6η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Δρυς-Πεντζάκης Οδυσσεύς el19192
Μαρκαντωνάτος Γεράσιμος el19149

6.1

Για να απεικονίσουμε το κουμπί που πατιέται στην οθόνη σκανάρουμε το keypad εκτελώντας την scan_keypad η οποία εκτελεί 4 φορές την scan_row, απομονώνει τα 4 MSB των τιμών που παίρνει από τις σειρές και ελέγχει ποιο κουμπί είναι πατημένο.
Αφού βρούμε ποιο κουμπί είναι πατημένο καλούμε την scan_keypad_rising_edge η οποία συγκρίνει την τιμή του keypad με την τιμή του μετά από 10 ms και εφόσον οι δυο τιμές είναι διαφορετικές και η τελευταία τιμή είναι το κενό (δηλαδή έχουμε αφήσει το κουμπί) παίρνουμε την αρχική τιμή ως το κουμπί που έχουμε πατήσει.
Τέλος σύμφωνα με τον παρακάτω πίνακα ανάλογα το κουμπί που έχει πατηθεί στέλνουμε την κατάλληλη τιμή στην οθόνη ώστε να βγάλει το κουμπί που έχουμε πατήσει.

BUTTON	KEYPAD	ASCII
1	0b11100111	00110001
2	0b11010111	00110010
3	0b10110111	00110011
A	0b01110111	01000001
4	0b11101011	00110100
5	0b11011011	00110101
6	0b10111011	00110110
B	0b01111011	01000010
7	0b11101101	00110111
8	0b11011101	00111000
9	0b10111101	00111001

C	0b01111101	01000011
*	0b11101110	00101010
0	0b11011110	00110000
#	0b10111110	00100011
D	0b01111110	01000100

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#define PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
#define TWI_READ 1 // reading from twi device
#define TWI_WRITE 0 // writing to twi device
#define SCL_CLOCK 100000L // twi clock in Hz
//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2
// PCA9555 REGISTERS
void Write_2_Nibbles(uint8_t in) {
    uint8_t temp = in;
    uint8_t p = PIND;
    p &= 0x0F;
    in &= 0xF0;
    in |= p;
    PORTD = in;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    in = temp;
    in &= 0x0F;
    in = in << 4;
    in |= p;
    PORTD = in;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    return;
}

void LCD_data(uint8_t c) {
    PORTD |= 0x04;
```

```
    Write_2_Nibbles(c);
    _delay_us(100);
    return;
}

void LCD_command(uint8_t c) {
    PORTD &= 0xFB;
    Write_2_Nibbles(c);
    _delay_us(100);
    return;
}

void LCD_init(void) {
    _delay_ms(40);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(100);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(100);

    PORTD = 0x20;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(100);

    LCD_command(0x28);
    LCD_command(0x0C);
    LCD_command(0x01);
    _delay_us(5000);

    LCD_command(0x06);
}
typedef enum {
 REG_INPUT_0 = 0,
 REG_INPUT_1 = 1,
 REG_OUTPUT_0 = 2,
 REG_OUTPUT_1 = 3,
 REG_POLARITY_INV_0 = 4,
 REG_POLARITY_INV_1 = 5,
 REG_CONFIGURATION_0 = 6,
 REG_CONFIGURATION_1 = 7,
} PCA9555_REGISTERS;
//---------- Master Transmitter/Receiver ----------
```

```
#define TW_START 0x08
#define TW_REP_START 0x10
//---------------- Master Transmitter ----------------------
#define TW_MT_SLA_ACK 0x18
#define TW_MT_SLA_NACK 0x20
#define TW_MT_DATA_ACK 0x28
//---------------- Master Receiver ----------------
#define TW_MR_SLA_ACK 0x40
#define TW_MR_SLA_NACK 0x48
#define TW_MR_DATA_NACK 0x58

#define TW_STATUS_MASK 0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)
//initialize TWI clock
void twi_init(void)
{
 TWSR0 = 0; // PRESCALER_VALUE=1
 TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
}
// Read one byte from the twi device ( request more data from device)
unsigned char twi_readAck(void)
{
TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
while(!(TWCR0 & (1<<TWINT)));
 return TWDR0;
}
// Issues a start condition and sends address and transfer direction.
// return 0 = device accessible, 1= failed to access device
unsigned char twi_start(unsigned char address)
{
 uint8_t twi_status;
// send START condition
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
// wait until transmission completed
while(!(TWCR0 & (1<<TWINT)));
// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
// send device address
TWDR0 = address;
TWCR0 = (1<<TWINT) | (1<<TWEN);
// wail until transmission completed and ACK/NACK has been received
while(!(TWCR0 & (1<<TWINT)));
// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if ( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
 {
 return 1;
```

