

# 第二次作业实验报告

信工 1602 班 孟繁阳 2016014507

## 【作业要求】

- (1) 将自己的学号以**长整型**和**浮点**的形式保存在**片外数据区指定的位置**，由自己指定位置，分析其在**存储器中的表示方法**，并将其换算成对应的十进制数，比较是否存在**表示误差**。
- (2) 将 1602 显示模块，与 STC 单片机实验箱正确连接，并在 1602 上**显示学号**。
- (3) 通过外部按键触发**中断**，实现学号在 1602 上的**左移/右移**。

## 【设计思路】

①第一问：这一问的主要重点在于一定要实现**指定位置保存**。通过查阅书籍我们可以知道，运用μkeil 中的 **\_at\_** 指令，即**绝对地址定位指令**，可以实现将特定数据存储在**片外数据区 xdata** 中的指定位置中。接下来，考虑到实际使用过程中，为了**防止学号这个变量在程序运行过程中被优化**，我们应该在**变量定义之前加上 volatile** 来达到这个目的，这样存储到指定位置的问题就被解决了。计算以及更多的相关解释将在后续进行详细阐述。

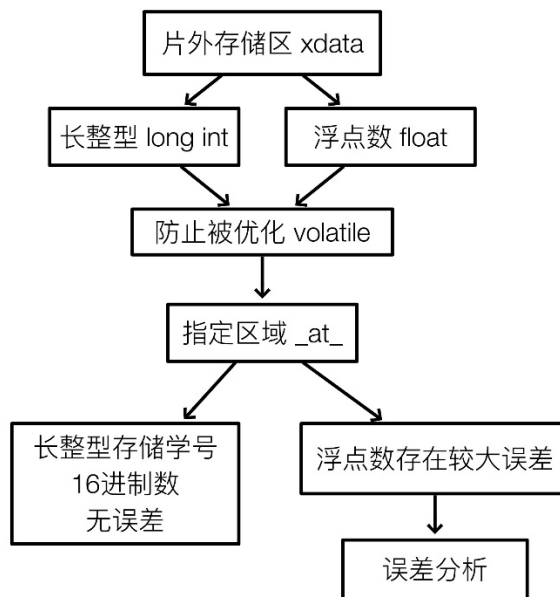


图 2.1 第(1)问思维导图

②第二问：实现学号在 1602 模块上的显示。解决此问题首先需要了解 1602 相关的原理。我在查询相关的 1602 模块原理时在网查到了 1602 显示模块的使用手册。我觉得非常有必要读一下这个手册。

要想实现显示操作，应该分为两部分：**初始化和显示**。

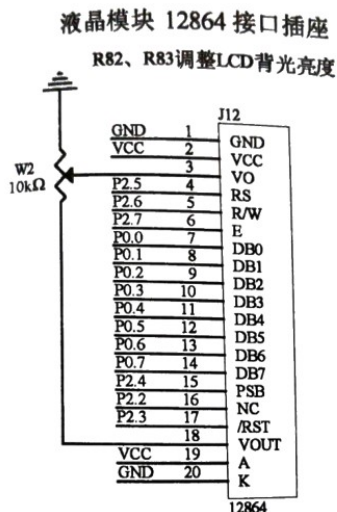


图 2.2 原理图

③第三问：使用中断实现学号的左移右移，首先就是要使能与按键对应的 0 号中断和 1 号中断，并设置为按键下降沿有效，同时使能中断响应。

接下来处理左移右移的问题，如果想要实现在任何位置，按下对应中断均能改变左移右移状态的话，需要设置一个标志位，标志位用来决定目前状态是左移还是右移，然后在无限循环中设定判决，判断标志位的状态，进入相应的循环左移右移。这样能够不受学号目前在位数限制，随时想改移动方向都可以。

