Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνμολογίας Πληροφορικής και Υπολογιστών

Εργαστήριο Μικροϋπολογιστών

4η Σειρά Ασκήσεων - AVR

Ον/μο: Τσάφος Αλέξανδρος

ΑΜ: 03118211, Ομάδα: 80

Ζήτημα 4.1

```
.DSEG
_tmp_: .byte 2

.CSEG
.include "m16def.inc"

;variables to make it easier

;b0 -> login_flag, b1-> gas_detected, b2-> alarm_flag, b3 -> gas_clear
;login_flag = 1 when the team goes in the room
;gas_detected = 1 when CO > 70ppm. Used in ADC int handler
;alarm_flag changes between 0-1 everytime the timer is called. Used for blinking
;gas_clear = 1 when the CO msr returns to normal
.def flags = r16
.def output = r17
.def lcd_message = r18 ;0x01 = gas_detected, 0x02 = clear

.org 0x00
rjmp start

.org 0x10
rjmp ISR_TIMER1_OVF

.org 0x1C
rjmp ADC_ISR

start:
	ldi r24, low(RAMEND) ;initialize stack pointer
	out SPL, r24
	ldi r24, high(RAMEND)
	out SPH, r24
	ser r24
	out DDRB, r24 ;output
	out DDRD, r24 ;output
	ldi r24, 0xF0
	out DDRC, r24 ;output and input
```

```
	rcall ADC_init
	ldi r24 ,(1<<TOIE1) ;EI
	out TIMSK ,r24
	ldi r24 ,(1<<CS12) | (0<<CS11) | (1<<CS10) ; CK/1024
	out TCCR1B ,r24
	;0xFCF3 = 64755
	ldi r24, 0xFC
	out TCNT1H, r24
	ldi r24, 0xF3
	out TCNT1L, r24
	clr lcd_message
	clr flags

;team 80, searching for '8' (r24 = 0x20) and '0' (r24 = 0x02) input

sei

read1:
	andi flags, 0x0C    ;keep alarm_flag and gas_clear as they are
	;rcall lcd_init_sim ;reset the display
	rcall scan_keypad_rising_edge_sim ;scan
	rcall keypad_to_ascii_sim ;match to ascii
	cpi r24, 0x00 ;did any button get pressed
	breq read1 ; if not, read again

	mov r21, r24 ; tempotarily store r24, to check for '8' later

read2:
	rcall scan_keypad_rising_edge_sim ;scan
	rcall keypad_to_ascii_sim ;match to ascii
	cpi r24, 0x00 ;did any button get pressed
	breq read2 ; if not, read again

	ldi r20, 0x08 ; 8 time counter
	cpi r21, '8' ; 1st digit must be '8'
	brne wrong
	cpi r24, '0' ; 2nd digit must be '0'
	brne wrong

access:
	;display the message
	rcall lcd_init_sim
	ldi r24,'W'
	rcall lcd_data_sim
	ldi r24,'E'
	rcall lcd_data_sim
	ldi r24,'L'
	rcall lcd_data_sim
	ldi r24,'C'
	rcall lcd_data_sim
	ldi r24,'O'
```

```
       rcall lcd_data_sim
       ldi r24,'M'
       rcall lcd_data_sim
       ldi r24,'E'
       rcall lcd_data_sim
       ldi r24,' '
       rcall lcd_data_sim
       ldi r24,'8'
       rcall lcd_data_sim
       ldi r24,'0'
       rcall lcd_data_sim
       ldi lcd_message, 0x11 ;no permission to display other message


welcome:
       ori output, 0x80 ;turn on the MSB
       ori flags, 0x01 ;login_flag
       out PORTB, output
       rcall scan_keypad_rising_edge_sim ; read and ignore
       ldi r24,low(3900) ;compensate for other delays
       ldi r25,high(3900)
       rcall wait_msec
  rcall scan_keypad_rising_edge_sim ; read and ignore
       andi output, 0x7F; turn off PB7
       andi flags, 0x0C
       out PORTB, output ;turn off the LED
       rcall lcd_init_sim ;erase welcome message
       ldi lcd_message, 0x00 ;allow other message
       rjmp read1 ;start again

wrong:
       ori output, 0x80
       out PORTB, output
       rcall scan_keypad_rising_edge_sim ; read and ignore
       ldi r24,low(450) ; 4x2x500ms delays
       ldi r25,high(450)
       rcall wait_msec ;500ms
       dec r20
       andi output, 0x7F   ; turn off PB7
