**Федеральное государственное автономное образовательное учреждение высшего образования**

**"Национальный исследовательский университет**

**"Высшая школа экономики"**

Московский институт электроники и математики им. Тихонова

Департамент компьютерной инженерии

**Дисциплина: Микроконтроллерные системы**

**ОТЧЕТ**

О выполнении домашнего задания

«Реализация восьмеричного калькулятора на PIC18F4520»

                         Студенты: Опекунова Алина

                                                        Группа: БИВ193

Преподаватель: Тув Александр Леонидович

Москва 2022

Оглавление

[Задание 3](#_Toc106116326)

[Алгоритм работы программы 4](#_Toc106116327)

[Программа 5](#_Toc106116328)

[1. Подключение LCD дисплея 5](#_Toc106116329)

[2. Подключение USART 8](#_Toc106116330)

[3. Работа с мини-клавиатурой 9](#_Toc106116331)

[4. Ввод данных 12](#_Toc106116332)

[5. Выполнение математических операций 15](#_Toc106116333)

[Функция main() 17](#_Toc106116334)

[Пример работы 18](#_Toc106116335)

[Приложение 1. Текст программы 20](#_Toc106116336)

Задание

Разработать и отладить ПО «Восьмеричный калькулятор»

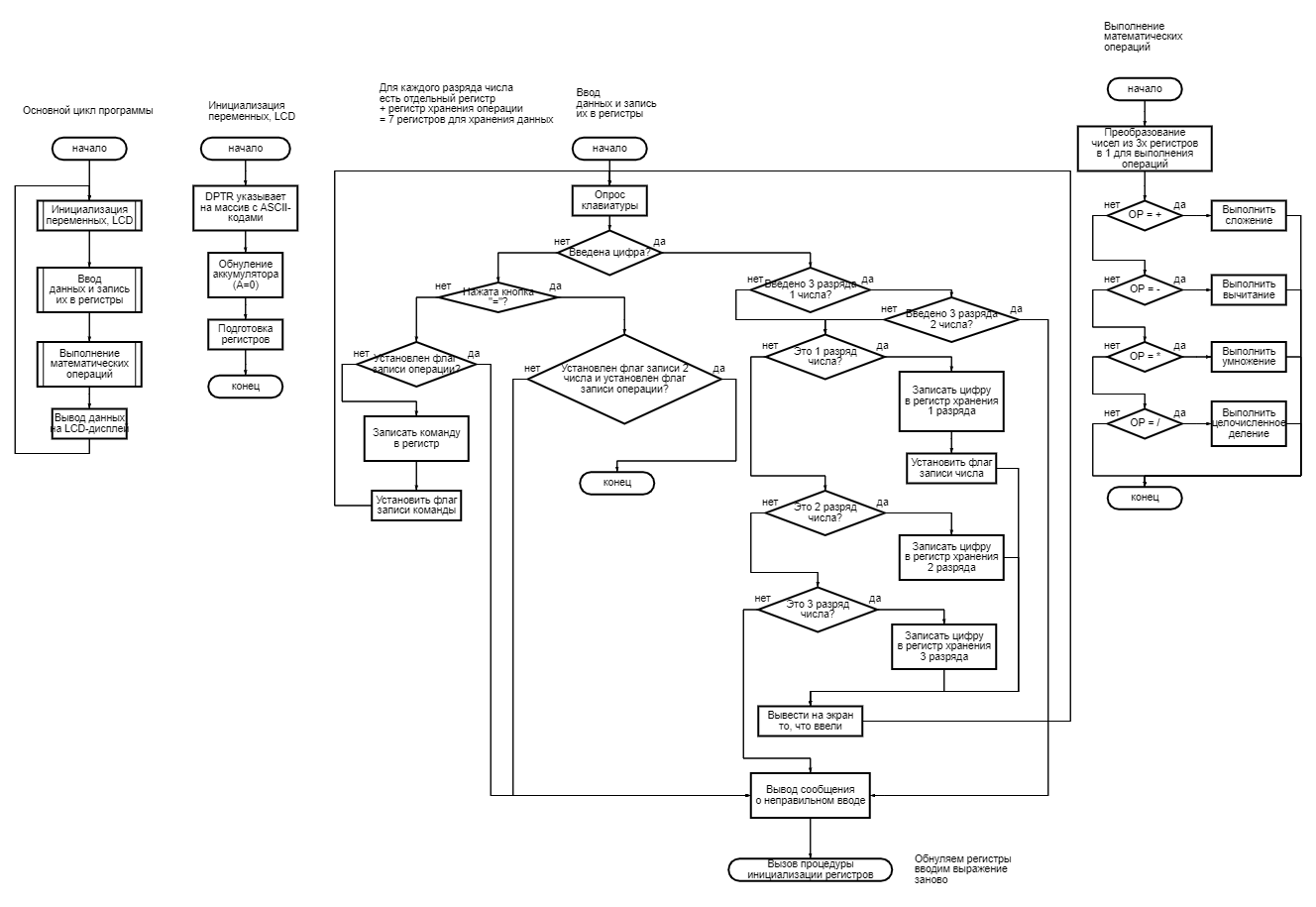
Ввод:

