苏州大学实验报告

院、系	计算机学院	年级专业	计算机科学与技术	姓名	柯骅	学号	2027405033
课程名称	微型计算机技术					成绩	
指导教师	姚望舒	同组实验者	无	实验日期	2023.05.30		

实验名称：实验五：理解中断与定时器

一. 实验目的

（1）熟悉定时中断计时的工作及编程方法。

（2）进一步深入理解 MCU 的串口通信的编程方法。

二. 实验准备

（1）硬件部分。PC 机或笔记本电脑一台、开发套件一套。

（2）软件部分。根据电子资源“..\02-Doc”文件夹下的电子版快速指南，下载合适的电子资源。

（3）软件环境。按照电子版快速指南中“安装软件开发环境”一节，进行有关软件工具的安装。

三. 实验参考样例

参照“Exam8_1”工程。该程序通过 SYSTICK 定时器，每一秒将红灯亮暗状态反转一次并输出相应的时间和提示信息。

四. 实验过程或要求

（1）验证性实验

① 下载开发环境 AHL-GEC-IDE。根据电子资源下“..\05-Tool\AHL-GEC-IDE 下载地址.txt”文件指引，下载由苏州大学-Arm 嵌入式与物联网技术培训中心（简称 SD-Arm）开发的金葫芦集成开发环境（AHL-GEC-IDE）到“..\05-Tool”文件夹。该集成开发环境兼容一些常规开发环境工程格式。

② 建立自己的工作文件夹。按照“分门别类，各有归处”之原则，建立自己的工作文件夹。并考虑随后内容安排，建立其下级子文件夹。

③ 拷贝模板工程并重命名。所有工程可通过拷贝模板工程建立。例如，“\04-Soft\ Exam4_1”工程到自己的工作文件夹，可以改为自己确定的工程名，建议尾端增加日期字样，避免混乱。

④ 导入工程。在假设您已经下载 AHL-GEC-IDE，并放入“..\05-Tool”文件夹，且按安装电子档快速指南正确安装了有关工具，则可以开始运行“..\05-Tool\AHL-GEC-IDE\AHL-GEC-IDE.exe”文件，这一步打开了集成开发环境 AHL-GEC-IDE。接着单击“ ”→“ ”→导入你拷贝到自己文件夹并重新命名的工程。导入工程后，左侧为工程树形目录，右边为文件内容编辑区，初始显示 main.s 文件的内容。

⑤ 编译工程。在打开工程，并显示文件内容前提下，可编译工程。单击“ ”→“ ”，则开始编译。

⑥ 下载并运行。

步骤一，硬件连接。用 TTL-USB 线（Micro 口）连接 GEC 底板上的“MicroUSB”串口与电脑的 USB 口。

步骤二，软件连接。单击“ ”→“ ”，将进入更新窗体界面。点击“ ”查找到目标 GEC，则提示“成功连接……”。

步骤三，下载机器码。点击“ ”按钮导入被编译工程目录下 Debug 中的.hex 文件（看准生成时间，确认是自己现在编译的程序），然后单击“ ”按钮，等待程序自动更新完成。

此时程序自动运行了。若遇到问题可参阅开发套件纸质版导引“常见错误及解决方法”一节，也可参阅电子资源“..\02-Doc”文件夹中的快速指南对应内容进行解决。

⑦ 观察运行结果与程序的对应。

第一个程序运行结果（PC 机界面显示情况）见图 4-7。为了表明程序已经开始运行了，在每个样例程序进入主循环之前，使用 printf 语句输出一段话，程序写入后立即执行，就会显示在开发环境下载界面的中的右下角文本框中，提示程序的基本功能。

利用 printf 语句将程序运行的结果直接输出到 PC 机屏幕上，使得嵌入式软件开发的输出调试变得十分便利，调试嵌入式软件与调试 PC 机软件几乎一样方便，改变了传统交叉调试模式。实验

步骤和结果

（2）设计性实验

复制样例程序，利用该程序框架实现：通过集成开发环境 AHL-GEC-IDE 中的“串口-串口工具”，发送当前系统时间如：“10:55:12”来设置开发板上的初始计时时间。

（3）进阶实验★

复制样例程序，利用该程序框架实现：通过 C#程序界面显示系统当前时间，通过按钮发送当前系统时间给 MCU 来设置开发板上的初始计时时间，MCU 将计时时间发送给 C #进行显示。C#界面设计如图 8-4 所示。不断电运行一段时间后，请给出 MCU 和 PC 系统时间的误差是多少？