       out PORTB, output
       rcall scan_keypad_rising_edge_sim ; read and ignore
       ldi r24,low(450) ; 4x2x500ms delays
       ldi r25,high(450); 450 + commands in the loop
       rcall wait_msec
       dec r20
       brne wrong
       rjmp read1 ;start again

lcd_alarm:
       ;display message
       rcall lcd_init_sim
```

```
	ldi r24,'G'
	rcall lcd_data_sim
	ldi r24,'A'
	rcall lcd_data_sim
	ldi r24,'S'
	rcall lcd_data_sim
	ldi r24,' '
	rcall lcd_data_sim
	ldi r24,'D'
	rcall lcd_data_sim
	ldi r24,'E'
	rcall lcd_data_sim
	ldi r24,'T'
	rcall lcd_data_sim
	ldi r24,'E'
	rcall lcd_data_sim
	ldi r24,'C'
	rcall lcd_data_sim
	ldi r24,'T'
	rcall lcd_data_sim
	ldi r24,'E'
	rcall lcd_data_sim
	ldi r24,'D'
	rcall lcd_data_sim
	ldi lcd_message, 0x01 ;indicate the alarm message is displayed
	ret

lcd_clear:
	rcall lcd_init_sim
	ldi r24,'C'
	rcall lcd_data_sim
	ldi r24,'L'
	rcall lcd_data_sim
	ldi r24,'E'
	rcall lcd_data_sim
	ldi r24,'A'
	rcall lcd_data_sim
	ldi r24,'R'
	rcall lcd_data_sim
	andi flags, 0x07 ; unset gas_clear
	ldi lcd_message, 0x02 ;indicate the clear message is displayed
	ret

ISR_TIMER1_OVF:
	push r24
  push r25
	in r24, ADCSRA      ;load ADCSRA and change the ADSC bit
	ori r24, (1<<ADSC) ;begin conversion
	out ADCSRA, r24
	ldi r24, 0xFC ;reset the timer
	out TCNT1H, r24
```

```
	ldi r24, 0xF3
	out TCNT1L, r24
	mov r24, flags	;complement the alarm_flag bit
	andi flags, 0x09 ;clear alarm_flag
	com r24
	andi r24, 0x04 ;isolate alarm_flag
	or flags, r24
  pop r25
  pop r24
	reti

;Thresholds decided:
;0-31, 32-63, 64-127, 128-255, 256-383, 384-511, 512-767
;easier to find the hex values, as they are powers of 2
ADC_ISR:
	push r24
	push r25
	push r26 ;used to store CO state

	in r24, ADCL
	in r25, ADCH
	andi r25, 0x03 ;keep the 2 LSB's
	cpi r25, 0x00
	breq less_than_256 ;if r25 is zero, we are below 256
	ori flags, 0x02 ;gas_detected
	cpi r25, 0x02 ;512 ?
	brlo check
	rjmp seven
	check:
		cpi r24, 0x80 ;384 ?
		brlo five
		rjmp six
	less_than_256:
		cpi r24, 0x20 ;<32 ?
		brlo one
		cpi r24, 0x40 ;<64 ?
		brlo two
		cpi r24, 0x80 ;<128 ?
		brlo three
		rjmp four		;<256
	one:
		ldi r26, 0x01
		rjmp leds_ok
	two:
		ldi r26, 0x03
		rjmp leds_ok
	three:
		ldi r26, 0x07
		rjmp leds_ok
	four:
		ldi r26, 0x0F
```

```
            rjmp flags_ok
      five:
            ldi r26, 0x1F
            rjmp flags_ok
      six:
            ldi r26, 0x3F
            rjmp flags_ok
      seven:
            ldi r26, 0x7F
            rjmp flags_ok

      leds_ok:
            cpi r24, 205 ; check if CO > 70ppm
            brlo flags_ok
            ori flags, 0x02 ;gas_detected

      flags_ok:
            andi output, 0x80 ; isolate MSB
            sbrc flags, 0 ;login_flag
            rjmp login
            sbrs flags, 1 ;gas_detected
            rjmp normal_gas
            rjmp over70

      over70:
            sbrs lcd_message, 0 ;if gas_detected is diplayed, don't call again
            rcall lcd_alarm
            ori flags, 0x08 ;set gas_clear
            sbrc flags, 2 ;alarm_flag
            or output, r26 ;add the CO msr only when alarm_flag is set
(alternating every 100ms)
            rjmp exit

      login:
            or output, r26 ;always show without blinking
            rjmp exit

      normal_gas:
