Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών

Εργαστήριο Μικροϋπολογιστών

Έκθεση 7ης Εργαστηριακής Άσκησης

Στοιχεία:

Ομάδα: **39**

Μέλη: **Ευάγγελος Μυργιώτης, Αριστοτέλης Γρίβας**

Ζήτημα 7.1 :

Ο βασικός στόχος αυτής της άσκησης ήταν η υλοποίηση των κατάλληλων συναρτήσεων για την επίτευξη σειριακής επικοινωνίας. Το μοναδικό σημείο ενδιαφέροντος είναι ότι πρέπει να περιμένουμε μέχρι το θερμόμετρο να έχει μετρήσει την θερμοκρασία ελέγχοντας συνέχεια την κατάλληλη σηματοδότηση από τον DS18B20.

Κώδικας :

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.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



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)
```

```
  uint8_t  t1,t2;

unsigned char keypad[16] = { 0b00101010, 0b00110000, 0b00100011, 0b01000100,
0b00110111, 0b00111000, 0b00111001, 0b01000011, 0b00110100, 0b00110101,
0b00110110, 0b01000010, 0b00110001, 0b00110010, 0b00110011, 0b01000001};




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;
}

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;
}
void write_2_nibbles(unsigned char q){
    int temp = q;   // r24 = temp = q
    int e = PIND;   //in r25, PIND
    e = e & 0x0f;    // andi r25, 0x0f
    temp = temp & 0xf0; // andi r24, 0xf0
    temp = temp + e;    // add r24, r25
    PORTD= temp;  //out PORTD ,r24
    PORTD |= 0x08;  //sbi PORTD ,3
    _delay_ms(1);
    PORTD &= 0b11110111; //cbi PORTD ,3
    _delay_ms(1);

    temp = q;   //pop r24
    temp = temp<<4;  //swap r24
    temp = temp & 0xf0;  //andi r24, 0xf0
    temp = temp + e;   //add r24,r25
    PORTD= temp;   //portd, r24
    PORTD |= 0x08;
    _delay_ms(1);
    PORTD &= 0b11110111;
    _delay_ms(2);

return;
}


void lcd_data(unsigned char p){
    PORTD |= 0x04;
    write_2_nibbles(p);
    _delay_ms(10);
    return;
}

void lcd_command(unsigned char z){
    PORTD &= 0b11111011;
    write_2_nibbles(z);
    _delay_ms(10);
    return;
}


void lcd_init(){

    _delay_ms(40); //wait_msec
    PORTD = 0x30;     //PORTD = r24 = 0x30
    PORTD |= 0x08;   // sbi PORTD, 3
    _delay_ms(1);
    PORTD &= 0b11110111; //cbi PORTD, 3
    _delay_ms(2);

     PORTD = 0x30;     //PORTD = r24 = 0x30
    PORTD |= 0x08;   // sbi PORTD, 3
    _delay_ms(1);
    PORTD &= 0b11110111; //cbi PORTD, 3
    _delay_ms(2);

    PORTD = 0x30;   //PORTD = r24 = 0x30
    PORTD |= 0x08;
    _delay_ms(1);
    PORTD &= 0b11110111;
    _delay_ms(2);

    PORTD = 0x20;
    PORTD |= 0x08;
    _delay_ms(1);
    PORTD &= 0b11110111;
    _delay_ms(2);
    lcd_command(0x28);
    lcd_command(0x0c);
    lcd_command(0x01);

    _delay_ms(10);

    lcd_command(0x06);

    _delay_ms(10);

    return;
}


uint8_t one_wire_reset(){

      uint8_t temp;
      DDRD |= 0b00010000;  // sbi DDRD, PD4
      PORTD &= 0b11101111; // cbi PORTD, PD4
      _delay_us(480);   //call wait_usec
      DDRD &= 0b11101111; //cbi DDRD, PD4
      PORTD &= 0b11101111; //cbi PORTD, PD4
      _delay_us(100);     //rcall wait_usec
      temp = PIND;            //temp = r24 = PORTD
      _delay_us(380);
      if(temp & 0b00010000) {       //if PD4 = 0 return 1 in temp = r24
            temp = 0;
            return temp;
      }
      else{                         //else return 0 in temp = r24
       temp = 1;
       return temp;
      }

