL, low(STRING_WINMESSAGE_2<<1) ; compare the value of the low_end after
            shifting one bit
        brne WIN_GAME_LOOP
      rcall LCDWrLn1
      ret
;-----------------------------------------------------------
; Func: LOST GAME
; Desc: Handles procedure for player losing game.
;    Prints lost message
;-----------------------------------------------------------
LOST_GAME:
      ldi     YL, 0x00      ; initialize address pointing to first line of low byte on
          LCD
      ldi     YH, 0x01      ; initialize address pointing to first line of high byte on
          LCD

      ldi     ZL, low(STRING_LOSEMESSAGE_1<<1) ; load lo string into Z
      ldi     Zh, high(STRING_LOSEMESSAGE_1<<1) ; load high string into Z
   LOSE_GAME_LOOP:
        lpm mpr, Z+             ; mpr gets lo Z
        st Y+, mpr              ; low Y gets lo Z
        cpi ZL, low(STRING_LOSEMESSAGE_2<<1) ; compare the value of the low_end after
            shifting one bit
        brne LOSE_GAME_LOOP
      rcall LCDWrLn1
      ret
;-----------------------------------------------------------
; Func: TIE GAME
; Desc: Handles procedure for player tying game.
;    Prints tie message
;-----------------------------------------------------------
TIE_GAME:
      ldi     YL, 0x00      ; initialize address pointing to first line of low byte on
          LCD
      ldi     YH, 0x01      ; initialize address pointing to first line of high byte on
          LCD

      ldi     ZL, low(STRING_DRAWMESSAGE_1<<1) ; load lo string into Z
      ldi     Zh, high(STRING_DRAWMESSAGE_1<<1) ; load high string into Z
   TIE_GAME_LOOP:
        lpm mpr, Z+             ; mpr gets lo Z
        st Y+, mpr              ; low Y gets lo Z
        cpi ZL, low(STRING_DRAWMESSAGE_2<<1) ; compare the value of the low_end after
            shifting one bit
        brne TIE_GAME_LOOP
      rcall LCDWrLn1
      ret
```

```
;-----------------------------------------------------------
; Func: CYCLE HAND
; Desc: Interrupt to change the selected hand for the player
;-----------------------------------------------------------
CYCLE_HAND:

     inc     handSelect

     cpi     handSelect, $01
     breq  ROCK_HAND_SELECT

     cpi     handSelect, $02
     breq  PAPER_HAND_SELECT

     cpi     handSelect, $03
     breq  SCISSOR_HAND_SELECT

     ROCK_HAND_SELECT:

          ldi     YL, 0x10      ; initialize address pointing to first line of low
              byte on LCD
          ldi     YH, 0x01      ; initialize address pointing to first line of high
              byte on LCD

          ldi     ZL, low(STRING_ROCK_1<<1) ; load lo string into Z
          ldi     Zh, high(STRING_ROCK_1<<1) ; load high string into Z
        MSG_USER_ROCK_SELECT:
          lpm mpr, Z+             ; mpr gets lo Z
          st Y+, mpr              ; low Y gets lo Z
          cpi ZL, low(STRING_ROCK_2<<1) ; compare the value of the low_end after
              shifting one bit
          brne  MSG_USER_ROCK_SELECT

          rcall LCDWrLn2
          ldi     mpr, 0b00000001
          out     EIFR, mpr
          ret
     PAPER_HAND_SELECT:
          ldi     YL, 0x10      ; initialize address pointing to first line of low
              byte on LCD
          ldi     YH, 0x01      ; initialize address pointing to first line of high
              byte on LCD

          ldi     ZL, low(STRING_PAPER_1<<1) ; load lo string into Z
          ldi     Zh, high(STRING_PAPER_1<<1) ; load high string into Z
        MSG_USER_PAPER_SELECT:
          lpm mpr, Z+             ; mpr gets lo Z
          st Y+, mpr              ; low Y gets lo Z
```

