7η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Δρυς-Πεντζάκης Οδυσσεύς el19192
Μαρκαντωνάτος Γεράσιμος el19149

7.1

Μεταφράζουμε τις εντολές που μας δόθηκαν από Assembly σε C ώστε να μπορούμε να κάνουμε reset, να λάβουμε byte και να στείλουμε byte.
Έπειτα φτιάχνουμε την συνάρτηση όπως περιγράφει η εκφώνηση, και κάνουμε πράξεις στον 16-bit αριθμό που λαμβάνουμε από τον αισθητήρα ώστε να τον απεικονίσουμε σωστά στην οθόνη. Τέλος, επειδή ο αισθητήρας χρησιμοποιεί το PORTD, συνδέουμε την οθόνη LCD μέσω του PORT EXPANDER.

```
#include <avr/io.h>
#define F_CPU   16000000UL
#include <util/delay.h>
#include<avr/interrupt.h>
#define cbi(reg,bit) (reg &= ~(1 << bit))
#define sbi(reg,bit) (reg |= (1 << bit))
#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
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;
}
//Read one byte from the twi device, read is followed by a stop condition
unsigned char twi_readNak(void)
{
TWCR0 = (1<<TWINT) | (1<<TWEN);
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));
}
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 one_wire_receive_bit(){
        uint8_t bit,temp;
        sbi(DDRD,PD4);
        cbi(PORTD,PD4);
        _delay_us(2);
        cbi(DDRD,PD4);
        cbi(PORTD,PD4);
        _delay_us(10);
        temp = (PIND & 0x10);
        bit = 0x00;
        if (temp == 0x10) bit = 0x01;
        _delay_us(49);
        return bit;
}

uint8_t one_wire_receive_byte(){
        uint8_t bit;
        uint8_t byte = 0x00;
        uint8_t i = 0x08;
        while(i != 0){
                bit = one_wire_receive_bit();
                byte = (byte >> 1);
                if (bit == 0x01) bit = 0x80;
                byte = (byte | bit);
                i--;
        }
        return byte;
}

void one_wire_transmit_bit(uint8_t bit){
        sbi(DDRD,PD4);
        cbi(PORTD,PD4);
        _delay_us(2);
        if (bit == 0x01) sbi(PORTD,PD4);
        if (bit == 0x00) cbi(PORTD,PD4);
        _delay_us(58);
        cbi(DDRD,PD4);
        cbi(PORTD,PD4);
        _delay_us(1);
        return;
}

void one_wire_transmit_byte(uint8_t byte){
        uint8_t bit;
        uint8_t i = 0x08;
        while(i != 0){
                bit = (byte & 0x01);
                one_wire_transmit_bit(bit);
                byte = (byte >> 1);
                i--;
        }
        return;
}
```

```
uint8_t one_wire_reset(){
            sbi(DDRD,PD4);
            cbi(PORTD,PD4);
            _delay_us(480);
            cbi(DDRD,PD4);
            cbi(PORTD,PD4);
            _delay_us(100);
            uint8_t temp = PIND;
            _delay_us(380);
            temp = (temp & 0x10);
            uint8_t res = 0x00;
            if (temp == 0x00)
        res = 0x01;
            return res;
}

/* LCD - EXPANDER */

void write_2_nibbles(uint8_t c) {
    uint8_t temp = c;
    uint8_t prev = PCA9555_0_read(REG_INPUT_0);
    prev &= 0x0F;
    c &= 0xF0;
    c |= prev;
    PCA9555_0_write(REG_OUTPUT_0, c);
    c |= 0x08;
    PCA9555_0_write(REG_OUTPUT_0, c);
    c &= 0xF7;
    PCA9555_0_write(REG_OUTPUT_0, c);

    c = temp;
    c &= 0x0F;
    c = c << 4;
    c |= prev;
    PCA9555_0_write(REG_OUTPUT_0, c);
    c |= 0x08;
    PCA9555_0_write(REG_OUTPUT_0, c);
    c &= 0xF7;
    PCA9555_0_write(REG_OUTPUT_0, c);

    return;
}

void LCD_data(uint8_t c) {
    uint8_t temp = PCA9555_0_read(REG_INPUT_0);
    temp |= 0x04;
    PCA9555_0_write(REG_OUTPUT_0, temp);
    write_2_nibbles(c);
    _delay_us(100);
    return;
}

void LCD_command(uint8_t c) {
    uint8_t temp = PCA9555_0_read(REG_INPUT_0);
    temp &= 0xFB;
    PCA9555_0_write(REG_OUTPUT_0, temp);
    write_2_nibbles(c);
    _delay_us(100);
    return;
}

void LCD_init(void) {
    _delay_ms(40);

    PCA9555_0_write(REG_OUTPUT_0, 0x30);
    PCA9555_0_write(REG_OUTPUT_0, 0x38);
    PCA9555_0_write(REG_OUTPUT_0, 0x30);

    _delay_us(100);