}


uint8_t one_wire_receive_bit(){
      uint8_t temp;
      DDRD |= 0b00010000;  //sbi DDRD, PD4
      PORTD &= 0b11101111;  //cbi PORTD, PD4
      _delay_us(2);
      DDRD &= 0b11101111;   //cbi DDRD, PD4
      PORTD &= 0b11101111;   //cbi PORTD, PD4
      _delay_us(10);

      if(PIND & 0b00010000) {        //if PD4 = 1 return 0 in temp = r24 and delay(49)
            temp = 1;

      }
      else{                          //else return 0 in temp = r24 and delay(49)
       temp = 0;
 }



 _delay_us(49);
 return temp;
}



void one_wire_transmit_bit(uint8_t x){

      uint8_t e;
      e = x;             // x will be r24 when the function is called
      DDRD |= 0b00010000;  //sbi DDRD, PD4
      PORTD &= 0b11101111;  //cbi PORTD, PD4
      _delay_us(2);

      if(e & 0x01){
            PORTD |= 0b00010000;  //sbi PORTD, PD4
      }
      else{
            PORTD &= 0b11101111;  //cbi PORTD, PD4
      }
      _delay_us(58);
      DDRD &= 0b11101111; //cbi DDRD, PD4
      PORTD &= 0b11101111; //bi PORTD, PD4
      _delay_us(1);

}


uint8_t one_wire_receive_byte(){

 uint8_t temp;
      uint8_t temp2 = 0;       //temp2 = r26
      for(int i = 0; i < 8; i++){
            temp = one_wire_receive_bit();  //rcall one_wire_receive_bit
            temp2 = temp2 >> 1;     //lsr r26
            if(temp & 1) temp = 0x80;   //if r24 == 1,r24 = 0x80
            temp2 = temp2 | temp;                  //r26 or r24
      }
      temp = temp2;  //mov r24, r26
      return temp;
}
```

```
void one_wire_transmit_byte(uint8_t x){

   uint8_t temp;
        uint8_t temp2 = x;        //r26 = x = r24   mov r26, r24
        for(int i = 0; i < 8; i++){
                temp = 0;                               //clr r24
                if(temp2 & 0x01) temp = 0x01; //ldi r24 ,0x01
                one_wire_transmit_bit(temp);
                temp2 = temp2 >> 1;   //lsr r26

        }
}


int main(){
   DDRB = 0xFF;
   DDRC = 0xFF;

   uint8_t x;


   x = one_wire_reset();
    if(x == 0) {
      t1 = 0x00;
      t2 = 0x80;

   }

   one_wire_transmit_byte(0xCC);
   one_wire_transmit_byte(0x44);
   while(!(one_wire_receive_bit())) {
   }
   x = one_wire_reset();
   one_wire_transmit_byte(0xCC);
   one_wire_transmit_byte(0xBE);
   t1 = one_wire_receive_byte();
   t2 = one_wire_receive_byte();

