Homework - Mastermind Report Draft

September 29, 2017

Sabbir Ahmed

1 Description

This project emulates a variant of the game "mastermind" on the AVR Dragon. The user will take guesses at the code using the joystick with combinations of up-down-left-right. A wrong input will sound a buzzer (piezoelectric element) and reset the user's game. Completing the code will light an LED. At this point the game can be reset by pressing the push-button. All inputs will be logged over a serial interface (UART) to a computer reporting the state of the game.

2 State Machine Implementation

The game is implemented as a state machine, where each successful inputs progresses through the states.

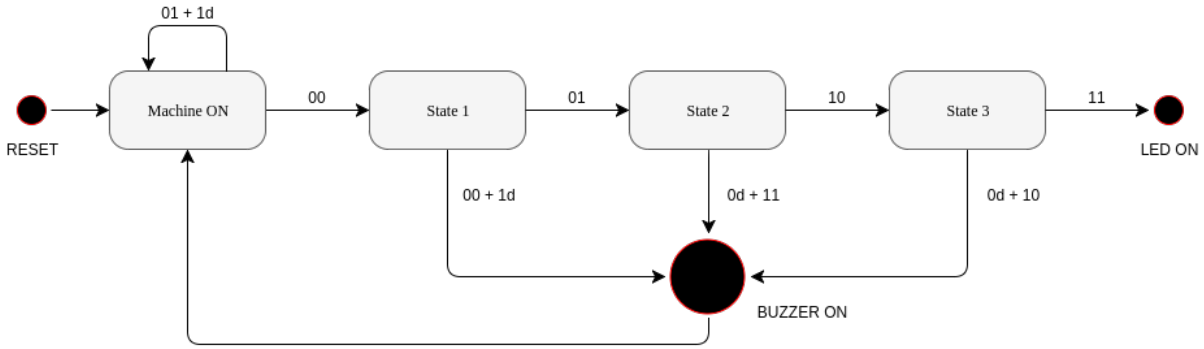

Figure 1: State Diagram Implementation of the Game with the Secret Code Mapped to 0b00011011

3 Assembly Implementation

The program is written in the AVR Assembler in scripts described in Section 4. 3 registers are used to store the user inputs and compare with the expected inputs. Another register is used to keep track of the number of shifts required for the other registers to be compared accurately.

The following snippets with visual aids will demonstrate the algorithm used in the program. The following walkthrough sets the secret code as 0b00011011, and the user successfully guesses the sequence.

3.1 Initialization

As soon as the game is loaded, the directives and labels are loaded with immediate values and the registers are initialized. The ports are initialized as well - PORTB is used for the UP and DOWN joystick inputs, and the LED and buzzer outputs, PORTD is used for the reset push button, and PORTE for the LEFT and RIGHT joystick inputs.

Table 1: Values of registers at initialization

Register	Before	After
USER	NULL	NULL
CURSOR	NULL	NULL
REALSTATE	NULL	NULL
NSHIFT	NULL	NULL
UART	NULL	NULL

3.2 State Subroutines

The program then moves to STATE0, where the secret code is loaded to a register CURSOR, the value 3 is loaded to a register NSHIFT and it waits for the first input. The current state and a delimiter are also transmitted. All the state subroutines consist of identical instructions, with the exception of the immediate values being loaded into the registers and the UART sequences.

```
; STATE0 is the initial state of the game, where the machine waits for the
; user's first input. The correct input progresses the game to the next state,
; and an incorrect input results in the buzzer being triggered.
STATE0:         RCALL TRANSMIT_0          ; transmit '0' for STATE0
                RCALL TRANSMIT_COMMA      ; transmit ','

                LDI CURSOR, SECRET        ; load and mask the secret code
                ANDI CURSOR, 0b11000000   ; into the CURSOR register
                LDI NSHIFT, 3             ; to shift USER BY 3*2 bits
                RCALL RDINPUT             ; get user's input
```

Table 2: Values of registers before RDINPUT

Register	Before	After
USER	NULL	NULL
CURSOR	NULL	0b00011011 & 0b11000000 = 0b00000000
REALSTATE	NULL	NULL
NSHIFT	NULL	0b00000010
UART	NULL	0,