请在实验报告中给出 MCU 端程序 main.c 和 isr.c 流程图及程序语句和 C#方主要程序段。

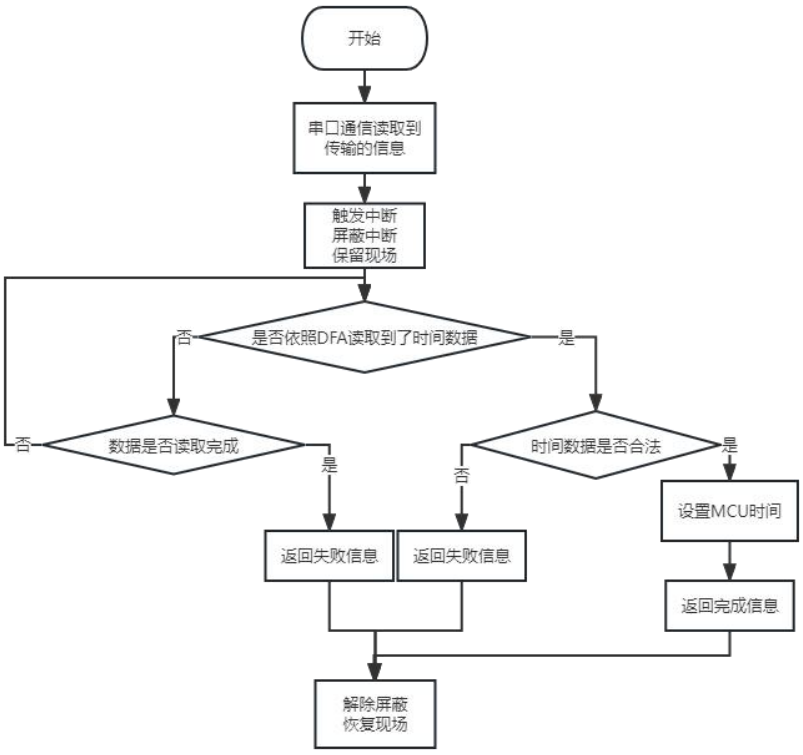
图 1　C#界面设计

四、实验结果

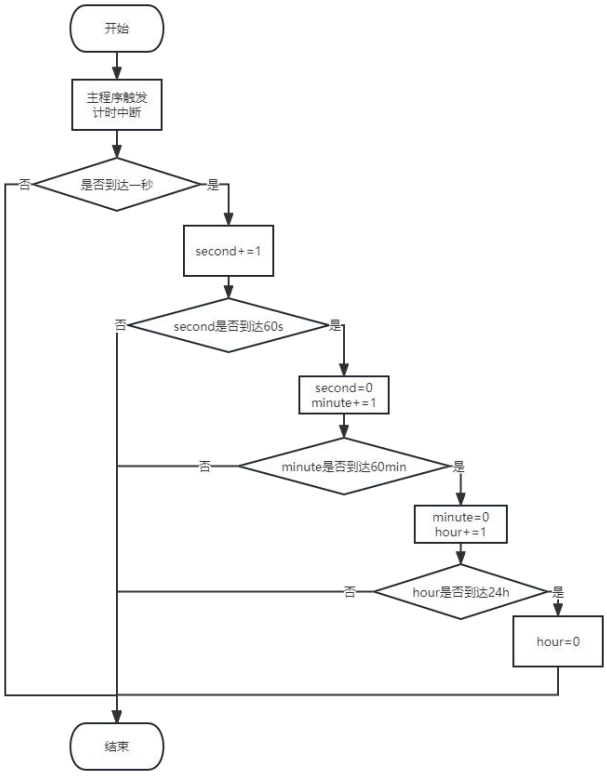
（1）用适当文字、图表描述实验过程。

在本实验中，同样使用类似实验四中的自动机来判定读取到的数据是否属于设置时间的指令，并作出相应操作。其中，中断程序分为两部分，即两种方式会出发中断程序，分别为串口通信、计时跳转，其中，串口通信中断用于处理串口通信信息，计时跳转中断用于程序计时。利用计时中断可以实时的计算出当前的时间，在串口通信接收到需要更改的时间之后，串口通信中断可以判断指令是否正确并更新设备的系统时间。以下分别为串口通信和计时跳转的流程图：