    PCA9555_0_write(REG_OUTPUT_0, 0x30);
    PCA9555_0_write(REG_OUTPUT_0, 0x38);
    PCA9555_0_write(REG_OUTPUT_0, 0x30);

    _delay_us(100);

    PCA9555_0_write(REG_OUTPUT_0, 0x20);
    PCA9555_0_write(REG_OUTPUT_0, 0x28);
    PCA9555_0_write(REG_OUTPUT_0, 0x20);

    _delay_us(100);

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

    LCD_command(0x06);
    return;
}
uint8_t temp_lo, temp_hi, temp_sign, temp_dec,one,two,three,four;
uint16_t temp_final, temp_hi_16, temp_final_o;
int dec_1 = 0;
        int dec_2 = 0;
        int dec_3 = 0;
        int dec_4 = 0;
        int sum=0;

uint16_t routine() {
                if (!one_wire_reset()) {
            return 0x8000;


                }
                one_wire_transmit_byte(0xCC);
                one_wire_transmit_byte(0x44);
                while(one_wire_receive_bit() != 0x01);
                if (!one_wire_reset()) {

            return 0x8000;

                }

                one_wire_transmit_byte(0xCC);
                one_wire_transmit_byte(0xBE);
                temp_lo = one_wire_receive_byte();
                temp_hi = one_wire_receive_byte();
        temp_hi_16 = temp_hi << 8;
        temp_final = (temp_hi_16 + temp_lo);
        return temp_final;

}
```

7.2

```
#include <avr/io.h>
#define         F_CPU   16000000UL
#include <util/delay.h>
#include<avr/interrupt.h>
#define cbi(reg,bit) (reg &= ~(1 << bit))
#define sbi(reg,bit) (reg |= (1 << bit))
#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
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;
}
//Read one byte from the twi device, read is followed by a stop condition
unsigned char twi_readNak(void)
{
TWCR0 = (1<<TWINT) | (1<<TWEN);
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));
}
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 one_wire_receive_bit(){
        uint8_t bit,temp;
```

```
	sbi(DDRD,PD4);
	cbi(PORTD,PD4);
	_delay_us(2);
	cbi(DDRD,PD4);
	cbi(PORTD,PD4);
	_delay_us(10);
	temp = (PIND & 0x10);
	bit = 0x00;
	if (temp == 0x10) bit = 0x01;
	_delay_us(49);
	return bit;
}

uint8_t one_wire_receive_byte(){
	uint8_t bit;
	uint8_t byte = 0x00;
	uint8_t i = 0x08;
	while(i != 0){
		bit = one_wire_receive_bit();
		byte = (byte >> 1);
		if (bit == 0x01) bit = 0x80;
		byte = (byte | bit);
		i--;
	}
	return byte;
}

void one_wire_transmit_bit(uint8_t bit){
	sbi(DDRD,PD4);
	cbi(PORTD,PD4);
	_delay_us(2);
	if (bit == 0x01) sbi(PORTD,PD4);
	if (bit == 0x00) cbi(PORTD,PD4);
	_delay_us(58);
	cbi(DDRD,PD4);
	cbi(PORTD,PD4);
	_delay_us(1);
	return;
}

void one_wire_transmit_byte(uint8_t byte){
	uint8_t bit;
	uint8_t i = 0x08;
	while(i != 0){
		bit = (byte & 0x01);
		one_wire_transmit_bit(bit);
		byte = (byte >> 1);
		i--;
```

```
        }
        return;
}

uint8_t one_wire_reset(){
        sbi(DDRD,PD4);
        cbi(PORTD,PD4);
        _delay_us(480);
        cbi(DDRD,PD4);
        cbi(PORTD,PD4);
        _delay_us(100);
        uint8_t temp = PIND;
        _delay_us(380);
        temp = (temp & 0x10);
        uint8_t res = 0x00;
        if (temp == 0x00)
      res = 0x01;
        return res;
}

/* LCD - EXPANDER */

void write_2_nibbles(uint8_t c) {
   uint8_t temp = c;
   uint8_t prev = PCA9555_0_read(REG_INPUT_0);
   prev &= 0x0F;
   c &= 0xF0;
   c |= prev;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c |= 0x08;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c &= 0xF7;
   PCA9555_0_write(REG_OUTPUT_0, c);

   c = temp;
   c &= 0x0F;
   c = c << 4;
   c |= prev;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c |= 0x08;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c &= 0xF7;
   PCA9555_0_write(REG_OUTPUT_0, c);

   return;
}

void LCD_data(uint8_t c) {
   uint8_t temp = PCA9555_0_read(REG_INPUT_0);
   temp |= 0x04;
   PCA9555_0_write(REG_OUTPUT_0, temp);
   write_2_nibbles(c);
   _delay_us(100);
   return;
}

void LCD_command(uint8_t c) {
