ECE 375 Lab 7

Remotely communicated Rock Paper Scissors

Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

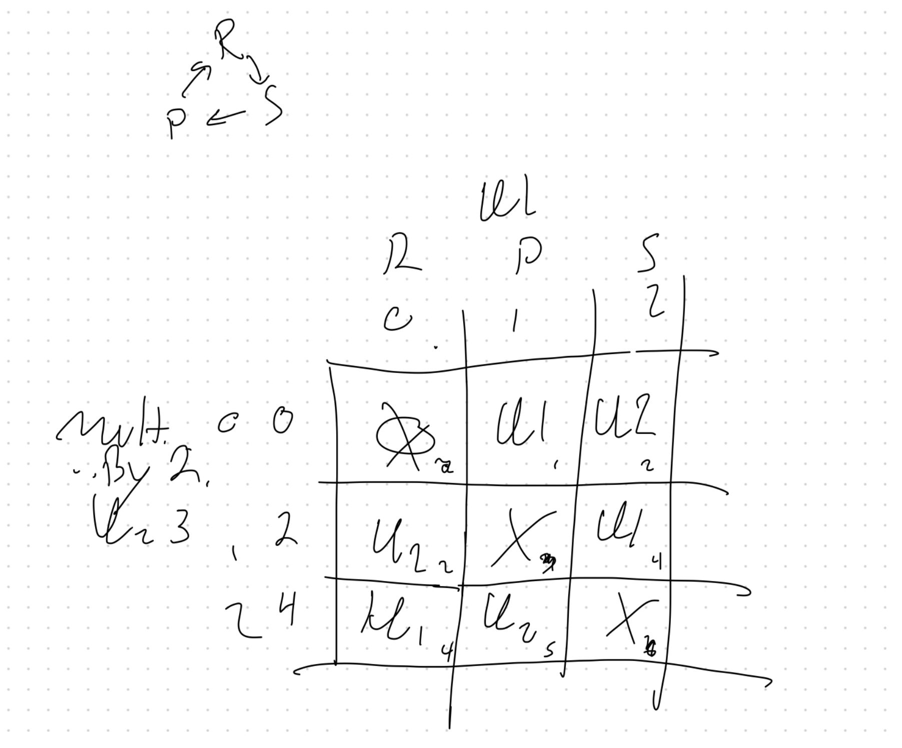
TA Signature

1 Introduction

This is the final lab for ECE 375 and as such it necessitates a challenge. The major task for this lab was to, with 2 Benny boards, have them communicate over USART to play a game of rock paper scissors with each other. This allowed the students to apply certain knowledge gained about timers/counters, and how they can apply when sending or receiving data.

2 Design

The design for this Lab can be seen below in the flow chart and the derision tree that we built.

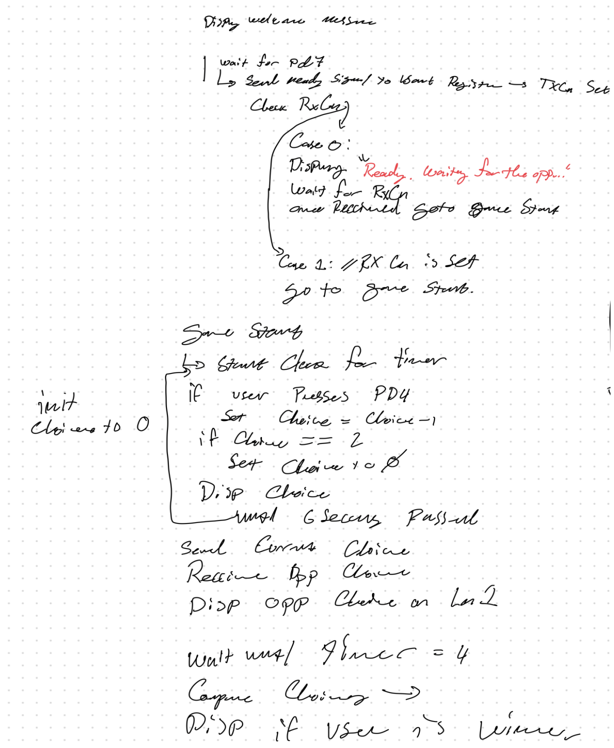


3 Assembly Overview

As for the Assembly program an overview can be seen below.

