Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Εργαστήριο Μικροϋπολογιστών

3η Σειρά Ασκήσεων - AVR

Ον/μο: Τσάφος Αλέξανδρος

ΑΜ: 03118211, Ομάδα: 80

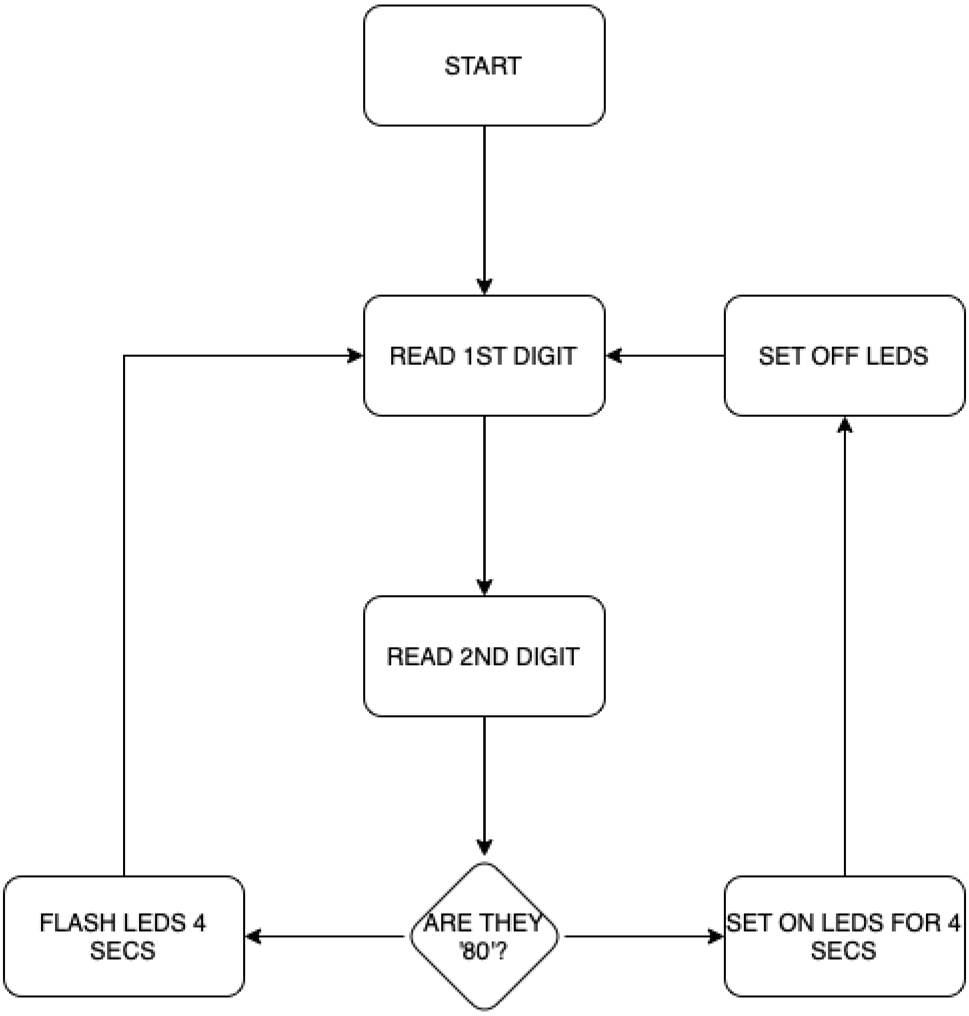
Ζήτημα 3.1

```c
#include <avr/io.h>


char ram[2], key[2], digit[2];

//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;
}
```

```
//if the password is wrong
//we flash the LED's for 4s
void fail(){
      for (int i = 0; i < 4; i++){
            PORTB = 0xFF;
            wait_msec(500);
            scan_keypad_rising_edge(); //read and ignore
            PORTB = 0x00;
            wait_msec(500);
            scan_keypad_rising_edge(); //read and ignore
      }
}


//if we login successfully
//we turn on the LED's for 4s
void login(){
      PORTB = 0xff;
      for (int i = 0; i < 10; i++){
            wait_msec(400);
            scan_keypad_rising_edge(); //read and ignore
      }
}

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

    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();
            }
    }
}
```



Ζήτημα 3.2

```
.DSEG
_tmp_: .byte 2

.CSEG
.include "m16def.inc"
```

```
.org 0x00
rjmp start

start:
      ldi r24, low(RAMEND) ;initialize stack pointer
      out SPL, r24
      ldi r24, high(RAMEND)
      out SPH, r24
      ser r24
      out DDRD, r24 ;input
      ldi r24, 0xF0
      out DDRC, r24 ;output and inout
;team 80, searching for '8' (r24 = 0x20) and '0' (r24 = 0x02) input

read1:
      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

      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
        ser r20 ;turn on the LED's
        out PORTB, r20
        ldi r20, 0x08 ; 8 time counter
        ldi r24,low(500) ; 8x500ms delays
        ldi r25,high(500)

welcome:
        rcall scan_keypad_rising_edge_sim ; read and ignore
        rcall wait_msec
        dec r20
        brne welcome      ;loop
        rcall scan_keypad_rising_edge_sim ; read and ignore
        clr r20
        out PORTB, r20 ;turn off the LED's
        rjmp read1 ;start again

wrong:
;display the message
        rcall lcd_init_sim
        ldi r24,'A'
        rcall lcd_data_sim
        ldi r24,'L'
        rcall lcd_data_sim
        ldi r24,'A'
        rcall lcd_data_sim
        ldi r24,'R'
        rcall lcd_data_sim
        ldi r24,'M'
        rcall lcd_data_sim
        ldi r24,' '
        rcall lcd_data_sim
        ldi r24,'O'
        rcall lcd_data_sim
        ldi r24,'N'
        rcall lcd_data_sim
```

```
	ldi r20, 0x04

alarm:
	ser r21 ;turn on
	out PORTB, r21
	rcall scan_keypad_rising_edge_sim ; read and ignore
	ldi r24,low(500) ; 4x2x500ms delays
	ldi r25,high(500)
	rcall wait_msec ;500ms
	clr r21 ;turn off
	out PORTB, r21
	rcall scan_keypad_rising_edge_sim ; read and ignore
	ldi r24,low(500) ; 4x2x500ms delays
	ldi r25,high(500)
	rcall wait_msec
	dec r20
	cpi r20, 0x00
	brne alarm
	rjmp read1 ;start again

; Calls Given in the PDF (Copied and Pasted)
;Everything below is not of interest
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


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
```