AVR GAME

INSTRUCTIONS, SUBROUTINE EXPLAINATION AND TESTING

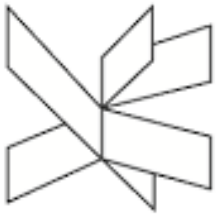
Supervisor:

Christian Flinker Sandbeck (CHFS)

Group:

Tor Jacobsen

Faizan Yousaf



VIA University College

Table of Contents

Instructions of the AVR game

At the beginning of a new game the player is shown the game start sequence. That means that all the LEDs will blink three times. Player should wait at this point.

The level LEDs will be shown, representing the current level and they will keep blinking until the player presses any button which will start the random sequence.

The first sequence has two output numbers. The user should wait until sequence is stopped.

The user should enter the sequence in exact order, any wrong press will trigger game over.

If the entered sequence was right, then level won sequence will be shown. The game level will increase by one, increasing also the sequence by one. At this point user should wait until the next random number sequence is shown.

If all levels are completed, then the game won sequence will be shown and the game will start again.

CONFIGURATION

The program requires configuration before starting the game that is stack configuration, DDR configuration for outputting values on a PORT and in case of Random Number Generation also saving the X value as a seed in memory.

Code for configuration:

```
;-----------------------------------------------------------
;--------------------- CONFIGURATION ----------------------
;-----------------------------------------------------------
; stack setup
ldi r16, high(RAMEND)       ; 0x21
out sph, r16
ldi r16, low(RAMEND) ; 0xff
out spl, r16

LDI R19, 0x7
STS 0x300, R19              ; X = 7 |  STORING X VALUE IN SRAM FOR RANDOM NUMBER
GENERATOR
;NOTE: R0 IS USED TO REPRESENT THE VALUE OF X FOR RANDOM NUMBER GENERATOR
ldi r17, 0xff        ; 1111_1111 this is used for setting up the output port
out ddra, r17
```

SUBROUTINES

Game welcome sequence

Since this sequence should turn on all LEDs, it is also used to check output components (8 LEDs) if they are in good condition.

The code provided below will make all LED'S blink three times. This is used in the AVR game before a new game starts. If required, the value that is being loaded into register 21 at the beginning can be changed according to how many times the LED's needs to blink. In the code below the value is three, so it will blink three times.

The DELAY and RESET_LIGHTS subroutines can be found at the bottom of this document.

Code:

```
;-----------------------------------------------------------
;---------------------- GAME WELCOME ----------------------
;-----------------------------------------------------------
WELCOME:
	PUSH R21
	PUSH R17

	LDI R21, 0x3  ; R21 = 3 | TO RUN SEQUENCE 3 TIMES
	WELCOME_LOOP:
		LDI R17, 0b0000_0000 ; 0000_0000 TO TURN ALL LIGHTS ON
		OUT PORTA, R17               ; TURN ALL LEDS ON
		CALL DELAY
		CALL RESET_LIGHTS     ; TURN ALL LEDS OFF
		CALL DELAY                   ; TIME DELAY
		DEC R21
		BRNE WELCOME_LOOP

	POP R17
	POP R21
	RET
```

Round won sequence

The code provided below will make all LED'S blink in a sequence from left to right. In the AVR game, this sequence is used after every round if the user puts in the correct inputs. To make the sequence last longer or shorter the variable that is being loaded into register 21 needs to be changed from it's current values which is 8.

The RESET_LIGHTS subroutine can be found at the bottom of this document.

Code:

```
;-----------------------------------------------------------
;---------------------- ROUND WON ------------------------
;-----------------------------------------------------------

; ALL LIGHTS SHOULD BLINK IN SEQUENCE FROM LEFT TO RIGHT
ROUND_WON:
	PUSH R16
	PUSH R21
	PUSH R17

	LDI R21, 0x3		; R21 = 3 | TO RUN SEQUENCE 3 TIMES
	ROUND_WON_LIGHTS_LOOP1:
		LDI R16, 8				; TO RUN LOOP 2 EIGHT TIMES TURNING
ON ONE LED IN EACH LOOP (LEFT TO RIGHT)
		LDI R17, 0b0000_0001  ; SAVE REGISTER VALUES TO TURN FIRST RIGHT LIGHT ON
		ROUND_WON_LIGHTS_LOOP2:
			COM R17
			OUT PORTA, R17		; TURN LIGHT ON
			COM R17
			LSL R17				; SHIFT R17 BITMASK TO LEFT
			RCALL SHORT_DELAY	; SHORT DELAY
			DEC R16				; DECREASE R16 UNTIL 0
			BRNE ROUND_WON_LIGHTS_LOOP2	; IF R16 NOT 0 THEN BRANCH TO
TURN NEXT LED ON
		DEC R21					; DECREASE R21
UNTIL 0
		BRNE ROUND_WON_LIGHTS_LOOP1	; IF R16 NOT 0 THEN BRANCH
	CALL RESET_LIGHTS		; RESET LIGHTS

	POP R17
	POP R21
	POP R16
	RET
```

Game won sequence

The code below will make half of LEDs blink in sequence from far left to right, half from far right to left, meeting together in middle, just like a clap with two hands. This sequence is being showed at the end of a game, if the game has been won. Values of register R21 can be changed to make the sequence last longer or shorter.