```
 }
return 0;
}
// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address)
{
 uint8_t twi_status;
 while ( 1 )
 {
// send START condition
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

 // wait until transmission completed
 while(!(TWCR0 & (1<<TWINT)));

 // check value of TWI Status Register.
 twi_status = TW_STATUS & 0xF8;
 if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;

 // send device address
 TWDR0 = address;
 TWCR0 = (1<<TWINT) | (1<<TWEN);

 // wail until transmission completed
 while(!(TWCR0 & (1<<TWINT)));

 // check value of TWI Status Register.
 twi_status = TW_STATUS & 0xF8;
 if ( (twi_status == TW_MT_SLA_NACK )||(twi_status ==TW_MR_DATA_NACK) )
 {
 /* device busy, send stop condition to terminate write operation */
 TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

 // wait until stop condition is executed and bus released
 while(TWCR0 & (1<<TWSTO));

 continue;
 }
break;
 }
}
// Send one byte to twi device, Return 0 if write successful or 1 if write failed
unsigned char twi_write( unsigned char data )
{
// send data to the previously addressed device
TWDR0 = data;
TWCR0 = (1<<TWINT) | (1<<TWEN);
```

```
// wait until transmission completed

while(!(TWCR0 & (1<<TWINT)));
if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
return 0;
}
// Send repeated start condition, address, transfer direction
//Return: 0 device accessible
// 1 failed to access device
unsigned char twi_rep_start(unsigned char address)
{
 return twi_start( address );
}
// Terminates the data transfer and releases the twi bus
void twi_stop(void)
{
 // send stop condition
TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
// wait until stop condition is executed and bus released
while(TWCR0 & (1<<TWSTO));
}
unsigned char twi_readNak(void)
{
        TWCR0 = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR0 & (1<<TWINT)));

    return TWDR0;
}
void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
{
 twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
 twi_write(reg);
 twi_write(value);
 twi_stop();
}
uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
{
uint8_t ret_val;

 twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
 twi_write(reg);
 twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
 ret_val = twi_readNak();
 twi_stop();

 return ret_val;
}
uint8_t scan_row (int row){
```

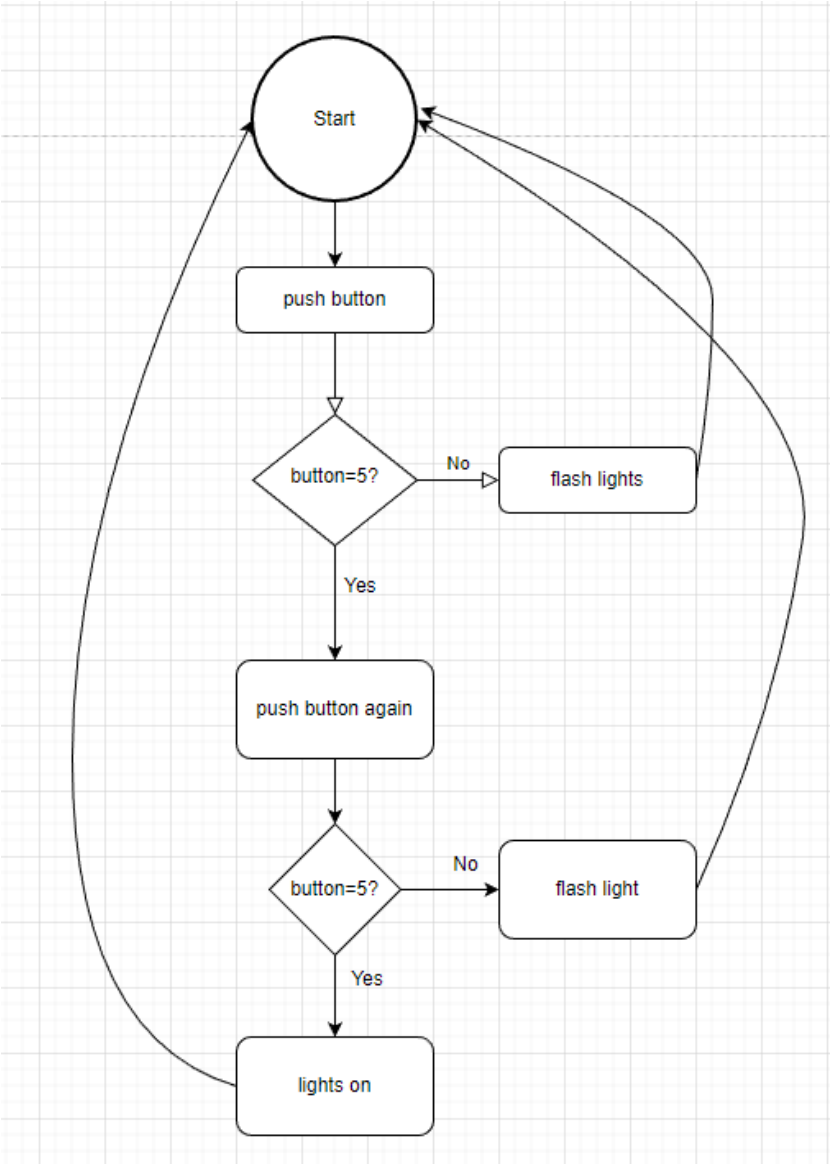
```
    if (row==1){
       PCA9555_0_write(REG_OUTPUT_1, 0xFE);

