太原理工大学
TAIYUAN UNIVERSITY OF TECHNOLOGY

本科实验报告

课程名称：嵌入式系统课程设计

实验项目：用8051+1601LCD设计的整型计算器

实验地点：计算机实验楼 110

专业班级：物联网 1501　学号：2015001965

学生姓名：高磊

指导教师：马建芬

2018 年　7 月　6　日

用 8051+1601LCD 设计的整型计算器

1. 问题描述

用单行字符液晶、键盘矩阵及 8051 单片机设计一部简易计算器，该计算器可进行四则运算的单次或连续整型数据运算，并且支持优先级的表达式求值。

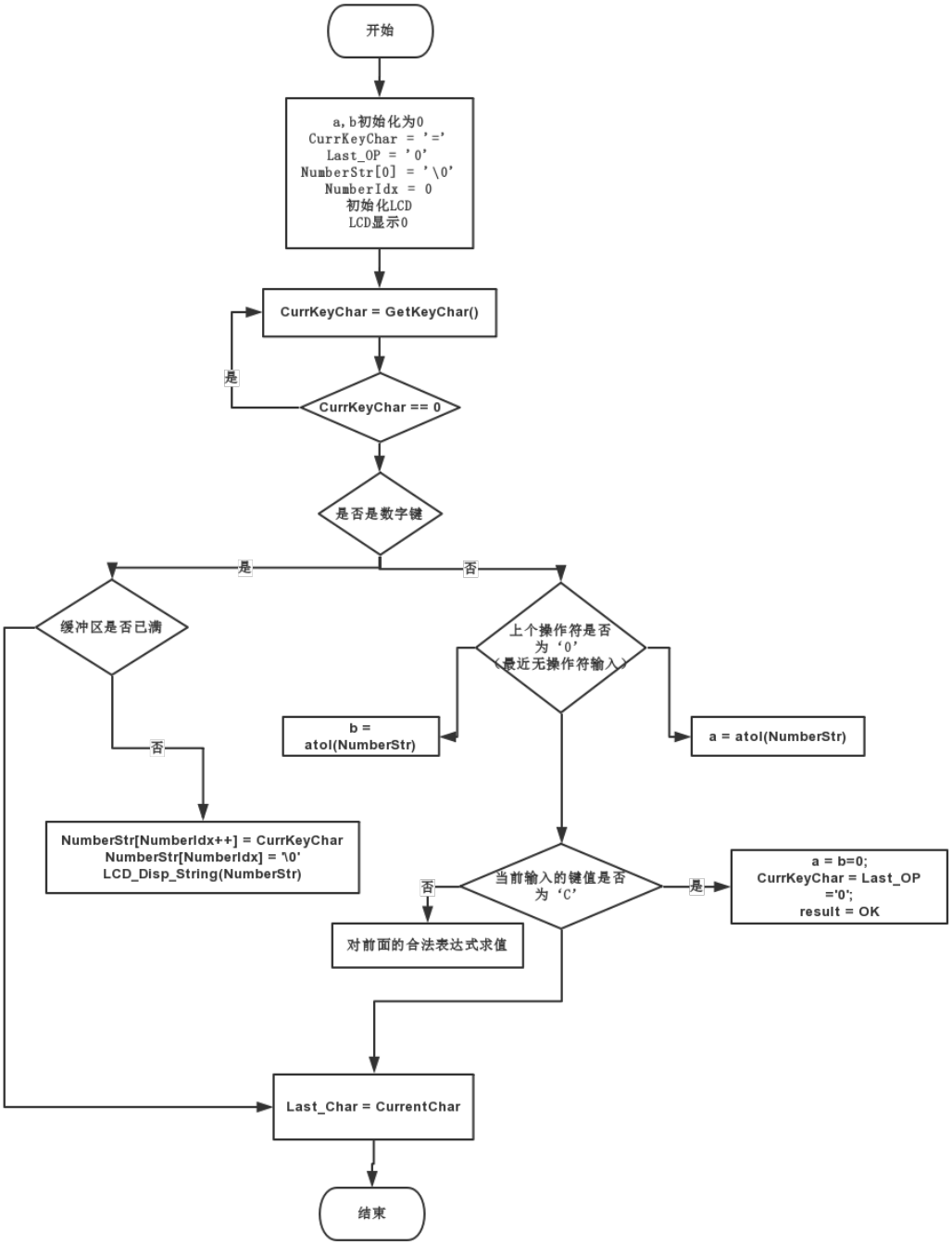
2. 设计需求与分析

1. LCD、RAM 接口地址分配：通过拓展接口方式控制 LCD1602 液晶及拓展 RAM。本电路中二者共用总线。

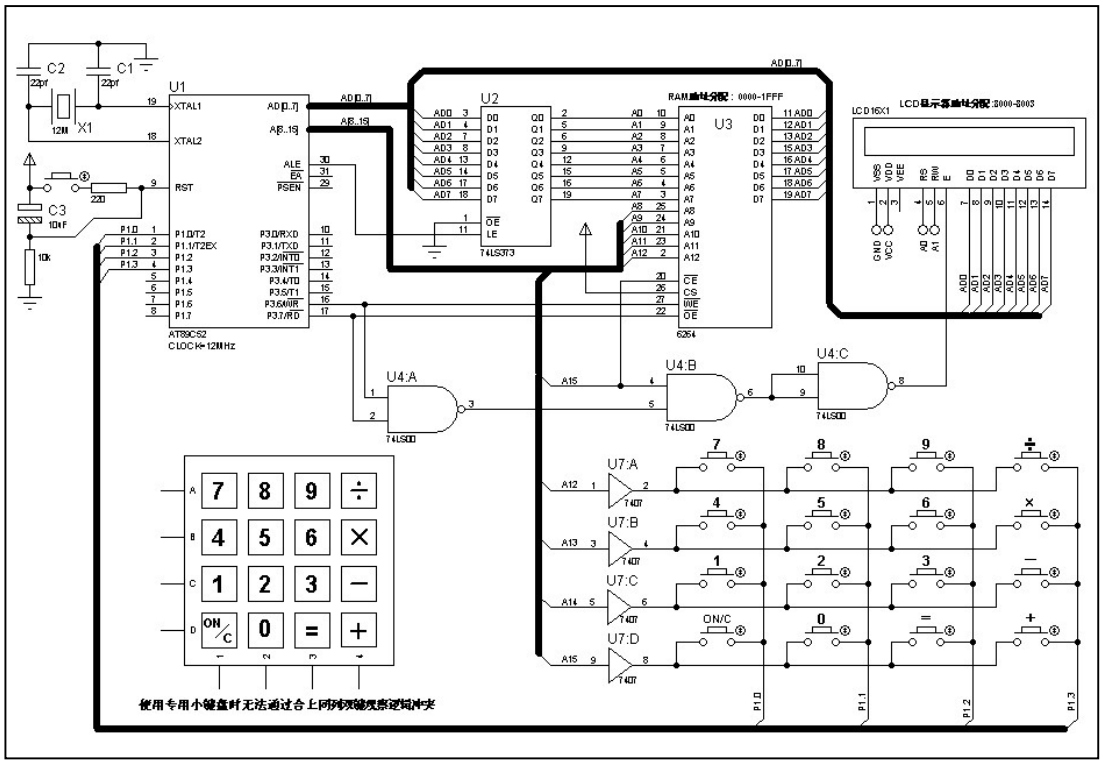
2. 矩阵键盘电路及扫描程序设计：P2 端口高 4 位通过非反响 OC 驱动器 7407 连接矩阵键盘行线，列线连接 P1 端口低四位。键盘扫描时从 0xEF 开始，通过循环移位分别在 P2.4~P2.7 上输出扫描码 1110、1101、1011、0111。并检查 P1 端口读取值的低 4 位中，在 P1.0~P1.3 是否有一位为 0，如果出现则表示找到了按键所在的行、列位置(r,c)，如果此位置在上次搜索后其状态未按下，则返回其键值，否则返回 0。

3. 计算器程序设计：该程序由 main.c、lcd.c、keypad.c、calc.h 四个文件构成。设计要点在于单行液晶以右端为起点的显示设计，数据输入及运算程序设计、矩阵键盘电路及扫描程序设计。

详细流程图如下所示：


开始
a,b初始化为0
CurrKeyChar = '='
Last_OP = '0'
NumberStr[0] = '\0'
NumberIdx = 0
初始化LCD
LCD显示0
CurrKeyChar = GetKeyChar()
是
CurrKeyChar == 0
是否是数字键
是
否
缓冲区是否已满
否
NumberStr[NumberIdx++] = CurrKeyChar
NumberStr[NumberIdx] = '\0'
LCD_Disp_String(NumberStr)
上个操作符是否为'0'(最近无操作符输入)
b = atoi(NumberStr)
a = atoi(NumberStr)
当前输入的键值是否为'C'
否
对前面的合法表达式求值
是
a = b=0;
CurrKeyChar = Last_OP ='0';
result = OK
Last_Char = CurrentChar
结束