* Мини-клавиатура - до 3 символов 0..7, +,-,\*,/,=
* Клавиатура ПК - до 3 символов 0..7, +,-,\*,/,=

Вывод:

* результат в восьмеричном коде.
* все вводимые символы должны отображаться на строчном LCD-дисплее.

Деление – целочисленное.

Алгоритм работы программы

Программа

1. Подключение LCD дисплея
   1. Начальная установка модуля

После подачи питания необходимо провести инициализацию дисплея для корректного функционирования. Порядок инструкций для инициализации представлен на рисунке 1:

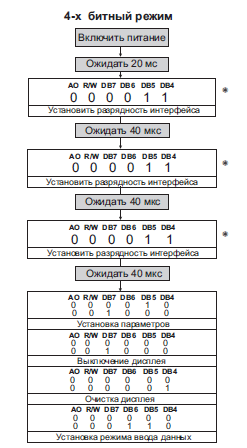


Рис. 1. Инициализация LCD-дисплея для 4-битного режима

Реализация этих команд показана на рис. 2. Эту функцию необходимо вызывать первой при работе с ЖК дисплеем. Сначала устанавливаем частоту на 8 МГц, далее устанавливаем выводы E и RS порта А как выходные. Далее следует несколько команд перехода в 4-битный режим, установка параметров, выключение дисплея и его очистка, последняя команда настройки дисплея является установка ввода данных.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 2. Инициализация LCD дисплея

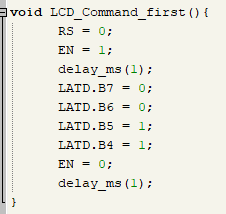


Рис. 3. Установка разрядности интерфейса

На рисунке 3 показана функция, выполняющая установку разрядности интерфейса (переход в 4-битный режим).

Изображение выглядит как стол

Автоматически созданное описание

Рис. 4. Передача данных в PORTD

На рис. 4. показана функция записи информации в порты D4-D7 LCD. В ней сначала передаются старшие 4 бита, далее после программной задержки – младшие 4 бита.

Так как запись команд и данных имеет одинаковый алгоритм, но при этом регистр RS должен иметь разные значения (1- данные , 0 - команды), создадим 2 функции, которые будут отвечать за передачу команд на модуль дисплея (рис. 5) и вывод символа (рис. 6) на модуль дисплея.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 5. Передача команды на модуль дисплея

Изображение выглядит как текст

Автоматически созданное описание

Рис. 6. Вывод символа на дисплей

1. Подключение USART

PIC18F4520 имеет встроенный модуль USART, который полезен для последовательной связи. С помощью USART мы можем отправлять/получать данные на компьютер или другие устройства.

Рассчитаем скорость передачи данных:

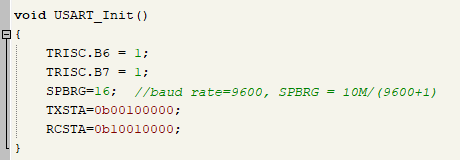


Рис. 7. Инициализация последовательного порта

На рисунке 7 показана функция инициализации последовательного порта. Сначала в ней установим 6 и 7 биты PORTC как входные. Инициализируем скорость передачи данных, загрузив значение в регистр SPBRG. Установим бит TXEN в регистре TXSTA для включения передачи и бит CREN в регистре RCSTA для включения приема.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 8. Режим передачи данных

На рисунке 8 показана функция, принимающая на вход данные и позволяющая выводить их в последовательный порт через регистр TXREG, в случае равенства флага прерываний TXIF = 1.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 9. Чтение данных из последовательного порта

На рисунке 9 функция, которая передает данные из последовательного порта. RCSTA — регистр управления и статуса приемника. Бит 1 (OERR) — ошибка переполнения внутреннего буфера (0 — нет переполнения, 1 — есть переполнение). Чтобы сбросить эту ошибку — нужно выключить и снова включить приемник. (сбросить и заново установить бит CREN).

