# 浙江大学实验报告

专业：生物医学工程
姓名：
学号：
日期：2025.4.22
地点：东 1B 416

课程名称：微机原理及其应用　　指导老师：陈星　　实验类型：微机实验
实验名称：中断模式下串行通信　　成　绩：　　签　名：

装 订 线

## 一、 实验目的和要求

### 1. 实验目的

本实验旨在通过实践加深学生对串行通信中断工作方式的理解，掌握单片机与上位机 PC 之间信息交互的技术。通过这一交互过程深入理解单片机串行通信中断工作方式的原理与应用，掌握相关程序的设计方法，并学会在实际操作中对程序进行调试。

### 2. 实验要求

(1). 通过 PC 电脑与开发板的串行接口连接，实现双向串行通讯，设定波特率为 4800。

(2). 在 PC 端输入任意字母表中的字符（单个字符），51 单片机需回复从此字符开始的后续 10 个字符，同时发送两个 3 位十进制数的计数值（分别表示单片机接收到的字符数量和发送出的字符数量）。

(3). 当输入字符达到字母表末尾时，遵循以下跳转规则：大写字母'Z' 后跳转至小写字母'a'，小写字母'z' 后跳转至数字'0'，数字'9' 后跳转至大写字母'A'。

(4). 必须考虑输入数据的合法性，对于不在处理范围内的字符，51 单片机回复“ERROR”，同时后续的两个 3 位十进制计数依然发送。

(5). 数据缓冲区不得设置在外部数据存储器。

(6). 对编写的程序先在 proteus 上仿真验证，后续用 keil 软件编译程序生成 HEX 文件，下载到开发板进行实机验证。

## 二、 实验内容和原理

### 1. 串行通信原理

串行通信是指数据一位一位地顺序传送，在单片机与 PC 之间进行串行通信时，需要遵循一定的通信速率。本实验中波特率设定为 4800，表示每秒传输 4800 个数据位（bit）。

数据在传输过程中，通过起始位、数据位、奇偶校验位（可无）和停止位等组成一组数据进行传输。

### 2. 中断工作方式原理

中断是指计算机在执行程序的过程中，当出现某些事件（如外部设备的数据到达）时，计算机停止当前正在执行的程序，转而去执行处理该事件的程序（中断服务程序），处理完毕后再返回原来被中断的程序处继续执行。

在本实验的串行通信中，采用中断方式可以及时响应 PC 端发送的数据，提高系统的实时性和效率。当单片机接收到来自 PC 端的字符时，会触发中断，然后在中断服务程序中置起串口接收数据标志位，并最终在主程序中进行处理。中断服务程序中的操作要越简单越好，常规的操作都是中断进来后置标志位然后在主程序查询标志位信息以决定是否去执行相应的操作。

装 订 线

## 三、 主要仪器设备

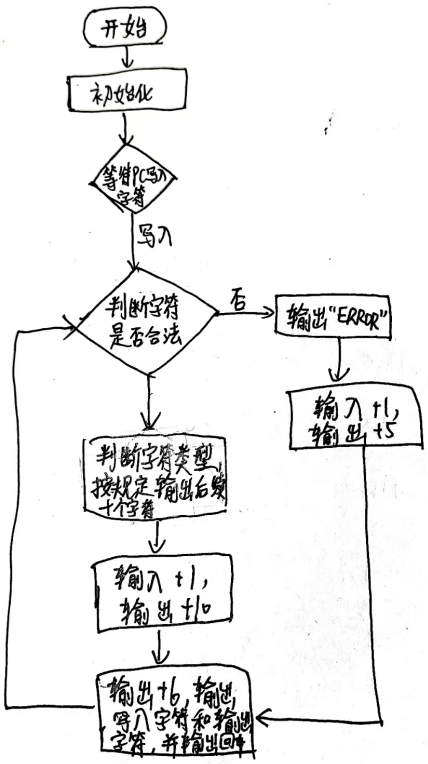
计算机、开发板：HC6800-ES V2.0 开发板、USB 转串口线（开发板自带）

## 四、 操作方法和实验步骤

### 1. 硬件电路设计

用到了 51 单片机、串口等器件，详见实验结果与分析部分的“硬件电路设计图”