        uint8_t temp ;
       temp =PCA9555_0_read(REG_INPUT_1);
       return temp;


    }
    if (row==2){
       PCA9555_0_write(REG_OUTPUT_1, 0xFD);

        uint8_t temp ;
        temp =PCA9555_0_read(REG_INPUT_1);
        return temp;

    }
    if (row==3){
       PCA9555_0_write(REG_OUTPUT_1, 0xFB);

        uint8_t temp ;
       temp =PCA9555_0_read(REG_INPUT_1);
       return temp;

    }
    if (row==4){
       PCA9555_0_write(REG_OUTPUT_1, 0xF7);

        uint8_t temp ;
       temp =PCA9555_0_read(REG_INPUT_1);
       return temp;


    }
}


uint8_t scan_keypad (){
    uint8_t row1, row2, row3, row4;
    row1= scan_row (1);
    row2= scan_row (2);
    row3= scan_row (3);
    row4= scan_row (4);
    uint8_t temp1= row1&0b11110000;
    uint8_t temp2= row2&0b11110000;
    uint8_t temp3= row3&0b11110000;
    uint8_t temp4= row4&0b11110000;
    if (temp1!=0b11110000)
```

```
        return row1;
    else if (temp2!=0b11110000)
        return row2;
     else if (temp3!=0b11110000)
        return row3;
     else if (temp4!=0b11110000)
        return row4;
    else return 0;


}




uint8_t scan_keypad_rising_edge() {
     uint8_t start, finish;
    start = scan_keypad();
    _delay_ms(10);
    finish = scan_keypad();

    if ((start!=finish) && (finish==0) ) {
        return start;}
    return 0;
}

uint8_t keypad_to_ascii() {
     uint8_t chari;
     chari = scan_keypad_rising_edge(); //
     if (chari == 0b11101110) return 0b00101010; //*
     if (chari == 0b11011110) return 0b00110000; //0
     if (chari == 0b10111110) return 0b00100011;
     if (chari == 0b01111110) return 0b01000100;
     if (chari == 0b11101101) return 0b00110111;
     if (chari == 0b11011101) return 0b00111000;
     if (chari == 0b10111101) return 0b00111001;
     if (chari == 0b01111101) return 0b01000011;
     if (chari == 0b11101011) return 0b00110100;//B
     if (chari == 0b11011011) return 0b00110101;
     if (chari == 0b10111011) return 0b00110110;
     if (chari == 0b01111011) return 0b01000010;
     if (chari == 0b11100111) return 0b00110001;
     if (chari == 0b11010111) return 0b00110010;
     if (chari == 0b10110111) return 0b00110011;
     if (chari == 0b01110111) return 0b01000001;
     return 0;
}
```

```
uint8_t key;

int main(void) {
 twi_init();
 DDRD = 0xFF;

 PCA9555_0_write(REG_CONFIGURATION_0, 0x00);
 PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
 LCD_init();
    _delay_ms(2);


while(1) {
    key = keypad_to_ascii();
    if (key != 0)  {
    LCD_command(0x01);
    _delay_ms(2);
    LCD_data(key);
    }
    }
}
```

6.2

Χρησιμοποιώντας τις συναρτήσεις από την προηγούμενη άσκηση ελέγχουμε αν η τιμή που πατάμε στο keypad είναι το 55, αφού είμαστε η ομάδα 55. Στην περίπτωση που πρώτα πατηθεί αριθμός διάφορος του 5 δεν περιμένουμε να πατηθεί άλλος αριθμός για να αναβοσβήσουμε τα φωτάκια ενώ αν πρώτα πατηθεί το 5 και μετά αριθμός διάφορος του 5 αναβοσβήνουμε τότε τα λαμπάκια. Αν επιτυχώς πατήσουμε 55 τότε ανάβουμε τα λαμπάκια για 5 δευτερόλεπτα και μετά τα κλείνουμε.