```
            cpi ZL, low(STRING_PAPER_2<<1) ; compare the value of the low_end after
               shifting one bit
            brne  MSG_USER_PAPER_SELECT

         rcall LCDWrLn2
         ldi     mpr, 0b00000001
         out     EIFR, mpr
         ret
      SCISSOR_HAND_SELECT:
         ldi     YL, 0x10      ; initialize address pointing to first line of low
             byte on LCD
         ldi     YH, 0x01      ; initialize address pointing to first line of high
             byte on LCD

         ldi     ZL, low(STRING_SCISSOR_1<<1) ; load lo string into Z
         ldi     Zh, high(STRING_SCISSOR_1<<1) ; load high string into Z
        MSG_USER_SCISSOR_SELECT:
            lpm mpr, Z+             ; mpr gets lo Z
            st Y+, mpr              ; low Y gets lo Z
            cpi ZL, low(STRING_SCISSOR_2<<1) ; compare the value of the low_end
               after shifting one bit
            brne  MSG_USER_SCISSOR_SELECT

         rcall LCDWrLn2
         ldi     mpr, 0b00000001
         out     EIFR, mpr
         ret
      ret

;-----------------------------------------------------------
; Func: FULL GAME WAIT
; Desc: Handles the time management for the game. Tracks
;     the time on 4 LEDS
;-----------------------------------------------------------
FULL_GAME_WAIT:

      push  mpr
      in    mpr, SREG
      push  mpr
      push  ilcnt
      push  olcnt

      ldi     olcnt, 4

      ldi     mpr, $F0
      out     PORTB, mpr

      OUTER_ONE_AND_HALF_SEC_WAIT:
        ldi     ilcnt, 3
```

```
        INNER_ONE_AND_HALF_SEC_WAIT:
           ldi     mpr, $85
           sts     TCNT1H, mpr
           ldi     mpr, $ED
           sts     TCNT1L, mpr

           CHECK_LOOP:
              in    mpr, TIFR1
              andi  mpr, 0b00000001
              brne  CHECK_LOOP

           ldi     mpr, 0b00000001
           out     TIFR1, mpr

           dec     ilcnt

           cpi     ilcnt, $00
           brne  INNER_ONE_AND_HALF_SEC_WAIT

        in mpr, PORTB
        lsr mpr
        andi  mpr, 0xF0
        out     PORTB, mpr

        dec     olcnt

        cpi     olcnt, $00
        brne  OUTER_ONE_AND_HALF_SEC_WAIT

     pop     olcnt
     pop     ilcnt
     pop     mpr
     out     SREG, mpr
     pop     mpr

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

;-----------------------------------------------------------
; An example of storing a string. Note the labels before and
; after the .DB directive; these can help to access the data
;-----------------------------------------------------------
STRING_PREGAME_1:
    .DB    "Welcome!      Please Press PD7"   ; Declaring data in ProgMem
STRING_PREGAME_2:

STRING_READY_1:
```

```
    .DB     "READY, Waiting for the Opponent "  ; Declaring data in ProgMem
STRING_READY_2:

STRING_START_1:
    .DB     "GAME START     "      ; Declaring data in ProgMem
STRING_START_2:

STRING_ROCK_1:
    .DB     "Rock           "      ; Declaring data in ProgMem
STRING_ROCK_2:

STRING_PAPER_1:
    .DB     "Paper          "      ; Declaring data in ProgMem
STRING_PAPER_2:

STRING_SCISSOR_1:
    .DB     "Scissors       "      ; Declaring data in ProgMem
STRING_SCISSOR_2:

STRING_LOSEMESSAGE_1:
    .DB     "You Lose!      "      ; Declaring data in ProgMem
STRING_LOSEMESSAGE_2:

STRING_WINMESSAGE_1:
    .DB     "You Win!       "      ; Declaring data in ProgMem
STRING_WINMESSAGE_2:

STRING_DRAWMESSAGE_1:
    .DB     "Draw!          "      ; Declaring data in ProgMem
STRING_DRAWMESSAGE_2:

;***********************************************************
;* Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"    ; Include the LCD Driver
```