### 2. 软件流程图



装 订 线

### 3. 程序代码

汇编版，输出字符最多记录 256 个，超过 256 个会溢出清零。

```
        ORG 0000H
        LJMP MAIN
        ORG 0023H
        LJMP PINT

        ORG 0100H
MAIN:   MOV TMOD,#20H  ; 定时器设置为模式 2，自动装填
        MOV TH1, #0F3H
        MOV TL1, #0F3H
        SETB TR1
        MOV SCON, #50H  ; 串口设置为方式 1，允许输入
        MOV PCON, #80H
        SETB EA
        SETB ES          ; 允许中断
        MOV R1, #00H    ;R1 判断输入的字符，0 为没有输入，不符合则为 1
        MOV R2, #00H
        MOV R3, #00H
        MOV R4, #00H    ; 以上三个用于判断输入的数字是否违规
        MOV R5, #00H    ; 记录接收几个
        MOV R6, #00H    ; 记录输出几个
        MOV R7, #00H     ;R7 存放需要输出的字符个数（10）
        MOV R0, #00H     ; 输入数字时用于暂存

LOOP:   CJNE R1, #00H, PHAS0     ; 没有输入，循环等待
        JMP LOOP
PHAS0: CJNE R1, #01H, PHAS1      ; 超出范围执行，输出 ERROR
        CLR ES
        MOV A, #45H             ; 输出 E(ERROR)
        INC R6
        MOV SBUF, A
        JNB TI, $            ; 检测是否发送完毕
        CLR TI
        MOV A, #52H        ; 输出 R
        INC R6
        MOV SBUF, A
        JNB TI, $
        CLR TI
        MOV A, #52H     ; 输出 R
        INC R6
        MOV SBUF, A
        JNB TI, $
        CLR TI
```

装 订 线

```
        MOV A, #4FH     ; 输出 O
        INC R6
        MOV SBUF, A
        JNB TI, $
        CLR TI
        MOV A, #52H     ; 输出 R
        INC R6
        MOV SBUF, A
        JNB TI, $
        CLR TI

        INC R5

        LCALL TOUT

        MOV R1, #00H     ; 恢复 R1
        SETB ES
        JMP LOOP

PHAS1:  MOV R7, #0AH    ; 输出字数设置为 10 个字
        CLR ES
PHAS2:  INC R1
        CJNE R1, #3AH, NEXT   ;9 跳到 Z
        MOV R1, #41H
NEXT:   CJNE R1, #5BH, NEXT2   ;Z 跳到 a
        MOV R1, #61H
NEXT2:  CJNE R1, #7BH, NEXT3   ;z 跳到 0
        MOV R1, #30H
NEXT3:  MOV A, R1       ; 输出此时的 R1 值
        MOV SBUF, A
        JNB TI, $
        CLR TI
        DEC R7
        INC R6
        CJNE R7, #00H, PHAS2  ;10 个没输出完，继续输出

        INC R5

        LCALL TOUT

        MOV R1, #00H
        SETB ES
        LJMP LOOP

TOUT:   MOV A, #20H     ; 空格
```

装 订 线

```
MOV SBUF, A
JNB TI, $
CLR TI

MOV B, #100
MOV A, R5
DIV AB
ADD A, #30H
MOV SBUF, A      ; 写入个数的百位
JNB TI, $
CLR TI

MOV A, B
MOV B, #10
DIV AB
ADD A, #30H
MOV SBUF, A      ; 十位
JNB TI, $
CLR TI

MOV A, B
ADD A, #30H
MOV SBUF, A     ; 个位
JNB TI, $
CLR TI

MOV A, #20H    ; 空格
MOV SBUF, A
JNB TI, $
CLR TI

MOV B, #100
MOV A, R6
ADD A, #06H
MOV R6,A
DIV AB
ADD A, #30H
MOV SBUF, A      ; 输出个数的百位
JNB TI, $
CLR TI

MOV A, B
MOV B, #10
DIV AB
ADD A, #30H
```