3.3 Reading Inputs

RDINPUT waits in a loop for the user's input. Once an input via the joystick is triggered, the program counter moves to the subroutine that has been specified to the input. As per this demonstration, JOYSTICK UP was triggered, leading the program counter to call the JOYSTICKUP subroutine.

```
; if joystick up was pressed, the UART transmits 'U,' and the current state
; register loads the code for UP which is then shifted
JOYSTICKUP:    RCALL TRANSMIT_U          ; transmit 'U'
               RCALL TRANSMIT_COMMA      ; transmit ','

               LDI USER, UP              ; load joystick input code to USER
               RCALL LSHIFT              ; shift left USER by NSHIFT*2 bits
               RJMP DEBOUNCEUP
                   RET

; waits for user to stop pressing and then returns
DEBOUNCEUP:    SBIC PINB, 6
                   RET
               RJMP DEBOUNCEUP
```

Here, the UART is transmitted with the ASCII value for the initial of the input that was pressed, i.e. 'U', followed by the comma delimiter. The USER register is then loaded with the immediate value that was hardcoded as the input, in this case, 0b00000000 for UP.

Table 3: Values of registers right after JOYSTICKUP

Register	Before	After
USER	NULL	0b00000000
CURSOR	0b00000000	0b00000000
REALSTATE	NULL	NULL
NSHIFT	0b00000010	0b00000010
UART	0,	0,U,

The LSHIFT subroutine is then triggered, where the program decrements NSHIFT until it is ≥ 1. The USER register is left shifted twice per these iterations. All the joystick input subroutines consist of identical instructions, with the exception of the immediate values being loaded to USER.

```
; left shift the user input to match the position of the states in SECRET
LSHIFT:        LSL USER
               LSL USER                  ; left shift twice per iteration
               DEC NSHIFT                ; decrement the number of shifts
               CPI NSHIFT, 1
                   BRGE LSHIFT           ; if NSHIFT >= 1, keep looping
```

```
                RET                     ; else, breaks
```

Table 4: Values of registers after DEBOUNCEUP

Register	Before	After
USER	0b00000000	0b00000000 « 3 = 0b00000000
CURSOR	0b00000000	0b00000000
REALSTATE	NULL	NULL
NSHIFT	0b00000010	0b00000000
UART	0,U,	0,U,

3.4 Evaluating Inputs

The program counter then returns to the STATE0 subroutine, and proceeds through the rest of the lines without requiring an input.

```
STATE0:         .
                .
                .
                CLR NSHIFT
                LDI NSHIFT, 3           ; shift REALSTATE to map the codes
                MOV REALSTATE, CURSOR   ; of the joystick inputs
                RCALL RSHIFT
                RCALL EXPINPUT          ; check which input was expected

                RCALL TRANSMIT_COMMA
                RCALL CMPINPUT

                RCALL TRANSMIT_S        ; transmit 'S' for success
                RCALL TRANSMIT_NEWL     ; transmit '\n' to proceed to next
                                        ; state
```

Table 5: Values of registers right before RSHIFT

Register	Before	After
USER	0b00000000	0b00000000
CURSOR	0b00000000	0b00000000
REALSTATE	NULL	0b00000000
NSHIFT	0b00000000	0b00000010
UART	0,U,	0,U,

The NSHIFT register is reloaded with the immediate value, this time, to indicate the number of right shifts required for the REALSTATE register. The RCALL subroutine provides an identical procedure as LSHIFT, except for the direction of the bit shifting.

Table 6: Values of registers right before EXPINPUT

Register	Before	After
USER	0b00000000	0b00000000
CURSOR	0b00000000	0b00000000
REALSTATE	0b00000000	0b00000000 » 3 = 0b00000000
NSHIFT	0b00000010	0b00000000
UART	0,U,	0,U,U,1,S,\n