```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#define PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
#define TWI_READ 1 // reading from twi device
#define TWI_WRITE 0 // writing to twi device
#define SCL_CLOCK 100000L // twi clock in Hz
//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2
// PCA9555 REGISTERS
void Write_2_Nibbles(uint8_t in) {
    uint8_t temp = in;
    uint8_t p = PIND;
```

```
    p &= 0x0F;
    in &= 0xF0;
    in |= p;
    PORTD = in;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    in = temp;
    in &= 0x0F;
    in = in << 4;
    in |= p;
    PORTD = in;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    return;
}

void LCD_data(uint8_t c) {
    PORTD |= 0x04;
    Write_2_Nibbles(c);
    _delay_us(100);
    return;
}

void LCD_command(uint8_t c) {
    PORTD &= 0xFB;
    Write_2_Nibbles(c);
    _delay_us(100);
    return;
}

void LCD_init(void) {
    _delay_ms(40);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(100);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(100);

    PORTD = 0x20;
    PORTD |= 0x08;
    PORTD &= 0xF7;
```

```
    _delay_us(100);

    LCD_command(0x28);
    LCD_command(0x0C);
    LCD_command(0x01);
    _delay_us(5000);

    LCD_command(0x06);
}
typedef enum {
 REG_INPUT_0 = 0,
 REG_INPUT_1 = 1,
 REG_OUTPUT_0 = 2,
 REG_OUTPUT_1 = 3,
 REG_POLARITY_INV_0 = 4,
 REG_POLARITY_INV_1 = 5,
 REG_CONFIGURATION_0 = 6,
 REG_CONFIGURATION_1 = 7,
} PCA9555_REGISTERS;
//----------- Master Transmitter/Receiver -------------------
#define TW_START 0x08
#define TW_REP_START 0x10
//---------------- Master Transmitter ----------------------
#define TW_MT_SLA_ACK 0x18
#define TW_MT_SLA_NACK 0x20
#define TW_MT_DATA_ACK 0x28
//---------------- Master Receiver ----------------
#define TW_MR_SLA_ACK 0x40
#define TW_MR_SLA_NACK 0x48
#define TW_MR_DATA_NACK 0x58

#define TW_STATUS_MASK 0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)
//initialize TWI clock
void twi_init(void)
{
 TWSR0 = 0; // PRESCALER_VALUE=1
 TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
}
// Read one byte from the twi device ( request more data from device)
unsigned char twi_readAck(void)
{
TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
while(!(TWCR0 & (1<<TWINT)));
 return TWDR0;
}
// Issues a start condition and sends address and transfer direction.
// return 0 = device accessible, 1= failed to access device
```

```
unsigned char twi_start(unsigned char address)
{
 uint8_t twi_status;
// send START condition
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
// wait until transmission completed
while(!(TWCR0 & (1<<TWINT)));
// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
// send device address
TWDR0 = address;
TWCR0 = (1<<TWINT) | (1<<TWEN);
// wail until transmission completed and ACK/NACK has been received
while(!(TWCR0 & (1<<TWINT)));
// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if ( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
 {
 return 1;
 }
return 0;
}
// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address)
{
 uint8_t twi_status;
 while ( 1 )
 {
// send START condition
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

 // wait until transmission completed
 while(!(TWCR0 & (1<<TWINT)));

 // check value of TWI Status Register.
 twi_status = TW_STATUS & 0xF8;
 if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;

 // send device address
 TWDR0 = address;
 TWCR0 = (1<<TWINT) | (1<<TWEN);

 // wail until transmission completed
 while(!(TWCR0 & (1<<TWINT)));

 // check value of TWI Status Register.
```

```
 twi_status = TW_STATUS & 0xF8;
 if ( (twi_status == TW_MT_SLA_NACK )||(twi_status ==TW_MR_DATA_NACK) )
 {
 /* device busy, send stop condition to terminate write operation */
 TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