The DELAY, SHORT_DELAY and RESET_LIGHTS subroutines can be found at the bottom of this document.

Code:

```
;-----------------------------------------------------
;---------------------- GAME WON ----------------------
;-----------------------------------------------------
; HALF OF LIGHTS SHOULD BLINK IN SEQUENCE FROM FAR LEFT TO RIGHT,
; HALF FROM FAR RIGHT TO LEFT, MEETING TOGETHER IN MIDDLE(LIKE A CLAP)
GAME_WON:
	PUSH R17
	PUSH R21

	LDI R21, 0x32		; R21 = 50 | TO RUN SEQUENCE 50 TIMES
	END_GAME_LOOP:
		LDI R17, 0b0111_1110 ; SAVE REGISTERS VALUES
		OUT PORTA, R17		; TURN ALL LEDS ON
		CALL SHORT_DELAY	; SHORT DELAY

		LDI R17, 0b0011_1100 ; SAVE REGISTERS VALUES
		OUT PORTA, R17		; TURN ALL LEDS ON
		CALL SHORT_DELAY	; SHORT DELAY

		LDI R17, 0b0001_1000 ; SAVE REGISTERS VALUES
		OUT PORTA, R17		; TURN ALL LEDS ON
		CALL SHORT_DELAY	; SHORT DELAY

		LDI R17, 0b0000_0000 ; SAVE REGISTERS VALUES
		OUT PORTA, R17		; TURN ALL LEDS ON
		CALL SHORT_DELAY	; SHORT DELAY

		CALL RESET_LIGHTS	; TURN ALL LEDS OFF
		DEC R21			; DESCREASE R21 UNTIL 0
		BRNE END_GAME_LOOP	; BRANCH IF NOT 0
	CALL RESET_LIGHTS		; RESET LIGHTS
	CALL DELAY			; DELAY
	CALL DELAY			; DELAY

	POP R21
	POP R17

	RET
```

Game lost sequence

This subroutine makes the LED'S turn on for about three seconds. This sequence is used if a user loses a game by putting in the wrong input. Values of register R21 can be changed to make the sequence last longer or shorter.

The DELAY and RESET_LIGHTS subroutines can be found at the bottom of this document.

Code:

```
;-----------------------------------------------------------
;---------------------- GAME LOST ------------------------
;-----------------------------------------------------------
; ALL LIGHTS SHOULD TURN ON TOGETHER FOR A LONG TIME

GAME_LOST:
	PUSH R17
	PUSH R21

	LDI R17, 0b0000_0000 ; SAVE REGISTERS VALUES TO TURN ALL LIGHTS ON
	OUT PORTA, R17               ; TURN ALL LIGHTS ON

	LDI R21, 0x14         ; R21 = 20 | TO RUN DELAY SEQUENCE 20 TIMES
	GAME_LOST_DELAY:      ; DELAY TO KEEP LIGHTS LIT FOR LONG TIME
		CALL DELAY
		DEC R21
		BRNE GAME_LOST_DELAY

	CALL RESET_LIGHTS     ; RESET LIGHTS
	CALL DELAY                    ; DELAY
	CALL DELAY

	POP R21
	POP R17
	RE
```

Delay subroutine

Below are two subroutines that were used to waste clock, just for better user experience. *DELAY* wastes more clock cycles than *SHORT_DELAY*, thus it provides longer period of time delayed.