The EXPINPUT subroutine is then called, which compares the now shifted REALSTATE to the coded joystick immediate values. Once a match is found, the UART gets transmitted with the ASCII character of the initial of the expected input. The program counter returns to STATE0, and transmits another comma delimiter to the UART.

```
; compares the current state to find the expected output
EXPINPUT:     CPI REALSTATE, UP          ; if current state is 'UP'
                  RCALL TRANSMIT_U       ; transmit 'U'

              CPI REALSTATE, DOWN        ; if current state is 'DOWN'
                  RCALL TRANSMIT_D       ; transmit 'D'

              CPI REALSTATE, LEFT        ; if current state is 'LEFT'
                  RCALL TRANSMIT_L       ; transmit 'L'

              CPI REALSTATE, RIGHT       ; if current state is 'RIGHT'
                  RCALL TRANSMIT_R       ; transmit 'R'

                  RET
```

Finally, the CMPINPUT subroutine is called to compare the USER register with CURSOR. An equal comparison returns the program to the STATE0 subroutine, and proceeds to transmit a successful message to the UART followed by the next state. An unequal comparison branches to BUZZERON and resets the game to STATE0.

```
; compare user's input to the current state and returns if true, else, branches
; to BUZZERON to reset the game
CMPINPUT:     CP CURSOR, USER
                  RET                    ; if equal, return from subroutine
              BREQ BUZZERON              ; else, trigger buzzer
```

The program then proceeds through identical procedure for the remaining states, until the LEDON subroutine is called after the input of a successful sequence.

Table 7: Example of values of registers after STATE1

Register	Before	After
USER	0b01000000	0b01000000
CURSOR	0b01000000	0b01000000
REALSTATE	0b01000000	0b01000000
NSHIFT	0b00000001	0b00000000
UART	1,D,	1,D,D,2,S,\n

4 Code

The assembly code used for the implementation has been attached and provided at the end of the report.

4.1 main.asm

Contains the state machine implementation of the game and instructions to handle all the inputs and outputs.

4.2 uart.asm

Contains the specifications for the UART interface.

5 Testing and Troubleshooting

Because of hardware issues with the AVR Butterfly and its temporary supply issue, the program could not be tested until later in the design timeline. Testing was performed using supplies from peers, with limited availability. Early tests showed failures in the design where the inputs were incorrectly mapped to their ports. After correcting the I/O issues, complex subroutines such as loop shifting were added for elegant solutions and successful tests were performed. The LED was connected to PIN 1 of PORTB and the push button to PIN 7 of PORTD. Additional subroutines were added in the UART instructions later without the available hardware for testing.