Полученное в регистре приема данных RCREG и возвращается в основную программу

1. Работа с мини-клавиатурой

Изображение выглядит как текст

Автоматически созданное описание

Рис. 10. Структура для хранения состояния кнопок клавиатуры

Для хранения состояния клавиш клавиатуры создана структура, изображенная на рис. 10.

Строки клавиатуры подключены к выводам PORTB.4, PORTB.5, PORTB.6, PORTB.7. Они настроены на выход и в начальном состоянии на этих выводах напряжение логической единицы. Столбцы подключены к выводам PORTB.0, PORTB.1, PORTB.2, PORTB.3 и настроены на вход.

Процедура сканирования клавиатуры выглядит следующим образом. Выставляем 1 на выводе PORTB.4 и проверяем состояние выводов PORTB.0, PORTB.1, PORTB.2, PORTB.3. Если на каком-то из выводов установлен 0, значит, на клавиатуре в данный момент нажата кнопка, подключенная к первой строке. Сохраняем биты порта B в одном из полей структуры key\_state – по этому коды можно определить символ нажатой кнопки. Если ни одна из кнопок не нажата, продолжаем процедуру сканирования. Устанавливаем 1 на выводе PORTB.4 и устанавливаем 0 на следующем выводе. Повторяем описанную последовательность для двух оставшихся строк. Описанная выше процедура показана на рис. 11, а расшифровка нажатия на рис. 12.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 11. Функция опроса клавиатуры

Изображение выглядит как стол

Автоматически созданное описание

Рис. 12. Расшифровка нажатых клавиш

1. Ввод данных

Изображение выглядит как текст

Автоматически созданное описание

Рис. 13. Структура для отслеживания состояния программы

Для отслеживания состояния работы программы создан перечислимый тип State\_program (рис. 13), который содержит 4 вида состояния: WAIT\_FIRST\_DIGIT – ожидание ввода 1 числа, WAIT\_OPERATION – ожидание ввода операции, WAIT\_SECOND\_DIGIT – ожидание ввода 1 числа, DO\_OPERATION – выполнение операции.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 14. Структура для хранения операнд

Для хранения чисел создана структура, которая хранит 3 разряда числа и число введенных разрядов (рис. 14).

Все операции в программе выполняются в вечном цикле после всех необходимых инициализаций. Далее происходит чтение данных с клавиатуры или последовательного порта. Считанный символ подается на вход функции decodPressedKey, где в зависимости от состояния программы он записывается в поле структуры или переменную хранения операции, либо при неправильном вводе происходит очищение LCD-дисплея и ранее введенных данных с выводом сообщения об ошибке.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 15. Состояние программы ввода первого числа

На рисунке 15 показан код, позволяющий поразрядно записывать вводимое число. Здесь мы проверяем полученный символ, и если введена цифра и счетчик разрядов не равен 3, то записываем этот символ, иначе проверяем, был ли введен хотя бы один разряд, в противном случае выводим сообщение об ошибке. Также если мы ввели 3 разряда, то программа автоматически переходит в режим ожидания ввода операции. Для ввода второго операнда используем похожий алгоритм (рис. 16).

Изображение выглядит как текст

Автоматически созданное описание

Рис. 16. Состояние программы ввода второго числа

Изображение выглядит как текст

Автоматически созданное описание

Рис. 17. Состояние ввода операции

На рисунке 17 показан код, позволяющий запомнить введенную операцию.

1. Выполнение математических операций

Все математические операции выполняются в десятичной системе счисления, поэтому реализованы функции для перевода из 8-ричной СС в 10-тичную и наоборот (рис. 18, рис. 19).