            sbrc flags, 3 ;gas_clear
            rcall lcd_clear     ;display to lcd, and clear flag
            or output, r26      ;add the CO msr

      exit:
            out PORTB, output
            andi flags, 0x0D ;clear the gas_detected flag
            pop r26
            pop r25
            pop r24
            reti

; Calls Given in the PDF (Copied and Pasted)
```

```
; No need to check
scan_row_sim:
	out PORTC, r25
	push r24
	push r25
	ldi r24,low(500)
	ldi r25,high(500)
	rcall wait_usec
	pop r25
	pop r24
	nop
	nop
	in r24, PINC
	andi r24 ,0x0f
	ret


scan_keypad_sim:
	push r26
	push r27
	ldi r25 , 0x10
	rcall scan_row_sim
	swap r24
	mov r27, r24
	ldi r25 ,0x20
	rcall scan_row_sim
	add r27, r24
	ldi r25 , 0x40
	rcall scan_row_sim
	swap r24
	mov r26, r24
	ldi r25 ,0x80
	rcall scan_row_sim
	add r26, r24
	movw r24, r26
	clr r26
	out PORTC,r26
	pop r27
	pop r26
	ret

scan_keypad_rising_edge_sim:
	push r22
	push r23
	push r26
	push r27
	rcall scan_keypad_sim
	push r24
	push r25
	ldi r24 ,15
	ldi r25 ,0
```

```
        rcall wait_msec
        rcall scan_keypad_sim
        pop r23
        pop r22
        and r24 ,r22
        and r25 ,r23
        ldi r26 ,low(_tmp_)
        ldi r27 ,high(_tmp_)
        ld r23 ,X+
        ld r22 ,X
        st X ,r24
        st -X ,r25
        com r23
        com r22
        and r24 ,r22
        and r25 ,r23
        pop r27
        pop r26
        pop r23
        pop r22
        ret

keypad_to_ascii_sim:
        push r26
        push r27
        movw r26 ,r24
        ldi r24 ,'*'
        sbrc r26 ,0
        rjmp return_ascii
        ldi r24 ,'0'
        sbrc r26 ,1
        rjmp return_ascii
        ldi r24 ,'#'
        sbrc r26 ,2
        rjmp return_ascii
        ldi r24 ,'D'
        sbrc r26 ,3
        rjmp return_ascii
        ldi r24 ,'7'
        sbrc r26 ,4
        rjmp return_ascii
        ldi r24 ,'8'
        sbrc r26 ,5
        rjmp return_ascii
        ldi r24 ,'9'
        sbrc r26 ,6
        rjmp return_ascii
        ldi r24 ,'C'
        sbrc r26 ,7
        rjmp return_ascii
        ldi r24 ,'4'
```

```
        sbrc r27 ,0
        rjmp return_ascii
        ldi r24 ,'5'
        sbrc r27 ,1
        rjmp return_ascii
        ldi r24 ,'6'
        sbrc r27 ,2
        rjmp return_ascii
        ldi r24 ,'B'
        sbrc r27 ,3
        rjmp return_ascii
        ldi r24 ,'1'
        sbrc r27 ,4
        rjmp return_ascii
        ldi r24 ,'2'
        sbrc r27 ,5
        rjmp return_ascii
        ldi r24 ,'3'
        sbrc r27 ,6
        rjmp return_ascii
        ldi r24 ,'A'
        sbrc r27 ,7
        rjmp return_ascii
        clr r24
        rjmp return_ascii

return_ascii:
        pop r27
        pop r26
        ret

write_2_nibbles_sim:
        push r24
        push r25
        ldi r24 ,low(6000)
        ldi r25 ,high(6000)
        rcall wait_usec
        pop r25
        pop r24
        push r24
        in r25, PIND
        andi r25, 0x0f
        andi r24, 0xf0
        add r24, r25
        out PORTD, r24
        sbi PORTD, PD3
        cbi PORTD, PD3
        push r24
        push r25
        ldi r24 ,low(6000)
        ldi r25 ,high(6000)
```

```
        rcall wait_usec
        pop r25
        pop r24
        pop r24
        swap r24
        andi r24 ,0xf0
        add r24, r25