Listing 1: main.asm

```
/* main.asm
State machine implementation of Mastermind
*/

; include the Butterfly definitions for the M169P
.INCLUDE "m169pdef.inc"

; initialize the stack and also define the port functionality
.DEF PORTDEF   = R16

; Counters 1 and 2 are used to waste time so that the buzzer sound is
   hearable
.DEF CTR       = R17
.DEF CTR2      = R18

; registers to handle the comparisons
.DEF USER      = R19
.DEF CURSOR    = R20
.DEF REALSTATE = R21
.DEF NSHIFT    = R22

; secret code to win the game (UP, DOWN, LEFT, RIGHT)
.EQU SECRET    = 0b00011011

; mapping of joystick inputs to codes embedded in SECRET
.EQU UP        = 0b00000000
.EQU DOWN      = 0b00000001
.EQU LEFT      = 0b00000010
.EQU RIGHT     = 0b00000011


.ORG $0000

; include instructions from 'uart.asm' to implement transmission with the
   UART
.INCLUDE "uart.asm"

               RJMP START

; initialize the stack.
START:         LDI PORTDEF, HIGH(RAMEND) ; upper byte
               OUT SPH, PORTDEF          ; to stack pointer

               LDI PORTDEF, LOW(RAMEND)  ; lower byte
               OUT SPL, PORTDEF          ; to stack pointer
```

```
; initialize the ports for I/O
SETUPPORTS:    LDI PORTDEF, 0b00100010   ; Pin 1 is the LED (output)
               OUT DDRB, PORTDEF         ; Pin 5 is the buzzer (output)
                                         ; Pins 6 and 7 are the UP and DOWN
                                         ; inputs of the joystick (input)

               LDI PORTDEF, 0b10000000   ; Pin 7 is the push button (input)
               OUT PORTD, PORTDEF

               LDI PORTDEF, 0b00001100   ; Pins 2 and 3 are the LEFT and
               OUT PORTE, PORTDEF        ; RIGHT inputs of the joystick
                                         ; (input)

; descriptions of states
;
   ------------------------------------------------------------------------
; STATE0 is the initial state of the game, where the machine waits for the
; user's first input. The correct input progresses the game to the next
   state,
; and an incorrect input results in the buzzer being triggered.
STATE0:        RCALL TRANSMIT_0          ; transmit '0' for STATE0
               RCALL TRANSMIT_COMMA      ; transmit ','

               LDI CURSOR, SECRET        ; load and mask the secret code
               ANDI CURSOR, 0b11000000   ; into the CURSOR register
               LDI NSHIFT, 3             ; to shift USER BY 3*2 bits
               RCALL RDINPUT             ; get user's input

               CLR NSHIFT
               LDI NSHIFT, 3             ; shift REALSTATE to map the codes
               MOV REALSTATE, CURSOR     ; of the joystick inputs
               RCALL RSHIFT
               RCALL EXPINPUT            ; check which input was expected

               RCALL TRANSMIT_COMMA
               RCALL CMPINPUT

               RCALL TRANSMIT_1          ; transmit '1' for STATE1
               RCALL TRANSMIT_COMMA      ; transmit ','
               RCALL TRANSMIT_S          ; transmit 'S' for success
               RCALL TRANSMIT_NEWL       ; transmit '\n' to proceed to next
                                         ; state


; STATE1 is the second state of the game, where the machine waits for the
; user's second input. The correct input progresses the game to the next
   state,
; and an incorrect input results in the buzzer being triggered.
```

```
STATE1:         CLR USER
                LDI CURSOR, SECRET        ; load and mask the secret code
                ANDI CURSOR, 0b00110000   ; into the CURSOR register
                LDI NSHIFT, 2             ; to shift USER BY 2*2 bits
                RCALL RDINPUT             ; get user's input

                CLR NSHIFT
                LDI NSHIFT, 2             ; shift REALSTATE to map the codes
                MOV REALSTATE, CURSOR     ; of the joystick inputs
                RCALL RSHIFT
                RCALL EXPINPUT            ; check which input was expected

                RCALL TRANSMIT_COMMA
                RCALL CMPINPUT

                RCALL TRANSMIT_2          ; transmit '2' for STATE2
                RCALL TRANSMIT_COMMA      ; transmit ','
                RCALL TRANSMIT_S          ; transmit 'S' for success
                RCALL TRANSMIT_NEWL       ; transmit '\n' to proceed to next
                                          ; state

; STATE2 is the third state of the game, where the machine waits for the
; user's third input. The correct input progresses the game to the next
    state,
; and an incorrect input results in the buzzer being triggered.
STATE2:         CLR USER
                LDI CURSOR, SECRET        ; load and mask the secret code