Изображение выглядит как текст

Автоматически созданное описание

Рис. 18. Функция перевода из 8-ричной СС в 10-тичную

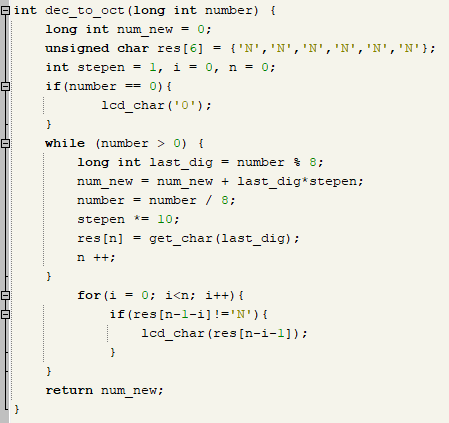


Рис. 19. Функция перевода из 10-тичной СС в 8-ричную

При переходе программы в состояние DO\_OPERATION срабатывает функция подсчета результата executeOperation (рис. 20), где происходит преобразование трех разрядов операнд в одно число типа long int, далее они переводятся в десятичную систему счисления и в зависимости от знака операции производится подсчет результата, обратный перевод его в восьмеричную СС и вывод на экран LCD-дисплея.



Рис. 20. Функция подсчета ответа

Функция main()

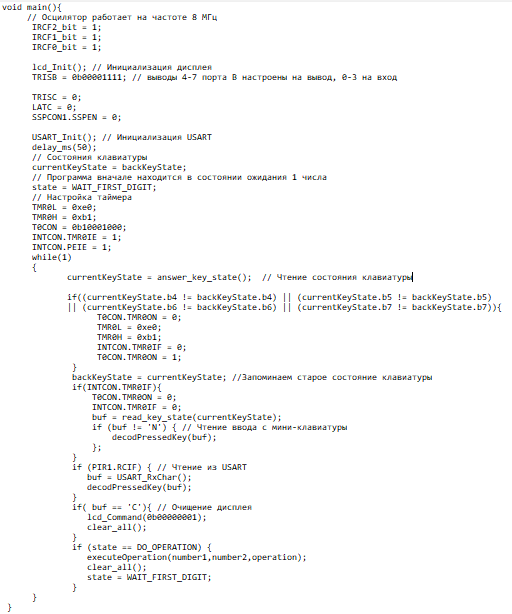


Рис. 21. Основная функция

Пример работы

1. Умножение

Изображение выглядит как текст, электроника, цепь

Автоматически созданное описание

1. Деление

Изображение выглядит как текст, электроника, цепь

Автоматически созданное описание

1. Сложение

Изображение выглядит как текст, электроника, цепь

Автоматически созданное описание

1. Вычитание

Изображение выглядит как текст, электроника, цепь

Автоматически созданное описание

1. Неправильное выражение – вывод сообщения об ошибке

Изображение выглядит как текст, электроника, цепь

Автоматически созданное описание

**Приложение 1. Текст программы**

#include <stdio.h>

#include <string.h>

#define RS LATA.B4

#define EN LATA.B5

#define LCD\_Port TRISD

#define D4 LATD.B4

#define D5 LATD.B5

#define D6 LATD.B6

#define D7 LATD.B7

#define true 1

#define false 0

typedef union {

unsigned char byte;

struct {

unsigned char bit0: 1;

unsigned char bit1: 1;

unsigned char bit2: 1;

unsigned char bit3: 1;

unsigned char bit4: 1;

unsigned char bit5: 1;

unsigned char bit6: 1;

unsigned char bit7: 1;

};

struct {

unsigned char lowNibble: 4;

unsigned char highNibble: 4;

};

} MyByte;

typedef union {

long int byte;

struct {

long int byte0;

long int byte1;

long int byte2;

long int counter;

};

} termStruct;

enum State\_program{

WAIT\_FIRST\_DIGIT,

WAIT\_OPERATION,

WAIT\_SECOND\_DIGIT,

DO\_OPERATION} state;

typedef union {

unsigned char byte;

struct {

unsigned char b4: 0b11101111;

unsigned char b5: 0b11011111;

unsigned char b6: 0b10111111;

unsigned char b7: 0b01111111;

};

} key\_state;

char operation;

char buf = 'N';

termStruct number1;

termStruct number2;

key\_state currentKeyState;

key\_state backKeyState;

key\_state answer\_key\_state(){

key\_state cur\_state;