        out PORTD, r24
        sbi PORTD, PD3
        cbi PORTD, PD3
        ret

lcd_data_sim:
        push r24
        push r25
        sbi PORTD, PD2
        rcall write_2_nibbles_sim
        ldi r24 ,43
        ldi r25 ,0
        rcall wait_usec
        pop r25
        pop r24
        ret

lcd_command_sim:
        push r24
        push r25
        cbi PORTD, PD2
        rcall write_2_nibbles_sim
        ldi r24, 39
        ldi r25, 0
        rcall wait_usec
        pop r25
        pop r24
        ret

lcd_init_sim:
        push r24 push r25
        ldi r24, 40
        ldi r25, 0
        rcall wait_msec
        ldi r24, 0x30
        out PORTD, r24
        sbi PORTD, PD3
        cbi PORTD, PD3
        ldi r24, 39
        ldi r25, 0
        rcall wait_usec
        push r24
        push r25
        ldi r24,low(1000)
```

```
      ldi r25,high(1000)
      rcall wait_usec
      pop r25
      pop r24
      ldi r24, 0x30
      out PORTD, r24
      sbi PORTD, PD3
      cbi PORTD, PD3
      ldi r24,39
      ldi r25,0
      rcall wait_usec
      push r24
      push r25
      ldi r24 ,low(1000)
      ldi r25 ,high(1000)
      rcall wait_usec
      pop r25
      pop r24
      ldi r24,0x20
      out PORTD, r24
      sbi PORTD, PD3
      cbi PORTD, PD3
      ldi r24,39
      ldi r25,0
      rcall wait_usec
      push r24
      push r25
      ldi r24 ,low(1000)
      ldi r25 ,high(1000)
      rcall wait_usec
      pop r25
      pop r24
      ldi r24,0x28
      rcall lcd_command_sim
      ldi r24,0x0c
      rcall lcd_command_sim
      ldi r24,0x01
      rcall lcd_command_sim
      ldi r24, low(1530)
      ldi r25, high(1530)
      rcall wait_usec
      ldi r24 ,0x06
      rcall lcd_command_sim
      pop r25
      pop r24
      ret

ADC_init:
  ldi r24,(1<<REFS0) ; Vref: Vcc
  out ADMUX,r24      ;MUX4:0 = 00000 for A0.
  ;ADC is Enabled (ADEN=1)
```

```
  ;ADC Interrupts are Enabled (ADIE=1)
  ;Set Prescaler CK/128 = 62.5Khz (ADPS2:0=111)
  ldi r24,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  out ADCSRA,r24
 ret

wait_msec:
       push r24
       push r25
       ldi r24 , low(998)
       ldi r25 , high(998)
       rcall wait_usec
       pop r25
       pop r24
       sbiw r24 , 1
       brne wait_msec
       ret

wait_usec:
       sbiw r24 ,1
       nop
       nop
       nop
       nop
       brne wait_usec
       ret
```

Ζήτημα 4.2

```
#define F_CPU 8000000                    // FREQUENCY OF ATMEGA16

#include <avr/io.h>
#include <avr/interrupt.h>

char ram[2], key[2], digit[2], output = 0x00, msb_flag = 0x00;
int alarm = 0, login_flag = 0, CO_msr = 0;


//translated from assembly. 1 us delay
void wait_usec(int j){
       for(int i = 0; i < j; i++){
              asm("nop");
              asm("nop");
              asm("nop");
              asm("nop");
       }
}
```

```
//also tranlated from assembly. 1ms delay
void wait_msec(int j){
	for (int i = 0; i < j; i++){
		wait_usec(1000);
	}
}

//Scan a row of the keypad for input
//input: row of choice
//output: row's status
char scan_row(char c){
	PORTC = c;
	wait_usec(500);
	asm("nop");
	asm("nop");
	return (PINC & 0x0f);
}

//swap the 4 MSB with the 4 LSB of a variable
char swap(char word){
	return ((word & 0x0f) << 4 | (word & 0xf0) >> 4);
}

//scan the whole keypad's status.