                ANDI CURSOR, 0b00001100   ; into the CURSOR register
                LDI NSHIFT, 1             ; to shift USER BY 1*2 bits
                RCALL RDINPUT             ; get user's input

                CLR NSHIFT
                LDI NSHIFT, 1             ; shift REALSTATE to map the codes
                MOV REALSTATE, CURSOR     ; of the joystick inputs
                RCALL RSHIFT
                RCALL EXPINPUT            ; check which input was expected

                RCALL TRANSMIT_COMMA
                RCALL CMPINPUT

                RCALL TRANSMIT_3          ; transmit '3' for STATE3
                RCALL TRANSMIT_COMMA      ; transmit ','
                RCALL TRANSMIT_S          ; transmit 'S' for success
                RCALL TRANSMIT_NEWL       ; transmit '\n' to proceed to next
                                          ; state

; STATE3 is the fourth state of the game, where the machine waits for the
; user's fourth input. The correct input progresses the game to the next
```

```
; state, and an incorrect input results in the buzzer being triggered
STATE3:         CLR USER
                LDI CURSOR, SECRET        ; load and mask the secret code
                ANDI CURSOR, 0b00000011   ; into the CURSOR register
                LDI NSHIFT, 0             ; to shift USER BY 0*2 bits
                RCALL RDINPUT             ; get user's input

                CLR NSHIFT
                LDI NSHIFT, 0             ; shift REALSTATE to map the codes
                MOV REALSTATE, CURSOR     ; of the joystick inputs
                RCALL RSHIFT
                RCALL EXPINPUT            ; check which input was expected

                RCALL TRANSMIT_COMMA
                RCALL CMPINPUT

                RCALL TRANSMIT_S          ; transmit 'S' for success
                RCALL TRANSMIT_NEWL       ; transmit '\n' to proceed to next
                                          ; state

                RCALL LEDON


; instructions to handle the I/O in the program
;
    ------------------------------------------------------------------------
; sits in a loop and waits for the user to input any of the 5 possible
    inputs
; joystick up, joystick down, joystick left, joystick right and push button
; once an input is established, their respective subroutine is called
RDINPUT:        SBIS PINB, 6              ; joystick up
                    RCALL JOYSTICKUP

                SBIS PINB, 7              ; joystick down
                    RCALL JOYSTICKDN

                SBIS PINE, 2              ; joystick left
                    RCALL JOYSTICKLT

                SBIS PINE, 3              ; joystick right
                    RCALL JOYSTICKRT

                SBIS PIND, 7              ; push button
                    RCALL RESETPB

                RJMP RDINPUT              ; if nothing was pressed, loop
```

```
; joystick inputs
;
    ------------------------------------------------------------------------
; if joystick up was pressed, the UART transmits 'U,' and the current state
; register loads the code for UP which is then shifted
JOYSTICKUP:    RCALL TRANSMIT_U          ; transmit 'U'
               RCALL TRANSMIT_COMMA      ; transmit ','

               LDI USER, UP              ; load joystick input code to USER
               RCALL LSHIFT              ; shift left USER by NSHIFT*2 bits
               RJMP DEBOUNCEUP
                   RET

; waits for user to stop pressing and then returns
DEBOUNCEUP:    SBIC PINB, 6
                   RET
               RJMP DEBOUNCEUP


; if joystick down was pressed, the UART transmits 'D,' and the current
    state
; register loads the code for DOWN which is then shifted
JOYSTICKDN:    RCALL TRANSMIT_D          ; transmit 'D'
               RCALL TRANSMIT_COMMA      ; transmit ','

               LDI USER, DOWN            ; load joystick input code to USER
               RCALL LSHIFT              ; shift left USER by NSHIFT*2 bits
               RJMP DEBOUNCEDN
                   RET

; waits for user to stop pressing and then returns
DEBOUNCEDN:    SBIC PINB, 7
                   RET
               RJMP DEBOUNCEDN


; if joystick left was pressed, the UART transmits 'L,' and the current
    state
; register loads the code for LEFT which is then shifted
JOYSTICKLT:    RCALL TRANSMIT_L          ; transmit 'L'
               RCALL TRANSMIT_COMMA      ; transmit ','

               LDI USER, LEFT            ; load joystick input code to USER
               RCALL LSHIFT              ; shift left USER by NSHIFT*2 bits
               RJMP DEBOUNCELT
                   RET

; waits for user to stop pressing and then returns
```

```
DEBOUNCELT:    SBIC PINE, 2