LATB.B5 = 1;

LATB.B6 = 1;

LATB.B7 = 1;

LATB.B4 = 0;

nop();

nop();

cur\_state.b4 = PORTB;

LATB.B4 = 1;

LATB.B5 = 0;

nop();

nop();

cur\_state.b5 = PORTB;

LATB.B5 = 1;

LATB.B6 = 0;

nop();

nop();

cur\_state.b6 = PORTB;

LATB.B6 = 1;

LATB.B7 = 0;

nop();

nop();

cur\_state.b7 = PORTB;

LATB.B7 = 1;

return cur\_state;

}

unsigned char read\_key\_state(key\_state cur\_state){

unsigned char def = 'N';

unsigned char num = cur\_state.b4;

switch(num){

case 0b11100111:

def = 0x2b;

break;

case 0b11101011:

def = 0x33;

break;

case 0b11101101:

def = 0x32;

break;

case 0b11101110:

def = 0x31;

break;

}

num = cur\_state.b5;

switch(num){

case 0b11010111:

def = '-';

break;

case 0b11011011:

def = 0x36;

break;

case 0b11011101:

def = 0x35;

break;

case 0b11011110:

def = 0x34;

break;

}

num = cur\_state.b6;

switch(num){

case 0b10110111:

def = 0x2a;

break;

case 0b10111011:

def = 0x39;

break;

case 0b10111101:

def = 0x38;

break;

case 0b10111110:

def = 0x37;

break;

}

num = cur\_state.b7;

switch(num){

case 0b01110111:

def = 0x2f;

break;

case 0b01111011:

def = 0x3d;

break;

case 0b01111101:

def = 0x30;

break;

case 0b01111110:

def = 0x43;

break;

}

return def;

}

void lcd\_print(unsigned char dat){

MyByte a;

a.byte = dat;

EN = 1;

nop();

LATD.B7 = a.bit7;

LATD.B6 = a.bit6;

LATD.B5 = a.bit5;

LATD.B4 = a.bit4;

nop();

EN = 0;

nop();

EN = 1;

nop();

LATD.B7 = a.bit3;

LATD.B6 = a.bit2;

LATD.B5 = a.bit1;

LATD.B4 = a.bit0;

nop();

EN = 0;

nop();

}

void lcd\_char(unsigned char dat){

RS = 1;

delay\_ms(1);

lcd\_print(dat);

}

void lcd\_Command(unsigned char cmd){

RS = 0;

delay\_ms(1);

lcd\_print(cmd);

}

void lcd\_string(char \*str)

{ unsigned char i = 0;

while(str[i]!='\0')

{

lcd\_char(str[i]);

i++;

}

}

void LCD\_Command\_first(){

RS = 0;

EN = 1;

delay\_ms(1);

LATD.B7 = 0;

LATD.B6 = 0;

LATD.B5 = 1;

LATD.B4 = 1;

EN = 0;

delay\_ms(1);

}

void lcd\_Init(){

ADCON1 = 0b0001111;

// E и RS на вывод

TRISA.B5 = 0;

TRISA.B4 = 0;

TRISD = 0;

delay\_ms(20);

LCD\_Command\_first();

delay\_us(40);

LCD\_Command\_first();

delay\_us(40);

LCD\_Command\_first();

lcd\_Command(0b00101000);

delay\_ms(2);

lcd\_Command(0b00001111);

delay\_ms(2);

lcd\_Command(0b00000001);

delay\_ms(2);

lcd\_Command(0b00000110);

}

int get\_num(char ch)

{

int num = 0;

switch(ch)

{

case '0':

num = 0;

break;

case '1':

num = 1;

break;

case '2':

num = 2;

break;

case '3':

num = 3;

break;

case '4':

num = 4;

break;

case '5':

num = 5;

break;

case '6':

num = 6;

break;

case '7':

num = 7;

break;

case '8':

num = 8;

break;

case '9':

num = 9;

break;

case 'A':

num = 0xA;

break;

case 'B':

num = 0xb;

break;

case 'C':

num = 0xc;

break;

case 'D':

num = 0xd;

break;

case 'E':

num = 0xe;

break;

case 'F':

num = 0xf;

break;

}

return num;

}

char get\_char(int ch)

{

char num = 'N';

switch(ch)