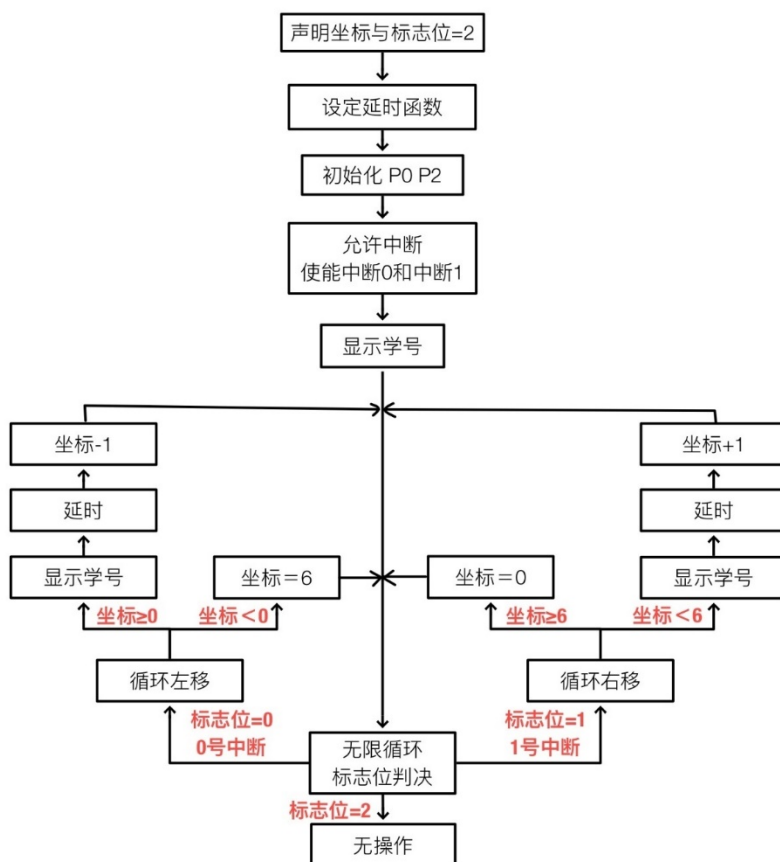


图 2.3 第(2)(3)问思维导图

### 【问题详解】

#### 【第 (1) 问】

##### 【设计代码】

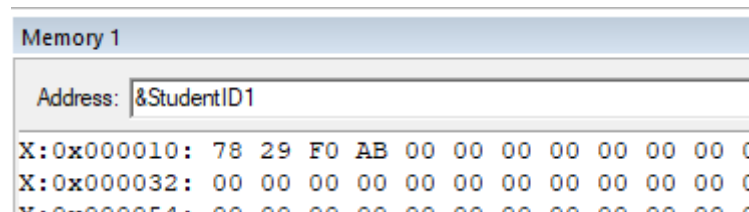
```
/******  
/*将自己的学号以长整形和浮点的形式保存在片外数据区指定的位置*/  
/******  
#include "reg51.h"  
#include "stdio.h" //加载标准头文件  
  
xdata volatile long int StudentID1 _at_ 0x10; //定义长整型学号，存储在  
0x10 区域  
xdata volatile float StudentID2 _at_ 0x1C; //定义浮点数值学号，存储在  
0x1C 区域  
  
void main()  
{  
    StudentID1=2016014507;  
    StudentID2=2016014507; //定义学号的具体数值  
}
```

##### 【问题相关计算】

$$(2016014507)_{10} = (7829F0AB)_{16}$$

##### 【在线调试】

首先，我们定位到 StudentID1 所在的地址，查看长整型变量是否存储成功并验证是否存在误差：

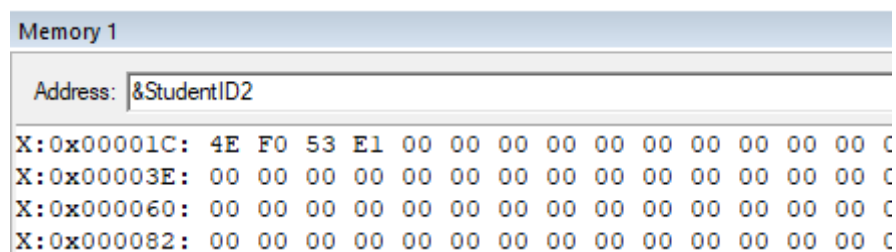


Memory 1	
Address: &StudentID1	
X:0x000010:	78 29 F0 AB 00 00 00 00 00 00 00 00 00 00 00 00
X:0x000032:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x000054:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

图 2.4 StudentID1 验证

由图 2.3 可以看出，存储为长整型变量的 StudentID1 以 16 进制数字的形式将我的学号存储在了我设定的位置中，并且是不存在任何误差的。

接着我们定位到 StudentID2 所在的地址，验证浮点数是否存储成功并是否存在误差：



Memory 1	
Address: &StudentID2	
X:0x00001C:	4E F0 53 E1 00 00 00 00 00 00 00 00 00 00 00 00
X:0x00003E:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x000060:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x000082:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

图 2.5 StudentID2 验证

我们可以看到，原本的  $(2016014507)_{10}$  在以浮点数存储之后变为了  $(4EF053E1)_{16}$ 。  
首先我们先来将 16 进制数字转换为 2 进制数