//input: none
//output: none
//The keypad's status is stored in key[1] and key[2]
void scan_keypad(){
	char ret;

	ret = scan_row(0x10); //1st line
	key[1] = swap(ret);

	ret = scan_row(0x20); //2nd line
	key[1] += ret;

	ret = scan_row(0x40); //3rd line
	key[0] = swap(ret);

	ret = scan_row(0x80); //4th line
	key[0] += ret;
}

//scan the keypad for recently pressed buttons
//input: none
//output: none
void scan_keypad_rising_edge(){
	char ret[2];

	scan_keypad(); //scan and store
```

```
	ret[0] = key[0];
	ret[1] = key[1];

	wait_msec(15); //prevent sparkling

	scan_keypad();

	key[0] &= ret[0]; //check if the button is indeed pressed
	key[1] &= ret[1];

	ret[0] = ram[0];  //restore the last call's pressed buttons
	ret[1] = ram[1];

	ram[0] = key[0];  //store this call's pressed buttons
	ram[1] = key[1];

	key[0] &= ~ret[0]; //check if the button is newly pressed
	key[1] &= ~ret[1];
}


//match the button pressed, to it's ascii char,
//according to the manual
char keypad_to_ascii(){
	if (key[0] & 0x01) return '*';

	if (key[0] & 0x02) return '0';

	if (key[0] & 0x04) return '#';

	if (key[0] & 0x08) return 'D';

	if (key[0] & 0x10) return '7';

	if (key[0] & 0x20) return '8';

	if (key[0] & 0x40) return '9';

	if (key[0] & 0x80) return 'C';

	if (key[1] & 0x01) return '4';

	if (key[1] & 0x02) return '5';

	if (key[1] & 0x04) return '6';

	if (key[1] & 0x08) return 'B';

	if (key[1] & 0x10) return '1';

	if (key[1] & 0x20) return '2';
```

```
	if (key[1] & 0x40) return '3';

	if (key[1] & 0x80) return 'A';

	return 0;
}

//Determine the number of LEDS
//that will be turned ON.
//Using increasing steps
unsigned char msr_to_hex(void){
	if (ADC < 32) return 0x01; //0000001
	if (ADC < 64) return 0x03; //0000011
	if (ADC < 128) return 0x07; //0000111
	if (ADC < 256) return 0x0F; //0001111
	if (ADC < 384) return 0x1F; //0011111
	if (ADC < 512) return 0x3F;;//0111111
	return 0x7F;
}

// Calculate CO_msr where Vin  = (ADC/5)/1024 and CO_msr = (1/M) * (Vin - Vgas0)
int calc_CO (void) {
	volatile float sensitivity = 129.0, Vgas0 = 0.1;
	volatile float Vin = (ADC*5.0)/1024.0; // Vin  = (ADC*Vref)/1024
	volatile float M = sensitivity * 0.0001; // CO_msr = (1/M) * (Vin -
Vgas0)
	return (int)((1/M) * (Vin - Vgas0));
}

//if the password is wrong
//we flash the LED's for 4s
void fail(){
	for (int i = 0; i < 4; i++){
		msb_flag = 0x80;
		PORTB = 0x80 | output;
		wait_msec(480);
		scan_keypad_rising_edge(); //read and ignore
		msb_flag = 0x00;
		PORTB = 0x00 | output;
		wait_msec(480);
		scan_keypad_rising_edge(); //read and ignore
	}
}


//if we login successfully
//we turn on the LED's for 4s
void login(){
	msb_flag = 0x80;
	login_flag = 1;
```

```
	PORTB = 0x80 | output; //Turn on PB7 and PB7 only
	for (int i = 0; i < 10; i++){
		wait_msec(380);
		scan_keypad_rising_edge(); //read and ignore
	}
	msb_flag = 0x00;
	login_flag = 0;
}

ISR(ADC_vect){
	CO_msr = calc_CO();
	output = msr_to_hex();
	if (CO_msr > 70){
		//if we are logged in, we always show the CO level without blinking
		//else we check the alarm flag, which is alternating between 1 and
0
		if (login_flag || alarm) PORTB = output | msb_flag;
		else PORTB = msb_flag;
	}
	else PORTB = output | msb_flag;
}

ISR(TIMER1_OVF_vect){

	ADCSRA |= (1<<ADSC); //allow ADC to interrupt and convert
	TCNT1 = 64755; //reset the Timer
	TCCR1B = 0x05;
	alarm = !alarm; //used in ADC intr for the blinking

}

int main(void){
	DDRB = 0xff; //output
	DDRC = 0xf0; //input and output

	//Initialize ADC

	ADMUX = 0x40;
	ADCSRA = 0x8F;

	//initialize timer 1

	TIMSK = 0x04; //TOIE1
	TCCR1B = 0x05;
	TCNT1 = 64755; //Timer set to 100ms

	asm("sei"); //allow interrupts


    while (1){
		ram[0] = 0; //initialize rmemory and PORTB
```

```
            ram[1] = 0;
            PORTB = 0x00;

            while(1){ //wait for the first digit
                  scan_keypad_rising_edge();
                  if ((digit[0] = keypad_to_ascii()) != 0) break;
            }

            while(1){ //wait for the second digit
                  scan_keypad_rising_edge();
                  if ((digit[1] = keypad_to_ascii()) != 0) break;
            }
            //if we get '80', we login
            if ((digit[0] == '8') && (digit[1] == '0')){
                  login();
            }
            else {
                  fail();
            }
    }
}
```