{

case 0:

num = '0';

break;

case 1:

num = '1';

break;

case 2:

num = '2';

break;

case 3:

num = '3';

break;

case 4:

num = '4';

break;

case 5:

num = '5';

break;

case 6:

num = '6';

break;

case 7:

num = '7';

break;

case 8:

num = '8';

break;

case 9:

num = '9';

break;

case 0xa:

num = 'A';

break;

case 0xb:

num = 'B';

break;

case 0xc:

num = 'C';

break;

case 0xd:

num = 'D';

break;

case 0xe:

num = 'E';

break;

case 0xf:

num = 'F';

break;

}

return num;

}

void print\_num(long int number);

int oct\_to\_dec(long int number) {

unsigned long int num\_new = 0;

unsigned long int last\_dig = 0;

int stepen = 1;

while (number > 0) {

last\_dig = (number % 10) \* stepen;

num\_new = num\_new + last\_dig;

number /= 10;

stepen \*= 8;

}

return num\_new;

}

int dec\_to\_oct(long int number) {

long int num\_new = 0;

unsigned char res[6] = {'N','N','N','N','N','N'};

int stepen = 1, i = 0, n = 0;

if(number == 0){

lcd\_char('0');

}

while (number > 0) {

long int last\_dig = number % 8;

num\_new = num\_new + last\_dig\*stepen;

number = number / 8;

stepen \*= 10;

res[n] = get\_char(last\_dig);

n ++;

}

for(i = 0; i<n; i++){

if(res[n-1-i]!='N'){

lcd\_char(res[n-i-1]);

}

}

return num\_new;

}

void print\_num(long int number){

unsigned int i = 0, n = 0;

char result[6] = {'N','N','N','N','N','N'};

if(number == 0){

lcd\_char('0');

}

while(number > 0){

result[n] = get\_char(number%10);

number = number / 10;

n++;

};

for(i = 0; i<n; i++){

if(result[n-1-i]!='N'){

lcd\_char(result[n-i-1]);

}

}

}

void USART\_Init()

{

TRISC.B6 = 1;

TRISC.B7 = 1;

SPBRG=16; //baud rate=9600, SPBRG = 10M/64\*(9600+1)

TXSTA=0b00100000;

RCSTA=0b10010000;

}

unsigned char USART\_TxChar(char data\_in)

{

if(PIR1.TXIF==1) {

TXREG = data\_in;

return true;

};

return false;

}

char USART\_RxChar()

{

if(RCSTA.OERR)

{

LATC.B5 = 0;//CREN = 0;

nop();

LATC.B5 = 1;//CREN=1;

}

return(RCREG);

}

void clear\_all(){

number1.byte0 = 0;

number1.byte1 = 0;

number1.byte2 = 0;

number1.counter = 0;

number2.byte0 = 0;

number2.byte1 = 0;

number2.byte2 = 0;

number2.counter = 0;

}

void decodPressedKey(unsigned char key) {

char data\_in;

switch (state) {

case WAIT\_FIRST\_DIGIT:

{

lcd\_char(key);

if((number1.counter<3)&&(key!='+' && key!='-' && key!='\*' && key!='/' && key!='8' && key!='9' && key!='=')){

data\_in = get\_num(key);

if(number1.counter == 2)

{

number1.byte0 = data\_in;

number1.counter ++;

state = WAIT\_OPERATION;

}

if(number1.counter == 1)

{

number1.byte1 = data\_in;

number1.counter ++;

}

if(number1.counter == 0)

{

number1.byte2 = data\_in;

number1.counter ++;

}

}

else{

if((key=='+' || key=='-' || key=='\*' || key=='/')&&number1.counter>0){

operation = key;

state = WAIT\_SECOND\_DIGIT;

}

else{

lcd\_string("Wrong Input");

nop();

nop();

lcd\_Command(0b00000001);

state = WAIT\_FIRST\_DIGIT;

clear\_all();

}

}

break;

}

case WAIT\_SECOND\_DIGIT:

{

lcd\_char(key);

if((number2.counter<3)&&(key!='+' && key!='-' && key!='\*' && key!='/' && key!='8' && key!='9' && key!='=')){

data\_in = get\_num(key);

if(number2.counter == 2)

{

number2.byte0 = data\_in;

number2.counter ++;

state = DO\_OPERATION;

lcd\_char('=');