Code:

```
; DELAY CALCUCATOIN
; Clock frequency = 125 kHz = 0,125 MHz
; 1 Machine cycle = 8 ns
; DELAY = ((74 * 1018 * 1018) + 12 + 4 + 1) * 8ns
;       = (76687976 + 17) * 8ns
;       = 613503944 ns = 0.613 seconds.
DELAY:                                          ; INSTRUCTION CYCLES
       PUSH R18                                 ; 2
       PUSH R19                                 ; 2
       PUSH R20                                 ; 2

       LDI r18, 255                             ; 1
       LOOP_1:
              LDI r19, 255                      ; 1
              INNERLOOP_1:
                     LDI r20, 25                ; 1
                            MOSTINNERLOOP_1:
                            DEC r20             ; 1
                            BRNE MOSTINNERLOOP_1 ; 2/1
                     DEC r19                    ; 1
                     BRNE INNERLOOP_1           ; 2/1
              DEC r18                           ; 1
              BRNE LOOP_1                       ; 2/1

       POP R20                                  ; 2
       POP R19                                  ; 2
       POP R18                                  ; 2
       RET                                      ; 4

;-----------------------------------------------------------
;-------------------- SHORT DELAY ------------------------
;-----------------------------------------------------------
; DELAY CALCUCATOIN
; Clock frequency = 125 kHz = 0,125 MHz
; 1 Machine cycle = 8 ns
; DELAY = ((44 * 384 * 384) + 12 + 4 + 1) * 8 ns
;       = (6488064 + 17) * 8
;       = 51904648 ns = 0,0519 seconds.
SHORT_DELAY:                                    ; INSTRUCTION CYCLES
       PUSH R18                                 ; 2
       PUSH R19                                 ; 2
       PUSH R20                                 ; 2

       LDI R18, 128                             ; 1
       SHORT_LOOP_1:
              LDI R19, 128                      ; 1
              SHORT_INNERLOOP_1:
                     LDI R20, 15                ; 1
                            SHORT_MOSTINNERLOOP_1:
                            DEC R20             ; 1
                            BRNE SHORT_MOSTINNERLOOP_1  ; 2/1
                     DEC R19                    ; 1
                     BRNE SHORT_INNERLOOP_1     ; 2/1
              DEC R18                           ; 1
              BRNE SHORT_LOOP_1                 ; 2/1

       POP R20                                  ; 2
```

```
        POP R19                                         ; 2
        POP R18                                         ; 2
        RET                                             ; 4
```

Reset lights subroutine

This subroutine resets lights by turning off all LED's

Code:

```
;-----------------------------------------------------------
; --------- RESET LIGHTS / TURN OFF ALL LIGHTS ------------
;-----------------------------------------------------------
RESET_LIGHTS:
        PUSH R17
        LDI R17, 0b1111_1111 ; SAVE REGISTERS VALUES TO TURN ALL LIGHTS OFF
        OUT PORTA, R17               ; TURN ALL LEDS OFF
        POP R17
        RET
```

Generating Random Number:

To generate a random number and show it on LED, Linear Congruential Generator (LCG) algorithm and a loop were used.

LCG algorithm was used to generate a random number and save it in register R0 register and SRAM ($300) memory for further use.

A loop was created that runs as many time as R0 register values, decreasing the value of R0 with each loop. For this loop, a register is loaded with value (xb000_0001) and with each repetition, it is summed up with itself, shifting the bit to left. When the R0 reaches 0 the generated value is shown on LED and saved in SRAM.

Initially the seed value was meant to be saved somewhere in AVR memory where it does not get lost after restarting the program, that is in EEPROM. It should have worked however it was unsuccessful and later it was decided to save the seed value in SRAM, consequently the program generate same numbers whenever restarted

TESTING

For testing most of the functionality separate test files were made:

- inputTest.asm
- sequenceTest.asm
- testRandomNumberGenerator.asm

Test sequences

For testing all the sequences mentioned above, *sequenceTest.asm* can be executed on the AVR. Test is conducted by observing all the sequences visually and if they are as described above then the test pass.

This test is also used to check output components (LEDs), by observing the first sequence (Welcome sequence). If all LEDs are turning on, then the output components are good.

Test input buttons

The code provided below is from *inputTest.asm*, which will get input from any button and show on the corresponding LED right above it. For this test user must press all the buttons in any desired order. The test is passed if each LED turns on by pressing its corresponding button.

Code:

```
; configuration of port
ldi r17, 0xff ; ; 1111 1111 this is used for setting up the port
out ddra, r17

START:
	I1:
		IN r17, pinb		; Gets input from button.
		OUT porta, r17		; Shows input on the led.
		COM r17			; complementing r17 because of stk600
		BREQ I1			; Check if a button has been pressed. If not it
keeps looping
	RJMP START			; start over again - loop forever -
```

Test Random Number Generation:

Since the output values generated are always same when the program is restarted, it can be verified that the algorithm and calculations are working fine by running the *testRandomNumberGenerator.asm* two times in simulator and checking values being stored in memory.

REFERENCES

- https://en.wikipedia.org/wiki/Linear_congruential_generator
- [**Mazidi, 2011**]: Muhammad Ali Mazidi, Sarmad Naimi and Sepehr Naimi - The AVR Microcontroller and Embedded System using Assembly and C