装 订 线

```
        MOV SBUF, A      ; 十位
        JNB TI, $
        CLR TI

        MOV A, B
        ADD A, #30H
        MOV SBUF, A     ; 个位
        JNB TI, $
        CLR TI

        MOV A, #0AH    ; 回车
        MOV SBUF, A
        JNB TI, $
        CLR TI
        RET

PINT:   CLR RI            ; 禁止输入
        MOV A, SBUF
        LCALL CHOSE         ; 判断是否在范围内
        RETI
CHOSE:  MOV R2, #30H
        MOV R3, #41H
        MOV R4, #61H
CHOSE1: MOV B, R2
         CJNE A, B, LOOP1   ; 判断是否为数字 0 到 9
         MOV R1, A
         JMP RETURN
LOOP1:  INC R2
        CJNE R2, #03AH, CHOSE1   ; 判断不是数字，跳至判断 A 到 Z
CHOSE2: MOV B, R3
         CJNE A, B, LOOP2    ; 判断是否为字母 A 到 Z
         MOV R1,A
         JMP RETURN
LOOP2:  INC R3
        CJNE R3, #05BH, CHOSE2
CHOSE3: MOV B, R4
         CJNE A, B, LOOP3    ; 判断是否为字母 a 到 z
         MOV R1,A
         JMP RETURN
LOOP3:  INC R4
        CJNE R4, #07BH, CHOSE3
        MOV R1, #01H
RETURN:
        RET
        END
```

C 语言版本，用 unsigned int 类型存储累计输出字符总数，输出字符最多能存储 65536 个

```c
#include <reg52.h>
unsigned char receive_data;         // 存储从串口接收到的字符
bit uart_flag = 0;                  // 串口接收完成标志位
unsigned int receive_num_total = 0; // 累计接收字符总数
unsigned int send_num_total = 0;    // 累计发送字符总数
unsigned char receive_num = 0;      // 单次接收字符计数（每次中断 +1）
unsigned char send_buffer[20];      // 发送缓冲区（10 字符 + 3 接收计数 + 5 发送计数）


unsigned char num_array[10] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};

void UART_Init();
void FrameProcess();

void UART_Init() {
    SCON = 0x50;    // 串口模式 1（8 位 UART），允许接收
    TMOD = 0x20;   // 定时器 1 模式 2（自动重装）
    PCON = 0X80;
    TH1 = 0xF3;     // 波特率 4800（12MHz 晶振计算值）
    TL1 = 0xF3;     // 定时器初值
    TR1 = 1;        // 启动定时器 1
    ES = 1;         // 允许串口中断
    EA = 1;         // 开总中断
}

void Usart() interrupt 4 {
    if (RI == 1) {            // 接收中断触发
        receive_data = SBUF;  // 读取接收到的字符
        RI = 0;               // 清除接收中断标志
        receive_num++;        // 单次接收计数 +1（实验要求每次收 1 字符）
        uart_flag = 1;        // 置位处理标志
    }
}


void main() {
    UART_Init();        // 初始化串口
    while (1) {         // 主循环
        if (uart_flag) {              // 检测到接收完成标志
            FrameProcess();           // 处理数据并发送（用户提供代码）
            uart_flag = 0;           // 清除标志位
        }
    }
}
```