 // wait until stop condition is executed and bus released
 while(TWCR0 & (1<<TWSTO));

 continue;
 }
break;
 }
}
// Send one byte to twi device, Return 0 if write successful or 1 if write failed
unsigned char twi_write( unsigned char data )
{
// send data to the previously addressed device
TWDR0 = data;
TWCR0 = (1<<TWINT) | (1<<TWEN);
// wait until transmission completed

while(!(TWCR0 & (1<<TWINT)));
if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
return 0;
}
// Send repeated start condition, address, transfer direction
//Return: 0 device accessible
// 1 failed to access device
unsigned char twi_rep_start(unsigned char address)
{
 return twi_start( address );
}
// Terminates the data transfer and releases the twi bus
void twi_stop(void)
{
 // send stop condition
TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
// wait until stop condition is executed and bus released
while(TWCR0 & (1<<TWSTO));
}
unsigned char twi_readNak(void)
{
        TWCR0 = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR0 & (1<<TWINT)));

    return TWDR0;
}
```

```
void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
{
 twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
 twi_write(reg);
 twi_write(value);
 twi_stop();
}
uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
{
uint8_t ret_val;

 twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
 twi_write(reg);
 twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
 ret_val = twi_readNak();
 twi_stop();

 return ret_val;
}
uint8_t scan_row (int row){
   if (row==1){
      PCA9555_0_write(REG_OUTPUT_1, 0xFE);

       uint8_t temp ;
      temp =PCA9555_0_read(REG_INPUT_1);
      return temp;


   }
   if (row==2){
      PCA9555_0_write(REG_OUTPUT_1, 0xFD);

       uint8_t temp ;
       temp =PCA9555_0_read(REG_INPUT_1);
       return temp;

   }
   if (row==3){
      PCA9555_0_write(REG_OUTPUT_1, 0xFB);

       uint8_t temp ;
      temp =PCA9555_0_read(REG_INPUT_1);
      return temp;

   }
   if (row==4){
      PCA9555_0_write(REG_OUTPUT_1, 0xF7);
```

```
        uint8_t temp ;
       temp =PCA9555_0_read(REG_INPUT_1);
       return temp;


   }
}


uint8_t scan_keypad (){
    uint8_t row1, row2, row3, row4;
    row1= scan_row (1);
    row2= scan_row (2);
    row3= scan_row (3);
    row4= scan_row (4);
    uint8_t temp1= row1&0b11110000;
    uint8_t temp2= row2&0b11110000;
    uint8_t temp3= row3&0b11110000;
    uint8_t temp4= row4&0b11110000;
    if (temp1!=0b11110000)
        return row1;
   else if (temp2!=0b11110000)
        return row2;
    else if (temp3!=0b11110000)
        return row3;
    else if (temp4!=0b11110000)
        return row4;
   else return 0;


}




uint8_t scan_keypad_rising_edge() {
    uint8_t start, finish;
   start = scan_keypad();
   _delay_ms(10);
   finish = scan_keypad();

   if ((start!=finish) && (finish==0) ) {
      return start;}
   return 0;
}

uint8_t keypad_to_ascii() {
    uint8_t chari;
```

```
    chari = scan_keypad_rising_edge(); //
    if (chari == 0b11101110) return 0b00101010; //*
    if (chari == 0b11011110) return 0b00110000; //0
    if (chari == 0b10111110) return 0b00100011;
    if (chari == 0b01111110) return 0b01000100;
    if (chari == 0b11101101) return 0b00110111;
    if (chari == 0b11011101) return 0b00111000;
    if (chari == 0b10111101) return 0b00111001;
    if (chari == 0b01111101) return 0b01000011;
    if (chari == 0b11101011) return 0b00110100;//B
    if (chari == 0b11011011) return 0b00110101;
    if (chari == 0b10111011) return 0b00110110;
    if (chari == 0b01111011) return 0b01000010;
    if (chari == 0b11100111) return 0b00110001;
    if (chari == 0b11010111) return 0b00110010;
    if (chari == 0b10110111) return 0b00110011;
    if (chari == 0b01110111) return 0b01000001;
    return 0;
}

uint8_t key1,key2;

int main(void) {
 twi_init();
 DDRD = 0xFF;
 DDRB = 0xFF;
 PORTB = 0;
 PCA9555_0_write(REG_CONFIGURATION_0, 0x00);
 PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
 LCD_init();
    _delay_ms(2);


while(1) {
    start:
    while(1) {
    key1 = keypad_to_ascii();
    if(key1 !=0) break;
    }
    if ( key1 != 0x35) {
        goto ff;}

    while(1) {
    key2 = keypad_to_ascii();
    if(key2 !=0) break;
    }
    if ( key2 != 0x35) {
        goto ff;}
```

```
    PORTB = 0xFF;
    _delay_ms(4000);
    PORTB = 0;
    _delay_ms(1000);
    }

ff:
    for (int i=0;i<10;i++) {
    PORTB = 0xFF;
    _delay_ms(250);
    PORTB = 0;
    _delay_ms(250);
    }
goto start;
}
```