0 1 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0 0 1

其中：

- ① 黄色的 0 表示**符号位**，可知当前为**正数**。
- ② 绿色的 100,1110,1 表示**阶数**，对应的十进制为 **157**。在浮点标准中，这个值已经加上了偏移量 127，所以**实际的阶数为**  $157 - 127 = 30$ ，对应于  $2^{30} = 1073741824$ 。
- ③ 蓝色的 111,0000,0101,0011,1110,0001 表示**尾数**，对应的十进制小数为 0.8775597810745239，因为总是隐含 1，所以表示的**实际值为** **1.8775597810745239**。  
**因此**  $1.8775597810745239 \times 2^{30} = 2016014463.999$

可以看出，对比于我的学号 2016014507 来说，误差达到了 43 之多。经过分析，其实这是因为是单精度浮点数的缘故，由于学号数字过大，这个误差实在允许范围之内的。

## 【第 (2) 问】

### 【问题详解】

由于我们在这一问中用到了 1602 模块，所以了解 1602 的初始化和读写操作流程是很有必要的

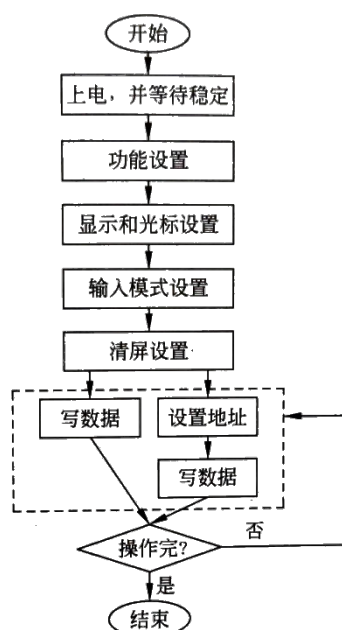


图 2.6 1602 初始化和读写操作流程

依照这张图的指示，我们可以写出 1602 的相关程序内容：

```

#include "led1602.h"
void lcdwait() //读忙，检测 LCD 是否处于忙状态
{
    LCD1602_DB=0xFF; //把 P0 端口设置为 0xFF
    _nop_();
}
  
```

```

    _nop_();
    _nop_();
    _nop_();
    LCD1602_RS=0; //设置 RS=0, RW=1, 对应的 1602 操作为读忙 (BF) 操作
    LCD1602_RW=1; //并且读取地址计数器的内容 (DB0~DB6)
    LCD1602_E=1; //读写操作均需要现将 E 拉高, 然后进行读写, 结束之后清零 E
    while(LCD1602_DB & 0x80); //等待标志位 BF 为低, 表示 LCD 空闲
    LCD1602_E=0; //将 LCD 的 E 信号拉低
}

```

```

void lcdwritecmd(unsigned char cmd) //传输指令
{
    lcdwait(); //等待 LCD 不忙
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    LCD1602_RS=0;
    LCD1602_RW=0; //RS=0, RW=0, 对应为写入数据到指令寄存器
    LCD1602_DB=cmd; //将指令控制码 cmd 放到 P0 端口
    LCD1602_E=1; //拉高 E, 开始传输指令
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    LCD1602_E=0; //将 LCD 的 E 信号拉低
}

```

```

void lcdwritedata(unsigned char dat) //传输数据
{
    lcdwait(); //等待 LCD 不忙
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    LCD1602_RS=1;
    LCD1602_RW=0; //RS=1, RW=0, 对应为写入数据到数据寄存器
    LCD1602_DB=dat; //将数据码 cmd 放到 P0 端口
    LCD1602_E=1; //拉高 E, 开始传输数据
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    LCD1602_E=0; //将 LCD 的 E 信号拉低
}

```

```

}

void lcdinit() //初始化 LCD
{
    lcdwritcmd(0x38); //设置数据宽度为 8 位, 2 行 5*8 字符字体
    lcdwritcmd(0x0c); //打开显示, 关闭光标, 关闭闪烁
    lcdwritcmd(0x06); //地址递增, 关闭文字移动
    lcdwritcmd(0x01); //清屏指令
}

void lcdsetcursor(unsigned char x, unsigned char y) //定位显示坐标
{
    unsigned char address;
    if(y==0) //如果第一行
        address=0x00+x; //存储器地址从 0x00 开始
    else //如果第二行
        address=0x40+x; //存储器地址从 0x40 开始
    lcdwritcmd(address|0x80); //写当前地址
}

void lcdshowstr(unsigned char x, unsigned char y,
                unsigned char *str) //在液晶屏上制定的 x 和 y 位置显示字符
{
    lcdsetcursor(x,y); //设置显示 RAM 的地址
    while((*str)!='\0') //如果不是字符串的结尾则继续
    {
        lcdwritdata(*str); //发写数据指令, 在 LCD 上显示数据
        str++; //指针+1, 指向下一个地址
    }
}