3.1 Internal Register Definitions and Constants

many different internal registers were used for this lab. Those include a multi-purpose register, 2 count registers (inner and outer) for counting lines on the LCD, a zero register to compare to, userChoice to hold the user rock paper scissors choice, tmrcnt a timer count register, button; a register to denounce the button, and oldbut, somewhere to hold the old button signal label. Additionally send ready was initialized to $FF and the LCD data memory locations were saved to their corresponding names.



3.2 Interrupt Vectors

no interrupt vectors were included in this assignment.

3.3 Initialization Routine

The initialization routine was quite complex as it had to setup the USART registers and enable the use of USART. All of the standard initialization parts are there, such as the initialization of the stack pointer, port B for output, Port D for input and the LCD. In addition to setting the baud rate for the USART, an additional timer counter was used to count on the LEDs.

3.4 Main Routine

The main program is quite simple as it first writes the welcome text to the LCD, next it querys for button 7. Once button 7 is pressed it makes sure it is connected to another board over USART. Once the key has been sent and received by both parties the mainprogram jumps into the game start routine:

3.5 Subroutines

3.6 USART_TX

This subroutine is responsible for sending the USART signal cleanly

3.7 USART_RX

This subroutine is responsible for receiving a sent USART signal.

3.7.1 GAMESTART

This subroutine takes care of most of the actual gameplay that occurs while the two boards are communicating. this includes getting the users choice, looping trough timers. Changing the LEDs, and checking to see who won.

3.7.2 WRITESCREEN

The Writescreen subroutine writes the queried data to the LCD. the input for this subroutine is 2 registers, those being ilcnt and olcnt.

3.7.3 STARTTIMER

This function starts the 1.5 second timer and will end and set the timer flag when it finishes.

3.8 SMALLWAIT

This function waits a small amount of time to allow for debouching of the button input of the methods outlined above.

4 Testing

Tested Each button press and compared to external calculations.

Case	Expected	Actual meet expected
d4	rock-¿ paper-¿ scissors	✓
d5	nothing	✓
d6	nothing	✓
d7	starts the game	✓

Table 1: Assembly Testing Cases

5 Study Questions

1. NONE!

6 Difficulties

This lab was quite difficult, mainly due to the fact of how verbose the manual was, and also the fact that the manual has some example code that does not work when applied directly. This caused major problems for us however we eventually found what we needed to do to fix it.

7 Conclusion

In conclusion, this lab allowed the student to experiment with sending data over USART and allowed us to play a game using 2 different micro controllers. It was challenging however it also taught many students how not to procrastinate and plan their projects out so they can turn them in on time.

8 Source Code

Listing 1: Assembely Script

```
;***********************************************************
;*
;*  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: Astrid Delestine & Lucas Plaisted
;*     Date: 3/13/2023
;*
;***********************************************************

.include "m32U4def.inc"         ; Include definition file

;***********************************************************
;*  Internal Register Definitions and Constants
;***********************************************************
;DO NOT USE 20-22
.def    mpr = r16               ; Multi-Purpose Register
.def    ilcnt = r18
.def    olcnt = r19
.def    zero = r2
.def    userChoice = r17
.def    tmrcnt = r15
.def    button = r13
.def    oldbut = r14
; Use this signal code between two boards for their game ready
.equ    SendReady = 0b11111111
.equ    lcd1L = 0x00            ; Make LCD Data Memory locations constants
.equ    lcd1H = 0x01
.equ    lcd2L = 0x10            ; lcdL1 means the low part of line 1's location
.equ    lcd2H = 0x01            ; lcdH2 means the high part of line 2's locatio
;***********************************************************
;*  Start of Code Segment
;***********************************************************
.cseg                           ; Beginning of code segment

;***********************************************************
;*  Interrupt Vectors
```

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


.org    $0056                   ; End of Interrupt Vectors

;***********************************************************
;*  Program Initialization
;***********************************************************
INIT:
    ; Most important thing possible!!!!!
        clr     zero
        clr     userChoice
        clr tmrcnt
            ;)
    ; Initialize the 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

    ; Initialize Port B for output
        ldi     mpr, $F0        ; 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

    ;init the LCD
        rcall LCDInit
        rcall LCDBacklightOn
        rcall LCDClr


/*  I/O Ports
    ;USART1
        Need to set USCR1B and C
        B: x00xxx00 -> 0b0_00_1_1_0_00
            2:   USCZ12
            3:   TXEN1: Transmitter enable
```