   uint8_t temp = PCA9555_0_read(REG_INPUT_0);
   temp &= 0xFB;
   PCA9555_0_write(REG_OUTPUT_0, temp);
   write_2_nibbles(c);
   _delay_us(100);
   return;
}

void LCD_init(void) {
   _delay_ms(40);

   PCA9555_0_write(REG_OUTPUT_0, 0x30);
   PCA9555_0_write(REG_OUTPUT_0, 0x38);
   PCA9555_0_write(REG_OUTPUT_0, 0x30);

   _delay_us(100);

   PCA9555_0_write(REG_OUTPUT_0, 0x30);
   PCA9555_0_write(REG_OUTPUT_0, 0x38);
   PCA9555_0_write(REG_OUTPUT_0, 0x30);

   _delay_us(100);

   PCA9555_0_write(REG_OUTPUT_0, 0x20);
   PCA9555_0_write(REG_OUTPUT_0, 0x28);
   PCA9555_0_write(REG_OUTPUT_0, 0x20);

   _delay_us(100);

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

   LCD_command(0x06);
   return;
}
uint8_t temp_lo, temp_hi, temp_sign, temp_dec,one,two,three,four;
uint16_t temp_final, temp_hi_16, temp_final_o;
```

```
int dec_1 = 0;
        int dec_2 = 0;
        int dec_3 = 0;
        int dec_4 = 0;
        int sum=0;

int main(void)
{

    twi_init();

            DDRB = 0x3F;
            DDRD = 0xFF;

    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);


    LCD_init();
    _delay_ms(2);
            while (1)
    {

                        if (!one_wire_reset()) {
            LCD_command(0x01);
            _delay_ms(2);
                                    LCD_data('N');
            LCD_data('O');
            LCD_data(' ');
            LCD_data('D');
            LCD_data('E');
            LCD_data('V');
            LCD_data('I');
            LCD_data('C');
            LCD_data('E');

                                    continue;
                        }
                        one_wire_transmit_byte(0xCC);
                        one_wire_transmit_byte(0x44);
                        while(one_wire_receive_bit() != 0x01);
                        if (!one_wire_reset()) {
            LCD_command(0x01);
            _delay_ms(2);
                                    LCD_data('N');
            LCD_data('O');
            LCD_data(' ');
            LCD_data('D');
```

```
   LCD_data('E');
   LCD_data('V');
   LCD_data('I');
   LCD_data('C');
   LCD_data('E');

                    continue;
            }

            one_wire_transmit_byte(0xCC);
            one_wire_transmit_byte(0xBE);
            temp_lo = 0xa9;
            temp_hi = 0xff;
temp_dec = temp_lo & 0x0F;
            temp_sign = temp_hi & 0xF8;
temp_hi_16 = temp_hi << 8;
temp_final = (temp_hi_16 + temp_lo);

if (temp_sign == 0xF8) {
   temp_final_o = 65534-temp_final;
}

            else
   temp_final_o = temp_final;

temp_dec = temp_final_o&15;

 dec_1 = 0;
 dec_2 = 0;
 dec_3 = 0;
 dec_4 = 0;
 sum=0;
 one = temp_dec&0x08;
if (one == 8)sum = sum + 5000;
 one = temp_dec&0x04;
if ( one == 4) sum = sum + 2500;
one = temp_dec&0x02;
if (one == 2) sum = sum + 1250;
one = temp_dec&0x01;
if (one == 1) sum = sum + 625;

LCD_data('-');
while (sum >= 1000) {
   dec_1++;
   sum = sum - 1000;
}

while (sum >= 100) {
    dec_2++;
    sum = sum - 100;
}

while (sum >= 10) {
    dec_3++;
    sum = sum - 10;
}

while (sum >= 1) {
    dec_4++;
    sum = sum - 1;
}




int i = 0;
int j = 0;
int k = 0;

temp_final_o = temp_final_o>>4;

while (temp_final_o >= 100) {
    i++;
    temp_final_o = temp_final_o - 100;
}

while (temp_final_o >= 10) {
    j++;
    temp_final_o = temp_final_o - 10;
}

while (temp_final_o >= 1) {
    k++;
    temp_final_o = temp_final_o - 1;
}


i |= 0x30;
j |= 0x30;
k |= 0x30;
dec_1 |= 0x30;
dec_2 |= 0x30;
dec_3 |= 0x30;
dec_4 |= 0x30;

LCD_command(0x01);
```

```
        _delay_ms(2);
        if (temp_sign == 0xF8)
           LCD_data('-');
        else
           LCD_data('+');

        if(i!=0x30)LCD_data(i);
        if(j!=0x30)LCD_data(j);
        LCD_data(k);
        LCD_data('.');
        LCD_data(dec_1);
        LCD_data(dec_2);
        LCD_data(dec_3);
        LCD_data(dec_4);
        LCD_data(223);
        LCD_data('C');
    }
}
```