```

以上指令关于 RS 和 R/W 的部分可以在书上找到此表格对应:

RS	R/W	操作说明
0	0	写入指令寄存器(清屏)
0	1	读 BF(忙)标志,并读取地址计数器的内容
1	0	写入数据寄存器(显示各字型等)
1	1	从数据寄存器读取数据

图 2.7 1602 字符 LCD 读写操作指令信号

至此, 所需的有关于 1602 部分的函数全部定义完毕。

接下来进行主程序的编辑。

回顾一下设计思路部分的思维导图, 可以知道我们首先需要完成需要用到的变量和相关函数与中断的定义:

```

int x=0;
int y=0; //定义显示字符的初始坐标

```

```
int flag=2; //定义标志位初始为 2，后续作为决定左右移
```

```
void delay() //设定延时函数
```

```
{
    unsigned long int i,j;
    for(i=200;i>0;i--)
        for(j=400;j>0;j--);
}
```

```
service_int2() interrupt 2 //定义 1 号中断
```

```
{
    flag=1; //将标志位变为 1，实现右移
}
```

```
service_int0() interrupt 0 //定义 0 号中断
```

```
{
    flag=0; //将标志位变为 0，实现左移
}
```

接下来继续依照思路，进行主函数的编写：

```
void main()
```

```
{
    unsigned int m; //设定参数 m 用作后续延时
    P0M0=0; //通过 P0M0 和 P0M1 寄存器将 P0 口
    P0M1=0; //定义为准双向，弱上拉
    P2M0=0; //通过 P2M0 和 P2M1 寄存器将 P2 口
    P2M1=0; //定义为准双向，弱上拉
    for(m=0;m<10000;m++); //延时函数，加上延时防止后续通信出现问题
```

```
    IT1=1; //设置外部中断 1 下降沿有效
    EX1=1; //设置外部中断 1 有效
    IT0=1; //设置外部中断 0 下降沿有效
    EX0=1; //设置外部中断 0 有效
    EA=1; //使能中断
```

```
    lcdwait(); //读忙操作，等待 1602 字符 LCD 稳定
    lcdinit(); //初始化 1602 字符 LCD 模块
    lcdshowstr(x,y,"2016014507"); //显示学号
```

```
while(1) //进入无限循环
```

```
{
    if(flag==1) //判断标志位是否为 1
    {
```

```
        if(x<6) //因为学号有 10 位，1602 一行 16 位，所以右移最大到 5 为止
```

```

{
    lcdwait();           //读忙操作，等待 1602 字符 LCD 稳定
    lcdinit();           //初始化 1602 字符 LCD 模块
    lcdshowstr(x,y,"2016014507"); //显示学号
    delay();             //延时一段时间
    x++;                 //横向右移
    y=~y;                //行数取反，实现了波浪形换行显示
}
else //如果 x 大于等于 6
{
    x=0; //将 x 恢复为 0，从头右移
}

}
else if(flag==0) //判断标志位是否为 0
{
    if(x>=0) //左移最小坐标是 0
    {
        lcdwait();           //读忙操作，等待 1602 字符 LCD 稳定
        lcdinit();           //初始化 1602 字符 LCD 模块
        lcdshowstr(x,y,"2016014507"); //显示学号
        delay();             //延时一段时间
        x--;                 //横向左移
        y=~y;                //行数取反，实现了波浪形换行显示
    }
    else //如果 x 小于等于 0
    {
        x=6; //将 x 恢复为 6，从头左移
    }
}
else //如果标志位既不是 0 也不是 1 就不做任何操作
{

}

}
}

```

主程序的编辑到这里就告一段落。最后应该在头文件里加上以下三行引用：

```

#include "reg51.h"
#include "stdio.h"
#include "led1602.h"

```

至此，主程序的编辑结束。



## 【在线调试与结果展示】

关于这一部分的详细内容，可以查看同目录下的[视频文件](#)。

## 【总结与思考】

本次实验中，我遇到了许多涉及到底层的问题，比如通过查询书上的开发板原理图才知道为什么只初始化 P0 和 P2 寄存器，是因为 1602 模块和开发板相连接的部分用到了相关寄存器引脚，所以一定要在程序开始的时候初始化模式寄存器，将 P0 和 P2 设置为准双向弱上拉。

同时我也系统性的再次了解了 1602 的原理，也通过一次次的 debug 知道了很多有趣的算法实现。

当然最主要的是我发现如果想要系统性的学习一个模块，一定要仔细阅读它的使用手册，这样才能保证模块编程过程中的正确设置。