```
                4:   RXEN1: Receiver enable
                7:   RXCIE1: Receive complete interrupt enable flag,
                             enable if using interrupts
            C: xxxxxxxx -> 0b00_00_1_11_0
                0:   UPOL1: Clock Polarity
                2-1: USCZ11 and USCZ10
                3:   USBS1 stop bit select
                5-4: UPM1 parity mode
                7-6: UMSEL1 USART mode select
            x's␣are␣bits␣that␣need␣to␣be␣set
␣␣␣␣␣␣␣␣0's are status bits, no setting, only reading
        USCZ1:  011 for 8 bit
        UMSEL1: 00 for asynchronous
        UMP1:   00 for disbled
        USBS1:  1 for 2-bit
        USPOL1: 0 for rising edge
*/
            ; Set baudrate at 2400bps, double data rate
            ; Asynchronous Double Speed mode eq:

/*  UBRR1 = fOSC/(8*BAUD)
            fOSC is just the system clock, so 8MHz
            BAUD is 2400
        UBRR1 = (8*10^6)/(8*2400) = 10^6/2400 = 416.66
        about 417 or 0b1_10100001
*/
                ldi mpr, 0b00000001
                sts UBRR1H, mpr
                ldi mpr, 0b10100001
                sts UBRR1L, mpr

                ldi mpr, 0b0_00_1_1_0_00
                sts UCSR1B, mpr
                ldi mpr, 0b00_00_1_11_0
                sts UCSR1C, mpr

        ;TIMER/COUNTER1
            ;Set Normal mode, WGM13:0 = 0b000
/*
TIMER MATH
        Need 1.5sec delay
        Max count of 2^16-1 = 65,535
        65,535/1.5 = 43690 counts/sec ideal, lower is okay
        CPU @ 8MHz = 8*10^6 counts/sec
        8*10^6/prescale <= 43690
        prescale >= 8*10^6/43690
```

```
     prescale >= 183
     prescale should be 256 :)
    WGM1 = 0b100
     at 256 prescale how much we counting?
    x/(8MHz/256) = 1.5s
    x = 1.5s(8Mhz/256) = 46,875
    so we need to load 65535-46875 = 18660
    into the counter in order to have it count for the
    correct amount of time

    In two 8-bit numbers, that value is
    High: 0b01001000
    Low:  0b11100100
*/
    ; Configure 16-bit Timer/Counter 1A and 1B
            ; TCCRIA Bits:
                ; 7:6 - Timer/CounterA compare mode, 00 = disabled
                ; 5:4 - Timer/CounterB compare mode, 00 = disabled
                ; 3:2 - Timer/CounterC compare mode, 00 = disabled
                ; 1:0 - Wave gen mode low half, 00 for normal mode
            ldi mpr, 0b00_00_00_00
            sts TCCR1A, mpr
            ; TCCRIB Bits:
                ; 7:5 - not relevant, 0's
                ; 4:3 - Wave gen mode high half, 00 for normal
                ; 2:0 - Clock selection, 100 = 256 prescale
            ldi mpr, 0b000_00_100
            sts TCCR1B, mpr

    ; Load text data from program mem to data mem for easy access
    ldi ZH, high(STRING1)
    ldi ZL, low(STRING1)
    lsl ZH      ; shift for program mem access
    lsl ZL
    adc ZH, zero ; shift carry from lower byte to upper byte
    ldi YH, high(welcome)
    ldi YL, low(welcome)
        ; Z has the loading address, Y the offloading address
        ; Need to load 16*number of phrases letters
        ;   16*11 = 176
    ldi ilcnt, 176
LOADLOOP:
        lpm mpr, Z+ ; load letter into mpr
        st Y+, mpr  ; store letter into data meme
        dec ilcnt   ; count 1 more done
        cp ilcnt, zero  ; are we done yet
```

```
        brne LOADLOOP




;***********************************************************
;*  Main Program
;***********************************************************
MAIN:
    ldi ilcnt, 0
    ldi olcnt, 1
    rcall WRITESCREEN
MAIN2:
    sbic PIND, 7 ;wait for 7 button
    rjmp MAIN2
    clr mpr
    clr olcnt

    ldi mpr, $FF
    rcall USART_TX ; send confirmation
    ldi ilcnt, 2
    ldi olcnt, 3
    rcall WRITESCREEN
    rcall USART_RX ; Wait until receive, placed in mpr
    cpi mpr, $FF
    brne MAIN
    rcall GAMESTART


    rjmp    MAIN

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


USART_TX: ; transmits mpr
    push mpr
    lds mpr, UCSR1A
    sbrs mpr,UDRE1
    rjmp USART_TX
    pop mpr
    sts UDR1, mpr
    ret