                   RET
               RJMP DEBOUNCELT


; if joystick right was pressed, the UART transmits 'R,' and the current
    state
; register loads the code for RIGHT which is then shifted
JOYSTICKRT:    RCALL TRANSMIT_R          ; transmit 'R'
               RCALL TRANSMIT_COMMA      ; transmit ','

               LDI USER, RIGHT           ; load joystick input code to USER
               RCALL LSHIFT              ; shift left USER by NSHIFT*2 bits
               RJMP DEBOUNCERT
                   RET

; waits for user to stop pressing and then returns
DEBOUNCERT:    SBIC PINE, 3
                   RET
               RJMP DEBOUNCERT


; if the reset push button was pressed, the program resets to STAT0
RESETPB:       RJMP LEDOFF               ; turn off LED
               RJMP DEBOUNCEPB           ; debounce push button
               RJMP STATE0               ; reset
                   RET

; waits for user to stop pressing and then returns
DEBOUNCEPB:    SBIC PIND, 7
                   RET
               RJMP DEBOUNCEPB


; instructions to compare user inputs to the secret code
;
    ------------------------------------------------------------------------
; left shift the user input to match the position of the states in SECRET
LSHIFT:        LSL USER
               LSL USER                  ; left shift twice per iteration
               DEC NSHIFT                ; decrement the number of shifts
               CPI NSHIFT, 1
                   BRGE LSHIFT           ; if NSHIFT >= 1, keep looping
                   RET                   ; else, break

; right shift the current state to match the position of the states in
    SECRET
RSHIFT:        LSR CURSOR
```

```
                LSR CURSOR              ; right shift twice per iteration
                DEC NSHIFT              ; decrement the number of shifts
                CPI NSHIFT, 1
                    BRGE LSHIFT         ; if NSHIFT >= 1, keep looping
                    RET                 ; else, break

; compares the current state to find the expected output
EXPINPUT:       CPI REALSTATE, UP       ; if current state is 'UP'
                    RCALL TRANSMIT_U    ; transmit 'U'

                CPI REALSTATE, DOWN     ; if current state is 'DOWN'
                    RCALL TRANSMIT_D    ; transmit 'D'

                CPI REALSTATE, LEFT     ; if current state is 'LEFT'
                    RCALL TRANSMIT_L    ; transmit 'L'

                CPI REALSTATE, RIGHT    ; if current state is 'RIGHT'
                    RCALL TRANSMIT_R    ; transmit 'R'

                    RET

; compare user's input to the current state and returns if true, else,
    branches
; to BUZZERON to reset the game
CMPINPUT:       CP CURSOR, USER
                    RET                 ; if equal, return from subroutine
                BREQ BUZZERON           ; else, trigger buzzer


; instructions for the LED
;
    ------------------------------------------------------------------------
; subroutine to turn on the LED
LEDON:          LDI PORTDEF, 0b00000010 ; turn on LED
                OUT PORTB, PORTDEF
                    RET

; subroutine to turn off the LED
LEDOFF:         LDI PORTDEF, 0b00000000 ; turn off LED
                OUT PORTB, PORTDEF
                    RET


; instructions for the buzzer
;
    ------------------------------------------------------------------------
; BUZZERON sets the buzzer high (at Port B, Pin 5) and then sits in a loop
    so
```

```
; that the buzzer is low enough frequency to be hearable to the human ear.
BUZZERON:       LDI PORTDEF, 0b00100000
                OUT PORTB, PORTDEF

                RCALL WASTETIME          ; WASTETIME is a counter that
                                         ; counts to 255 and then returns

                LDI PORTDEF, 0b11011111  ; once WASTETIME is finished, the
                OUT PORTB, PORTDEF       ; buzzer is turned off

                RCALL WASTETIME          ; WASTETIME is called again to
                                         ; make the period of the soundwave
                                         ; even lower

                DEC CTR2                 ; this proc is ran 255 times to
                    BRNE BUZZERON        ; make the buzzer hearable

                RCALL TRANSMIT_F         ; transmit 'F' for failure
                RCALL TRANSMIT_NEWL      ; transmit '\n'

                JMP STATE0               ; reset

; Used to make the buzzer sound hearable. Used to lower frequency enough so
; that the sound from the buzzer is hearable
WASTETIME:      CLR CTR

CONTWASTETIME: NOP
                DEC CTR
                BRNE CONTWASTETIME
                    RET
```