   PORTB = t1;
   PORTC = t1 >> 4;
}
```

Ζήτημα 7.2 :

Στο ζήτημα αυτό κάνουμε κατάλληλη επεξεργασία της ένδειξης θερμοκρασίας του DS18B20, ώστε να το εμφανίσουμε δεκαδική μορφή στον LCD display.

Συγκεκριμένα, αρχικά αποθηκεύουμε την 16-bit τιμή σε έναν διπλό καταχωρητή. Ανάλογα από το MSB, μπορούμε να κρίνουμε το πρόσημο του αριθμού, στην συνέχεια κοιτάμε το 4ο bit (ακρίβειας 0.5^{o}C) για να προσθέσουμε αργότερα 0,5 στο αποτέλεσμα. Αφού το έχω κάνει αυτό, γνωρίζοντας αν ο αριθμός είναι αρνητικός θα κάνουμε αντίστροφη συμπλήρωση ως προς 2 άμα είναι. Μόνο τότε θα κάνουμε shift 4 θέσεις το αποτέλεσμα ώστε το νέο LSB να αντιστοιχεί σε 1^{o}C. Η διαδικασία μετατροπής από εδώ και μετά σε δεκαδική μορφή είναι απλή. Έχοντας υπολογίσει τα ψηφία, τα παρουσιάζουμε στην οθόνη με το κατάλληλο πρόσημο και προσθέτουμε το 0.5 αν χρείαζεται.

Να σημειωθεί ότι άμα δεν υπάρχει συνδεδεμένη συσκευή, το get_temperature θα επιστρέψει την τιμή 0x8000. Σε αυτήν την περίπτωση το πρόγραμμα δεν θα προσπαθήσει να κάνει μετατροπή στην δεκαδική μορφή και απλά θα συνεχίσει να κοιτάει για συνδεδεμένη συσκευή.

Τελευταία, εφόσον το LCD είναι συνδεδεμένο δια μέσω του PORTD, φροντίζουμε ότι η PORTD είναι πάντα έξοδος όταν θέλουμε να στείλουμε εντολές στην οθόνη, ενώ επίσης αποφεύγουμε να στείλουμε τα αντίστοιχα σήματα για τις εντολές της οθόνης κατά λάθος όταν διαχειριζόμαστε το θερμόμετρο.

Κώδικας σε C :

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.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
//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
//#define  temp r24

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

//unsigned char A,B,C,D,F1,F0;
unsigned char t1,t2;
//unsigned char x1,x2;
uint16_t x1,x2;

unsigned char  a1 = 0;
unsigned char  a2 = 0;
unsigned char  a3 = 0;
unsigned char  a4 = 0;


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;
}

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;
}
void write_2_nibbles(unsigned char q){
    int temp = q;   // r24 = temp = q
    int e = PIND;   //in r25, PIND
    e = e & 0x0f;    // andi r25, 0x0f
    temp = temp & 0xf0; // andi r24, 0xf0
    temp = temp + e;    // add r24, r25
    PORTD= temp;  //out PORTD ,r24
    PORTD |= 0x08;  //sbi PORTD ,3
    _delay_ms(1);
    PORTD &= 0b11110111; //cbi PORTD ,3
    _delay_ms(1);

    temp = q;   //pop r24
    temp = temp<<4;  //swap r24
    temp = temp & 0xf0;  //andi r24, 0xf0
    temp = temp + e;   //add r24,r25
    PORTD= temp;   //portd, r24
    PORTD |= 0x08;
    _delay_ms(1);
    PORTD &= 0b11110111;
    _delay_ms(2);

return;
}
```

```
void lcd_data(unsigned char p){
    PORTD |= 0x04;
    write_2_nibbles(p);
    _delay_ms(10);
    return;
}

void lcd_command(unsigned char z){
    PORTD &= 0b11111011;
    write_2_nibbles(z);
    _delay_ms(10);
    return;
}


void lcd_init(){

    _delay_ms(40); //wait_msec

        PORTD = 0x30;     //PORTD = r24 = 0x30
        PORTD |= 0x08;   // sbi PORTD, 3
        _delay_ms(1);
        PORTD &= 0b11110111; //cbi PORTD, 3
        _delay_ms(2);

         PORTD = 0x30;     //PORTD = r24 = 0x30
        PORTD |= 0x08;   // sbi PORTD, 3
        _delay_ms(1);
        PORTD &= 0b11110111; //cbi PORTD, 3
        _delay_ms(2);

        PORTD = 0x30;   //PORTD = r24 = 0x30
        PORTD |= 0x08;
        _delay_ms(1);
        PORTD &= 0b11110111;
        _delay_ms(2);

        PORTD = 0x20;
        PORTD |= 0x08;
        _delay_ms(1);
        PORTD &= 0b11110111;
        _delay_ms(2);
        lcd_command(0x28);
        lcd_command(0x0c);
        lcd_command(0x01);

        _delay_ms(10);

        lcd_command(0x06);

        _delay_ms(10);

        return;
}


unsigned char one_wire_reset(){

           unsigned char  temp;
           DDRD |= 0b00010000;
           PORTD &= 0b11101111;
           _delay_us(480);
           DDRD &= 0b11101111;
           PORTD &= 0b11101111;
           _delay_us(100);
           temp = PIND;                //temp = r24 = PORTD
           _delay_us(380);
           if(temp & 0b00010000) {         //if PD4 = 0 return 1 in temp = r24
                   temp = 0;
                   return temp;
           }
           else{                       //else return 0 in temp = r24