3. 设计功能的实现



电路图

程序：

```
//main.c
#include <reg51.h>
#include <intrins.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "calc.h"
#define INT8U    unsigned char
#define INT16U unsigned int
static long a,b;              //当前运算符的前后两个操作数
static char CurrKeyChar; //当前按键字符
static char Last_OP;       //最近输入的操作符
static char Last_Char;          //所输入的前一字符
static char result;             //当前运算的结果状态
//显示缓冲,数字输入缓冲及数字输入缓冲区索引定义
static char xdata outputbuffer[MAX_DISPLAY_CHAR+1];
static char xdata NumberStr[MAX_DISPLAY_CHAR+1];
```

```
static char xdata NumberIdx;
char numbers[MAX_DISPLAY_CHAR] = {0};
int numsTop = 0;
char flags[MAX_DISPLAY_CHAR] = {0};
int flagsTop = 0;
char mid;
char flagCompare(char a,char b);//如果 a 运算符优先于 b，则返回 1 优先级相等返回 0 a 优先级小于 b 返回-1
int i;
int j;
//-----------------------------------------------------------------
// 主程序
//-----------------------------------------------------------------
void main()
{
    a = 0; b = 0;//两个操作数初始化为 0
    CurrKeyChar = '=' ; Last_OP = '0';//初始化当前按键及最近输入的操作符
    NumberStr[0] = '\0';NumberIdx = 0;//消除输入缓冲，缓冲索引归 0

    Initialise_LCD();//初始化 LCD
    LCD_Disp_String("0");//初始化显示 0
    while(1)//循环扫描键盘并进行运算处理与显示
    {
        //调用矩阵键盘扫描程序，有键按下时返回按键字符，无键按下时循环扫描
        do{CurrKeyChar = GetKeyChar();}while(!CurrKeyChar);

        if(CurrKeyChar!='=' && CurrKeyChar != 'C')
        {
            if(NumberIdx<MAX_DISPLAY_CHAR)
            {
                NumberStr[NumberIdx++]=CurrKeyChar;
                NumberStr[NumberIdx] = 0;
                LCD_Disp_String(NumberStr);
            }
            else
                LCD_Disp_String("Too Long! ");
        }
        else if(CurrKeyChar == 'C'){
            a = 0; b = 0;//两个操作数初始化为 0
            CurrKeyChar = '=' ; Last_OP = '0';//初始化当前按键及最近输入的操作符
            NumberStr[0] = '\0';NumberIdx = 0;//消除输入缓冲，缓冲索引归 0
            numbers[0] = 0;
            numsTop = 0;
            flags[0] = 0;
```

```
                flagsTop = 0;mid = 0;
                i=0;
                LCD_Disp_String(" ");
            }
            else{

                for (i = 0; i < NumberIdx; i++) {
                    if (isdigit(NumberStr[i]))
                    {
                        numbers[numsTop++] = NumberStr[i] - '0';
                        numbers[numsTop] = 0;
                    }
                    else {
                        if (flagsTop == 0 || (flagsTop != 0 && (flagCompare(NumberStr[i],
flags[flagsTop - 1]) == 1)))
                        {
                            flags[flagsTop++] = NumberStr[i];
                            flags[flagsTop] = 0;
                        }
                        else {
                            while (flagsTop!= 0 && (flagCompare(flags[flagsTop - 1],
NumberStr[i]) >= 0))
                            {
                                switch (flags[flagsTop-1])
                                {
                                case '+':
                                    mid = numbers[numsTop - 1] + numbers[numsTop - 2];
                                    numbers[numsTop - 1] = numbers[numsTop - 2] = 0;
                                    numsTop -= 1;
                                    numbers[numsTop - 1] = mid;
                                    flags[--flagsTop] = 0;
                                    break;
                                case '-':
                                    mid = numbers[numsTop - 2] - numbers[numsTop - 1];
                                    numbers[numsTop - 1] = numbers[numsTop - 2] = 0;
                                    numsTop -= 1;
                                    numbers[numsTop - 1] = mid;
                                    flags[--flagsTop] = 0;
                                    break;
                                case '*':
                                    mid = numbers[numsTop - 1] * numbers[numsTop - 2];
                                    numbers[numsTop - 1] = numbers[numsTop - 2] = 0;
                                    numsTop -= 1;
                                    numbers[numsTop - 1] = mid;
```

```
                                flags[--flagsTop] = 0;
                                break;
                            case '/':
                                mid = numbers[numsTop - 2] / numbers[numsTop - 1];
                                numbers[numsTop - 1] = numbers[numsTop - 2] = 0;
                                numsTop -= 1;
                                numbers[numsTop - 1] = mid;
                                flags[--flagsTop] = 0;
                                break;
                            }
                        }
                        flags[flagsTop++] = NumberStr[i];

                    }
                }
            }

            for (j = flagsTop - 1; j >= 0; j--)
            {
                switch (flags[j])
                {
                case '+':
                    mid = numbers[numsTop - 1] + numbers[numsTop - 2];
                    numbers[numsTop - 1] = numbers[numsTop - 2] = 0;
                    numsTop -= 1;
                    numbers[numsTop - 1] = mid;
                    break;
                case '-':
                    mid = numbers[numsTop - 2] - numbers[numsTop - 1];
                    numbers[numsTop - 1] = numbers[numsTop - 2] = 0;
                    numsTop -= 1;
                    numbers[numsTop - 1] = mid;
                    break;
                case '*':
                    mid = numbers[numsTop - 1] * numbers[numsTop - 2];
                    numbers[numsTop - 1] = numbers[numsTop - 2] = 0;
                    numsTop -= 1;
                    numbers[numsTop - 1] = mid;
                    break;
                case '/':
                    mid = numbers[numsTop - 2] / numbers[numsTop - 1];
                    numbers[numsTop - 1] = numbers[numsTop - 2] = 0;
                    numsTop -= 1;
                    numbers[numsTop - 1] = mid;
```

```
                        break;
                    }
                }
                mid = numbers[0];
                sprintf(numbers,"%d",(int)mid);
                LCD_Disp_String(numbers);
            }
        }
}

char flagCompare(char a,char b)
{
    switch(a)
    {
        case '*':
            if(b=='+' || b=='-') return 1;
            else return 0;
        case '/':
            if(b=='+' || b=='-') return 1;
            else return 0;
        default:
            if(b == '*' || b=='/') return -1;
            else return 0;
    }
}
```

```
//lcd.c
#include <reg51.h>
#include <absacc.h>
#define INT8U unsigned char
#define LCD_CMD_WR      0x00//接口扩展地址为 0x8000
#define LCD_DATA_WR     0x01//接口扩展地址为 0x8001
#define LCD_BUSY_RD     0x02//接口扩展地址为 0x8002
#define LCD_DATA_RD     0x03//接口扩展地址为 0x8003
#define LCD_CLS         1       //清屏命令
#define LCD_HOME    2       //光标归位
#define LCD_SETMODE     4       //模式设置
#define LCD_SETVISIBLE 8        //开显示
#define LCD_SHIFT       16      //移位方式
#define LCD_SETFUNCTION 32      //功能设置
#define LCD_SETCGADDR   64      //设置 CGRAM 地址
#define LCD_SETDDADDR   128     //设置 DDRAM 地址
sbit bflag = ACC^7;             //忙标志位
```

```
//-----------------------------------------------------------------
// 忙等待
//-----------------------------------------------------------------
void busywait()                         { do{ ACC = XBYTE[0x8002];}    while (bflag == 1); }
//-----------------------------------------------------------------
// 向 LCD 写命令字节
//-----------------------------------------------------------------
void Write_CMD(INT8U cmd) { XBYTE[0x8000] = cmd; busywait();}
//-----------------------------------------------------------------
// 向 LCD 写数据字节
//-----------------------------------------------------------------
void Write_Dat(char dat)       { XBYTE[0x8001] = dat; busywait(); }
//-----------------------------------------------------------------
// 清屏并将显示位置起点设置在最右边
//-----------------------------------------------------------------
void Clearscreen()                { Write_CMD(LCD_CLS); Write_CMD(LCD_SETDDADDR + 15);}
//-----------------------------------------------------------------
// 初始化 LCD
//-----------------------------------------------------------------
void Initialise_LCD()
{
      Write_CMD(0x30);//1 行 8 位
      Write_CMD(LCD_SETVISIBLE + 4);//显示开，关光标
      Write_CMD(LCD_SETDDADDR + 15);//从右边开始显示
      Write_CMD(LCD_SETMODE + 3);//递增左移

}
//-----------------------------------------------------------------
// LCD 显示字符串
//-----------------------------------------------------------------
void LCD_Disp_String(char buf[])
{
      INT8U i = 0;
      Clearscreen();//清屏
      while(buf[i]){
            Write_Dat(buf[i++]);//输出显示缓冲中的所有字符
      }
}
```