Listing 2: uart.asm

```
/* uart.asm
specifications for the UART interface
*/

; Instructions for the UART using the AVR Butteryfly @ default 8MHz with
; 4800 buad with 2 stop bits and no parity
.DEF TEMP      = R23
.DEF TEMP0     = R24

USARTINIT:     LDI TEMP, 00              ; Load UBRRH with 0 and UBRRL with
               STS UBRRH, TEMP           ; 103 - in other words
               LDI TEMP, 103             ; FOSC/16/BAUD-1 to set a baud rate
               STS UBRRL, TEMP           ; of about 4800 at 8MHz

               ; Clear all error flags
               LDI TEMP, 00
               STS UCSRA, TEMP

               ; Enable Transmission and Reception
               LDI TEMP, (1 << RXEN0) | (1 << TXEN0)
               STS UCSRB, TEMP

               ; Set frame format: 8data, 2stop bit
               LDI TEMP, (1 << USBS0) | (3 << UCSZ00)
               STS UCSR0C, TEMP

; data for delimiters
TRANSMIT_COMMA: LDS TEMP0, UCSR0A        ; Wait for empty transmit buffer
               SBRS TEMP0, UDRE
               RJMP TRANSMIT_COMMA

               ; send the data
               LDI TEMP, 0x2C            ; transmits ','
               STS UDR0, TEMP

TRANSMIT_NEWL: LDS TEMP0, UCSR0A         ; Wait for empty transmit buffer
               SBRS TEMP0, UDRE
               RJMP TRANSMIT_NEWL

               ; send the data
               LDI TEMP, 0x0A            ; transmits '\n'
               STS UDR0, TEMP


; data for input status
TRANSMIT_S:    LDS TEMP0, UCSR0A         ; Wait for empty transmit buffer
```

```
                SBRS TEMP0, UDRE
                RJMP TRANSMIT_S

                ; send the data
                LDI TEMP, 0x53          ; transmits 'S'
                STS UDR0, TEMP

TRANSMIT_F:     LDS TEMP0, UCSR0A       ; Wait for empty transmit buffer
                SBRS TEMP0, UDRE
                RJMP TRANSMIT_F

                ; send the data
                LDI TEMP, 0x46          ; transmits 'F'
                STS UDR0, TEMP


; data for joystick input
TRANSMIT_U:     LDS TEMP0, UCSR0A       ; Wait for empty transmit buffer
                SBRS TEMP0, UDRE
                RJMP TRANSMIT_U

                ; send the data
                LDI TEMP, 0x55          ; transmits 'U'
                STS UDR0, TEMP

TRANSMIT_D:     LDS TEMP0, UCSR0A       ; Wait for empty transmit buffer
                SBRS TEMP0, UDRE
                RJMP TRANSMIT_D

                ; send the data
                LDI TEMP, 0x44          ; transmits 'D'
                STS UDR0, TEMP

TRANSMIT_L:     LDS TEMP0, UCSR0A       ; Wait for empty transmit buffer
                SBRS TEMP0, UDRE
                RJMP TRANSMIT_L

                ; send the data
                LDI TEMP, 0x4C          ; transmits 'L'
                STS UDR0, TEMP

TRANSMIT_R:     LDS TEMP0, UCSR0A       ; Wait for empty transmit buffer
                SBRS TEMP0, UDRE
                RJMP TRANSMIT_R

                ; send the data
                LDI TEMP, 0x52          ; transmits 'R'
                STS UDR0, TEMP
```

```
; data for states
TRANSMIT_0:    LDS TEMP0, UCSR0A        ; Wait for empty transmit buffer
               SBRS TEMP0, UDRE
               RJMP TRANSMIT_0

               ; send the data
               LDI TEMP, 0x30           ; transmits '0'
               STS UDR0, TEMP

TRANSMIT_1:    LDS TEMP0, UCSR0A        ; Wait for empty transmit buffer
               SBRS TEMP0, UDRE
               RJMP TRANSMIT_1

               ; send the data
               LDI TEMP, 0x31           ; transmits '1'
               STS UDR0, TEMP

TRANSMIT_2:    LDS TEMP0, UCSR0A        ; Wait for empty transmit buffer
               SBRS TEMP0, UDRE
               RJMP TRANSMIT_2

               ; send the data
               LDI TEMP, 0x32           ; transmits '2'
               STS UDR0, TEMP

TRANSMIT_3:    LDS TEMP0, UCSR0A        ; Wait for empty transmit buffer
               SBRS TEMP0, UDRE
               RJMP TRANSMIT_3

               ; send the data
               LDI TEMP, 0x33          ; transmits '3'
               STS UDR0, TEMP

TRANSMIT_4:    LDS TEMP0, UCSR0A        ; Wait for empty transmit buffer
               SBRS TEMP0, UDRE
               RJMP TRANSMIT_4

               ; send the data
               LDI TEMP, 0x34           ; transmits '4'
               STS UDR0, TEMP

; now send the data using a function call
DONE:          RJMP DONE

; assumes data is in register TEMP
TRANSMIT:      LDS TEMP0, UCSR0A        ; wait for empty transmit buffer
```

```
SBRS TEMP0, UDRE
RJMP USART_TRANSMIT

STS UDR0, TEMP            ; send the data
RET
```