            temp = 1;
            return temp;
           }

}


unsigned char one_wire_receive_bit(){

           unsigned char temp;
           DDRD |= 0b00010000;
           PORTD &= 0b11101111;
           _delay_us(2);
           DDRD &= 0b11101111;
           PORTD &= 0b11101111;
           _delay_us(10);

           if(PIND & 0b00010000) {         //if PD4 = 1 return 0 in temp = r24 and delay(49)
                   temp = 1;

           }
           else{                       //else return 0 in temp = r24 and delay(49)
            temp = 0;
    }



    _delay_us(49);
    return temp;
}



void one_wire_transmit_bit(unsigned char x){

           unsigned char e;
           e = x;          // x will be r24 when the function is called
           DDRD |= 0b00010000;
           PORTD &= 0b11101111;
           _delay_us(2);

           if(e & 0x01){
                   PORTD |= 0b00010000;
           }
           else{
                   PORTD &= 0b11101111;
           }
           _delay_us(58);
           DDRD &= 0b11101111;
           PORTD &= 0b11101111;
           _delay_us(1);
```

```
}


unsigned char one_wire_receive_byte(){

    unsigned char temp;
            unsigned char temp2 = 0;       //temp2 = r26
            for(int i = 0; i < 8; i++){
                        temp = one_wire_receive_bit();
                        temp2 = temp2 >> 1;
                        if(temp & 1) temp = 0x80;   //if r24 == 1,r24 = 0x80
                        temp2 = temp2 | temp;                   //r26 or r24
            }
            temp = temp2;
            return temp;
}



void one_wire_transmit_byte(unsigned char x){

    unsigned char temp;
            unsigned char temp2 = x;        //r26 = x = r24
            for(int i = 0; i < 8; i++){
                        temp = 0;
                        if(temp2 & 0x01) temp = 0x01;
                        one_wire_transmit_bit(temp);
                        temp2 = temp2 >> 1;

            }
}

void get_temperature(){
    unsigned char  x;
    x = one_wire_reset();
    if(x == 0) {
        t1 = 0x00;
        t2 = 0x80;
     return;
    }
    one_wire_transmit_byte(0xCC);
    one_wire_transmit_byte(0x44);
    while(!(one_wire_receive_bit())) {
    }
    x = one_wire_reset();
    one_wire_transmit_byte(0xCC);
    one_wire_transmit_byte(0xBE);
    t1 = one_wire_receive_byte();
    t2 = one_wire_receive_byte();
//    t1 = 0xFF;          Test values for negative temperature.
//    t2 = 0xFF;

}






int main(){
    DDRD |= 0xFF;
    /////////////temp3 = r24 temp4 = r25
    int flag;



 while(1){
      get_temperature();
    if(t1 == 0x00 &&  t2 == 0x80){
        DDRD = 0xFF;
        lcd_init();
        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');



          _delay_ms(50);
    }
    else{
        DDRD = 0xFF;
    lcd_init();



    if(t2 & 0b10000000) {
        flag = 0;
    } // 0 = negative
    else{
        flag = 1;
    }



    a4 = 0;
    x1 = (uint16_t)t1;
    x2 = (uint16_t)t2;
    x2 = x2 << 8;
    x1 = x1 + x2;

    if(x1 & 0x08){
        a4 = 1;
    }

    if(flag){
        lcd_data('+');
    }
    else{
        lcd_data('-');
        x1 = x1 - 1;

        x1 = ~x1;

    }
```

```
    /*
    x1 &= 0b11110000;
    x1 = (x1 >> 4);
    x2 &= 0b00000111;
    x2 = (x2 << 4);
    */
    /*
    t1 = t1 >> 3;
    t2 = t2 << 5;
     */
    //x1 = x1 + x2;
    x1 = x1 >> 4;
    x1 = x1 & 0x00FF;

    a1 = 0;
    a2 = 0;
    a3 = 0;
        while(x1 >= 100){
            x1 = x1 -100;
            a1 += 1;
        }
        while(x1 >= 10){
            x1 = x1 - 10;
            a2 += 1;
        }
          while(x1 >= 1){
            x1 = x1 - 1;
            a3 += 1;
        }



        a1 |= 0b00110000;
        a2 |= 0b00110000;
        a3 |= 0b00110000;
        lcd_data(a1);
        lcd_data(a2);
        lcd_data(a3);
        if(a4 == 1){
            lcd_data(',');
            lcd_data('5');
        }
        lcd_data(' ');
        //lcd_data(0b01100000);
        //lcd_data(0b01000011);
        lcd_data('C');
        lcd_data('e');
        lcd_data('l');
        lcd_data('s');
        lcd_data('i');
        lcd_data('u');
        lcd_data('s');





        _delay_ms(500);
      }
 }

}
```