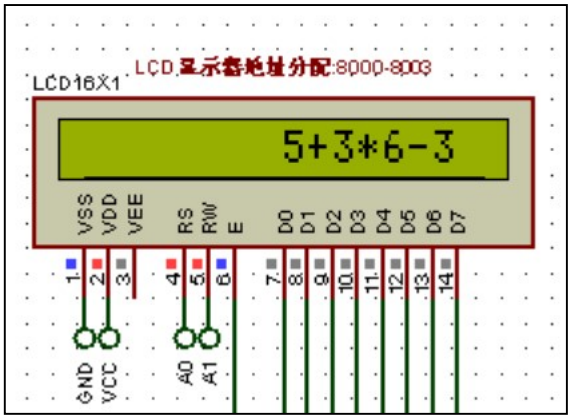
```
//keypad.c
#include <reg51.h>
#include <intrins.h>
```

```
char code keycodes[] =                          //矩阵键盘键值表
{     '7','8','9','/',
      '4','5','6','*',
      '1','2','3','-',
      'C','0','=','+'
};
//上述数组也可以定义为:
//char code keycodes = "789/456*123-C0=+";
char xdata keyflags[4][4];                  //16 键键位状态标识数组(1:按下  0:未按下)
//-----------------------------------------------------------------
//  获取键盘按键字符子程序
//-----------------------------------------------------------------
char GetKeyChar()
{
      char r,c,ColData = 0;
      for(r=0;r<4;r++)//循环扫描 4 行
      {
            P2 = (0xEF<<r); _nop_();//P2 输出行扫描码(初值 0xEF:1110 1111)
            ColData = P1 & 0x0F;//从 P1 端口读取列码数据()
            for(c=0;c<4;c++){//循环检查当前列的 4 行
                  //如果当前的 i 行 j 列有键按下
                  if( (ColData & (1<<c) ) == 0x00 ){
                        //且该位此前标识为 0(即无键按下，或按下后释放了)
                        if(keyflags[r][c]==0){
                              keyflags[r][c] = 1;
                              P2 = 0xFF;//结束扫描，在 P2 端口放置全 1
                              return keycodes[r*4+c];//最后返回键值 ASCII 码
                        }
                        else{
                              return 0;//否则表示虽然检测到改键按下，但此前状态亦为按下，故
不返回键值而返回 0
                        }
                  }
                  else{
                        keyflags[r][c] = 0;//在当前位置无键按下，keyflags 对应位置置 0
                  }
            }
      }
      P2 = 0xFF;//扫描结束，当前无键按下
      return 0;
}
```