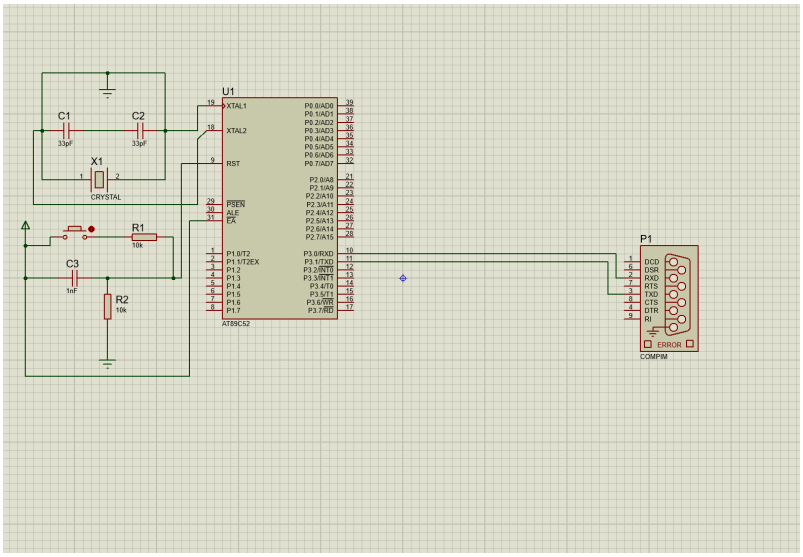
装 订 线

```
void FrameProcess() {
    unsigned char i, temp;
    unsigned char valid_data = receive_data;
    unsigned char send_cnt = 0;
    if (valid_data >= 0x30 && valid_data <= 0x39) { // 数字 0-9
        temp = valid_data;
        for (i = 0; i < 10; i++) {
            if (temp >= 0x39) {
                temp = valid_data + 8 + i; // 9→A
            } else {
                temp = valid_data + i + 1;
            }
            send_buffer[send_cnt++] = temp;
        }
    } else if (valid_data >= 0x41 && valid_data <= 0x5A) { // 大写 A-Z
        temp = valid_data;
        for (i = 0; i < 10; i++) {
            if (temp >= 0x5A) {
                temp = valid_data + 7 + i; // Z→a
            } else {
                temp = valid_data + i + 1;
            }
            send_buffer[send_cnt++] = temp;
        }
    } else if (valid_data >= 0x61 && valid_data <= 0x7A) { // 小写 a-z
        temp = valid_data;
        for (i = 0; i < 10; i++) {
            if (temp >= 0x7A || temp < 0x61) {
                temp = valid_data - 74 + i; // z→0
            } else {
                temp = valid_data + i + 1;
            }
            send_buffer[send_cnt++] = temp;
        }
    } else { // 非法字符，发送"ERROR"
        send_buffer[send_cnt++] = 0x45; // 'E'
        send_buffer[send_cnt++] = 0x52; // 'R'
        send_buffer[send_cnt++] = 0x52; // 'R'
        send_buffer[send_cnt++] = 0x4F; // 'O'
        send_buffer[send_cnt++] = 0x52; // 'R'
    }

    receive_num_total += receive_num; // 累计接收总数
    receive_num = 0;                  // 清零单次计数
```

```
    send_buffer[send_cnt++] = num_array[receive_num_total / 100];        // 百位
    send_buffer[send_cnt++] = num_array[(receive_num_total % 100) / 10];// 十位
    send_buffer[send_cnt++] = num_array[receive_num_total % 10];         // 个位

    send_num_total += send_cnt + 5;     // 累计发送总数（含后续 5 位）
    send_buffer[send_cnt++] = num_array[send_num_total / 10000];          // 万位
    send_buffer[send_cnt++] = num_array[(send_num_total / 1000) % 10];    // 千位
    send_buffer[send_cnt++] = num_array[(send_num_total / 100)%10];       // 百位
    send_buffer[send_cnt++] = num_array[(send_num_total % 100) / 10];     // 十位
    send_buffer[send_cnt++] = num_array[send_num_total % 10];             // 个位
    for (i = 0; i < send_cnt; i++) {
        SBUF = send_buffer[i]; // 写入发送缓冲区
        while (!TI);           // 等待发送完成
        TI = 0;                // 清除发送中断标志
    }
}
```