USART_RX:
```

```
	lds mpr, UCSR1A
	sbrs mpr, RXC1 ; received = skip
	rjmp USART_RX
	;get data from usart into mpr
	lds mpr, UDR1
	ret




GAMESTART:
	ldi olcnt, $FF  ; start screen
	ldi ilcnt, 4
	rcall WRITESCREEN
	;start clock for timer
	rcall STARTTIMER ; start 1.5sec timer
	clr userChoice
	inc userChoice
	inc userChoice
	ldi mpr, 0b11110000
	mov tmrcnt, mpr
	out PORTB, mpr
	clr oldbut  ; button has never had value checked!
GAMELOOP:
	;check if timer is over
	sbis TIFR1, TOV1    ; if timer overflowed
	rjmp NOTIMER
		lsl tmrcnt
		mov mpr, tmrcnt
		out PORTB, mpr
		cpi mpr, 0
		breq GAMESTART2 ; if all 4 done next
		rcall STARTTIMER ; start a new timer
	NOTIMER:
	mov mpr, oldbut
	cpi mpr, 0 ; if we weren't pressing the button already
	brne ALREADYPRESSED
		sbic PIND, 4 ; if button pressed
		rjmp ALREADYPRESSED
			ldi mpr, 1
			mov oldbut, mpr ; mark down for next loop that its pressed
			inc userChoice ; cycle to next choice
			cpi userChoice, 3
			brne BUTSKIP ; if we rolled over
				clr userChoice ; reset to rock
			BUTSKIP:
```

```
            ; Now we need to write the screen
            ldi ilcnt, 4
            ldi olcnt, 5
            add olcnt, userChoice
            rcall WRITESCREEN
    ALREADYPRESSED: ; button not pressed or was already pressed landing spot
    rcall SMALLWAIT
    sbic PIND, 4 ; if button 4 not pressed
        clr oldbut
    rjmp GAMELOOP

GAMESTART2:
    mov mpr, userChoice
    rcall USART_TX
    rcall USART_RX
    push mpr
    ldi olcnt, 5
    add olcnt, userChoice
    ldi ilcnt, 5
    add ilcnt, mpr
    rcall WRITESCREEN

    rcall STARTTIMER ; start 1.5sec timer
    ldi mpr, 0b11110000
    mov tmrcnt, mpr
    out PORTB, mpr
GAMELOOP2:
    ;check if timer is over
    sbis TIFR1, TOV1    ; if timer overflowed
    rjmp NOTIMER2
        lsl tmrcnt
        mov mpr, tmrcnt
        out PORTB, mpr
        cpi mpr, 0
        breq GAMEEND    ; if all 4 done next
        rcall STARTTIMER ; start a new timer
    NOTIMER2:
    rjmp GAMELOOP2
GAMEEND:
    pop mpr ;load mpr with p2 val
    cp userChoice, mpr
    breq uDraw

    lsl mpr ; effective mul 2
    add userChoice, mpr
    cpi userChoice, 1
```

```
    breq uWin
    cpi userChoice, 2
    breq theyWin
    cpi userChoice, 4
    breq uWin
    cpi userChoice, 5
    breq theyWin

    rjmp GAMEEND; THIS HSOULD NO THPPEN



uWin:
    ldi ilcnt, 8
    rcall WRITESCREEN
    rjmp ENDEND

theyWin:
    ldi ilcnt, 9
    rcall WRITESCREEN
    rjmp ENDEND

uDraw:
    ldi ilcnt, 10
    rcall WRITESCREEN
    rjmp ENDEND

ENDEND:
    rcall STARTTIMER ; start 1.5sec timer
    ldi mpr, 0b11110000
    mov tmrcnt, mpr
    out PORTB, mpr
ENDLOOP:
    ;check if timer is over
    sbis TIFR1, TOV1     ; if timer overflowed
    rjmp NOTIMER3
        lsl tmrcnt
        mov mpr, tmrcnt
        out PORTB, mpr
        cpi mpr, 0
        breq ENDENDEND   ; if all 4 done next
        rcall STARTTIMER ; start a new timer
    NOTIMER3:
    rjmp ENDLOOP
ENDENDEND:
    ret
```