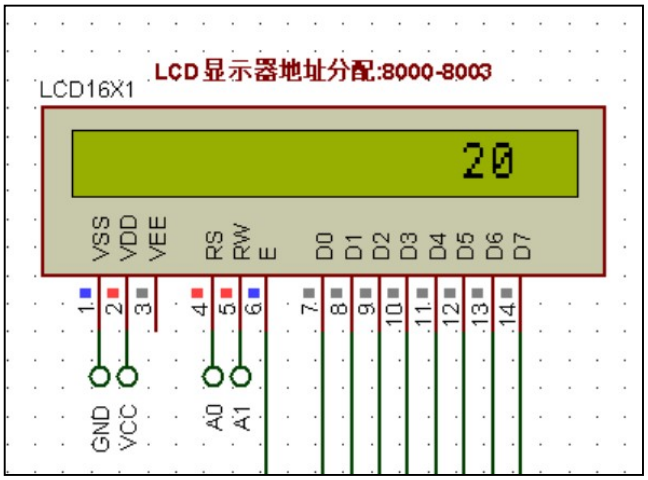
```
// calc.h  头文件
#define MAX_DISPLAY_CHAR 9     //定义适合屏幕显示的 ASCII 字符的最大个数
enum ERROR { OK = 0, SLEEP = 1, ERROR = 2};      //  错误处理状态
//-------------------------------------------------------------------
//  函数声明
//-------------------------------------------------------------------
void Operator_Process(char token);
int    calc_chkerror(long num);
void LCD_Disp_String(char buf[]);
void Initialise_LCD();
char GetKeyChar();
void Clearscreen();
```

4. 运行结果

输入算式：



得出答案：



5. 心得体会

通过单片机课程的学习以及本次课设的实践，我掌握了读取单片机键盘的原理与方法。了解了 lcd 显示器的显示原理与技术。并且对编译原理中平衡符号以及中缀表达式转后缀/逆波兰表达式的实现有了深刻的理解与运用，用以实现计算器中的计算顺序优先级。对数据结构中栈的理解更加深刻，并且用数组实现了栈的数据结构。对所学的知识融会贯通，对计算机的理解更深一步。