}

if(number2.counter == 1)

{

number2.byte1 = data\_in;

number2.counter ++;

}

if(number2.counter == 0)

{

number2.byte2 = data\_in;

number2.counter ++;

}

}

else{

if(key=='=' && number2.counter>0){

state = DO\_OPERATION;

}

else{

lcd\_string("Wrong Input");

nop();

nop();

lcd\_Command(0b00000001);

state = WAIT\_FIRST\_DIGIT;

clear\_all();

}

}

break;

}

case WAIT\_OPERATION:

{

lcd\_char(key);

if(key=='+' || key=='-' || key=='\*' || key=='/'){

operation = key;

state = WAIT\_SECOND\_DIGIT;

}

else{

lcd\_string("Wrong Input");

nop();

nop();

lcd\_Command(0b00000001);

state = WAIT\_FIRST\_DIGIT;

clear\_all();

}

break;

}

}

}

void executeOperation(termStruct key1, termStruct key2, unsigned char op){

long int num1\_oct, num2\_oct, num1\_dec, num2\_dec, res\_oct = 0, res\_dec = 0, t =0;

if(key1.counter == 1){

num1\_oct = key1.byte2;

}

else if(key1.counter == 2){

num1\_oct = key1.byte2\*10+key1.byte1;

}

else {

num1\_oct = key1.byte2\*100+key1.byte1\*10+key1.byte0;

}

if(key2.counter == 1){

num2\_oct = key2.byte2;

}

else if(key2.counter == 2){

num2\_oct = key2.byte2\*10+key2.byte1;

}

else{

num2\_oct = key2.byte2\*100+key2.byte1\*10+key2.byte0;

}

if((num2\_oct == 0)&&(op=='/')){

lcd\_string("Error");

nop();

nop();

lcd\_Command(0b00000001);

state = WAIT\_FIRST\_DIGIT;

clear\_all();

}

else{

num1\_dec = oct\_to\_dec(num1\_oct);

num2\_dec = oct\_to\_dec(num2\_oct);

switch(op)

{

case '+':

res\_dec = num1\_dec+num2\_dec;

break;

case '-':

res\_dec = num1\_dec-num2\_dec;

break;

case '\*':

res\_dec = num1\_dec\*num2\_dec;

break;

case '/':

res\_dec = num1\_dec/num2\_dec;

break;

}

if(res\_dec<0){

lcd\_char('-');

t = -res\_dec;

res\_dec=t;

}

res\_oct = dec\_to\_oct(res\_dec);

}

}

int counter\_ = 0;

int counter\_2 = 0;

void main(){

IRCF2\_bit = 1;

IRCF1\_bit = 1;

IRCF0\_bit = 1;

lcd\_Init();

TRISB = 0b00001111;

TRISC = 0;

LATC = 0;

SSPCON1.SSPEN = 0;

USART\_Init();

delay\_ms(50);

currentKeyState = backKeyState;

state = WAIT\_FIRST\_DIGIT;

TMR0L = 0xe0;

TMR0H = 0xb1;

T0CON = 0b10001000;

INTCON.TMR0IE = 1;

INTCON.PEIE = 1;

while(1)

{

currentKeyState = answer\_key\_state();

if((currentKeyState.b4 != backKeyState.b4) || (currentKeyState.b5 != backKeyState.b5) || (currentKeyState.b6 != backKeyState.b6) || (currentKeyState.b7 != backKeyState.b7)){

T0CON.TMR0ON = 0;

TMR0L = 0xe0;

TMR0H = 0xb1;

INTCON.TMR0IF = 0;

T0CON.TMR0ON = 1;

}

backKeyState = currentKeyState;

if(INTCON.TMR0IF){

T0CON.TMR0ON = 0;

INTCON.TMR0IF = 0;

buf = read\_key\_state(currentKeyState);

if (buf != 'N') {

decodPressedKey(buf);

};

}

if (PIR1.RCIF) {

buf = USART\_RxChar();

decodPressedKey(buf);

}

if( buf == 'C'){

lcd\_Command(0b00000001);

clear\_all();

}

if (state == DO\_OPERATION) {

executeOperation(number1,number2,operation);

clear\_all();

state = WAIT\_FIRST\_DIGIT;

}

}

}