```
;*********************************************************
;*        Write Screen
;*   Writes two words to the screen, assuming that they are
;*   stored in ilcnt and olcnt, il being the top line and
;*   ol being the bottom line
;*
;*   If the register has $FF written to it, write a blank line
;*
;*   The number stored in ilcnt will be from 0 to 10, referring
;*   to the words in the order shown at the bottom of the program
;*
;*********************************************************
WRITESCREEN:
        push XH
        push XL
        push YH
        push YL
        push ZH
        push ZL
        push mpr
        push r0
        push r1

        push ilcnt
        push olcnt

        ldi XH, $03
        ldi XL, $00

        rcall LCDClr

        pop   mpr           ; mpr has lower byte (olcnt)
        cpi mpr, $FF        ; if mpr != FF
        breq SKIPWRITE1
            ldi YH, lcd2H    ; load Y with line 2 location
            ldi YL, lcd2L
            ldi ilcnt, 16
            mul mpr, ilcnt
            mov ZH, r1
            mov ZL, r0   ; Z loaded with offset from $0300 of data
            add ZH, XH   ; offset ZH by 3
WRITELOOP1: ; moves one letter from data mem to screen data mem
```

```
            ld mpr, Z+  ; does this until 16 are moved
            st Y+, mpr
            dec ilcnt
            cp ilcnt, zero
            brne WRITELOOP1
            rcall LCDWrLn2
SKIPWRITE1:

        pop mpr         ; mpr has lower byte of top line phrase (ilcnt)
        cpi mpr, $FF    ; if mpr != FF
        breq SKIPWRITE2
            ldi YH, lcd1H   ; load Y with line 1 location
            ldi YL, lcd1L
            ldi ilcnt, 16
            mul mpr, ilcnt
            mov ZH, r1
            mov ZL, r0  ; Z loaded with offset from $0300 of data
            add ZH, XH  ; offset ZH by 3
WRITELOOP2: ; moves one letter from data mem to screen data mem
            ld mpr, Z+  ; does this until 16 are moved
            st Y+, mpr
            dec ilcnt
            cp ilcnt, zero
            brne WRITELOOP2
            rcall LCDWrLn1
SKIPWRITE2:

        pop r1
        pop r0
        pop mpr
        pop ZL
        pop ZH
        pop YL
        pop YH
        pop XL
        pop XH
        ret

;**********************************************************
;*      Start Timer
;*  Starts the timer for 1.5 seconds and clears the
;*  overflow flag
;**********************************************************
STARTTIMER:
        push mpr
        ;TIFR1 bit 0 has overflow flag
```

```
	/* Timer Value:
	High: 0b01001000
	Low:  0b11100100*/
	ldi mpr, 0b01001000 ; Must write H first
	sts TCNT1H, mpr
	ldi mpr, 0b11100100 ; If reading, L first
	sts TCNT1L, mpr ; timer reset
	ldi mpr, $01
	out TIFR1, mpr  ; clear overflow flag
	; Timer is running for 1.5 sec now,
	; just wait for bit 0 of TIFR1 to be set for the
	; timer to be done
	pop mpr
	ret
;***********************************************************
;*	Small Wait
;*  Waits for some amount of time. How much? Only god knows.
;*
;*  Useful for debouncing
;*
;***********************************************************
SMALLWAIT:
	push ilcnt
	ldi ilcnt, $FF
SMALLWAITLOOP:
	dec ilcnt
	nop	; if the switch is bouncing add more nops
	nop
	nop
	cpi ilcnt, 0
	brne SMALLWAITLOOP
	pop ilcnt
	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
;-----------------------------------------------------------
STRING1:
.DB	"Welcome!        "
STRING2:
.DB	"Please press PD7"
STRING3:
```

```
.DB       "Ready. Waiting  "
STRING4:
.DB       "for the opponent"
STRING5:
.DB       "Game start      "
STRING6:
.DB       "Rock            "
STRING7:
.DB       "Paper           "
STRING8:
.DB       "Scissor         "
STRING9:
.DB       "You won!        "
STRING10:
.DB       "You lost        "
STRING11:
.DB       "Draw            "

;***************************************************
;*  Data Memory Allocation
;***************************************************
.dseg
.org      $0300
welcome:     .byte 16
press:       .byte 16
ready:       .byte 16
for:         .byte 16
start:       .byte 16
rock:        .byte 16
paper:       .byte 16
scissor:     .byte 16
win:         .byte 16
lose:        .byte 16
draw:        .byte 16

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