## 五、　实验结果与分析

### 1. 硬件电路设计图



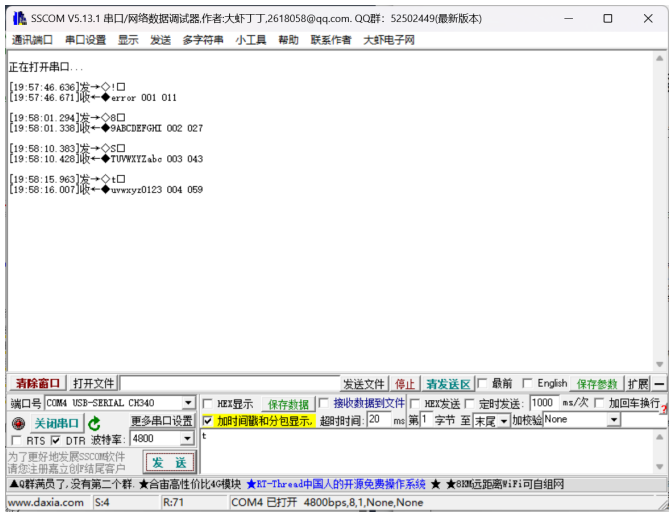

### 2. 软件编写经历

本次实验代码的整体逻辑还是比较清晰的，就是先判断串口输入数据的类型，再按照相应的规则进行输出。本次实验给出的例程是按照 C 语言来写的，整体 if/else 的框架十分清晰，能较快速的理解整个代码的功能。但由于我的汇编代码能力并不是特别好，所以一开始仍然选择使用汇编代码来完成本次代码的编写。经过本次的编写，发现汇编的分支结构其实也是比较固定的，只需要使用 CJNE 来判断几个关键节点即可。然后编写代码时有些细节不能忘记，比如最后输出的回车。

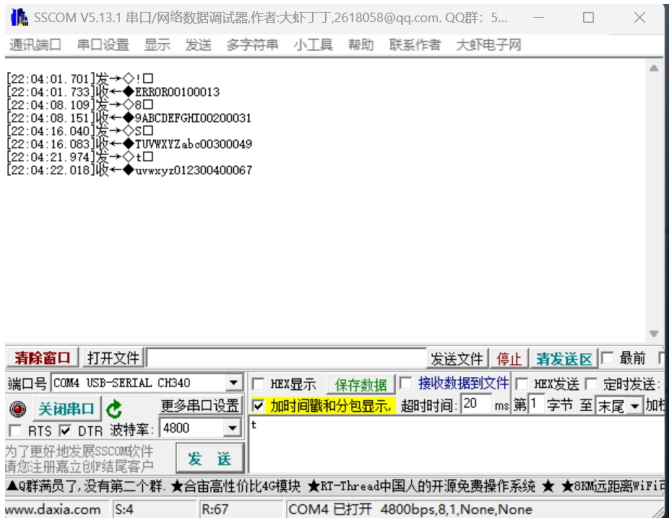
后面上课时老师提到由于 51 单片机的寄存器是 8 位的，所以输出的字符数达到 256 时就会溢出，所以后面参考实验文件中 C 语言的例程重新写了代码，实验文件中关于数据的输出已经比较完整了，只需要补上主循环，相关中断和串口通讯的初值即可。

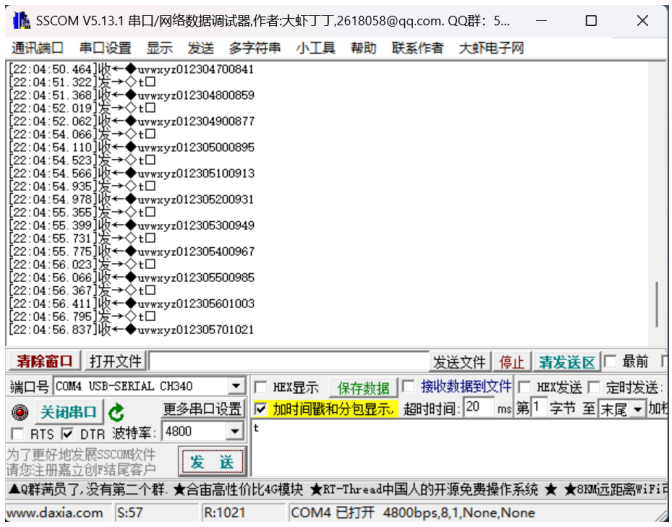
### 3. 软件仿真运行结果

汇编版本的程序，输入测试数据：!、8、S、t 后，串口输出结果如下：



改进后的 C 语言程序，输入测试数据：!、8、S、t 后，串口输出结果如下，并不断重复输入，测试能够存储的输出最大字符。




## 六、　讨论、心得

本次实验过后，我对于 51 单片机串行通讯的实现有了更加清晰的认识，首先是对于波特率的计算方法，再是串行通讯的具体实现思路，最后还有对于 keil 软件的功能和使用有了更好的认识。