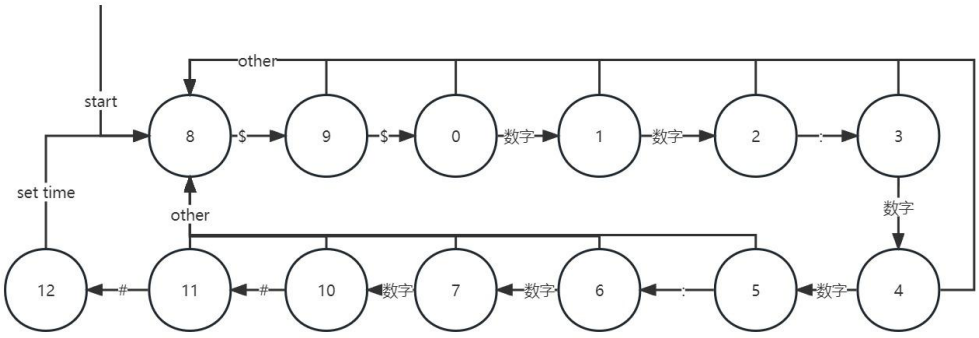
串口通信：



计时跳转中断流程图：

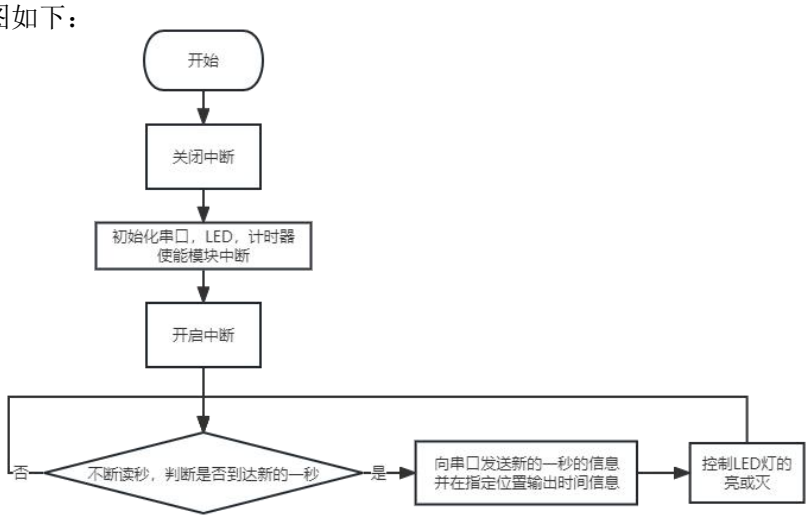


其中，串口通信的 DFA 图如下：



主程序中，不断利用计时跳转来判断是否到达了新的一秒，如果到达了新的一秒，那么就按照要求控制 LED 灯的闪烁，并将相关信息通过串口输出到指定位置。

main 的流程图如下：

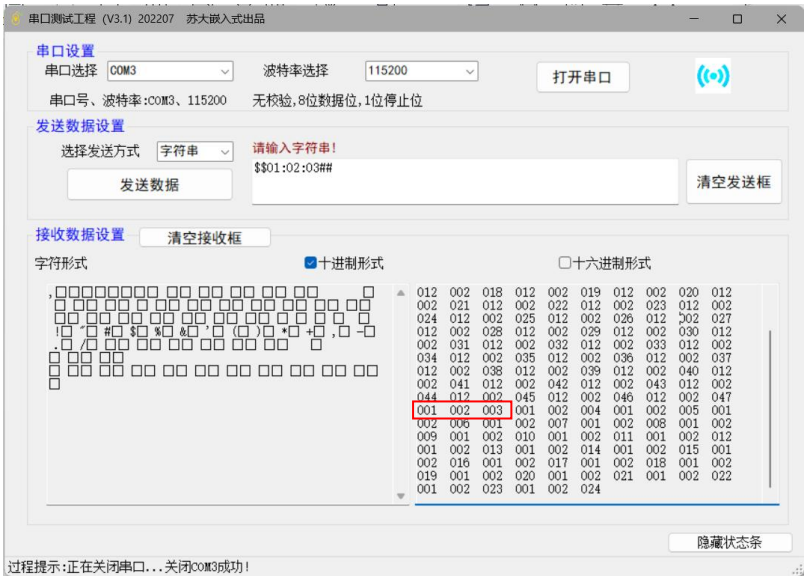


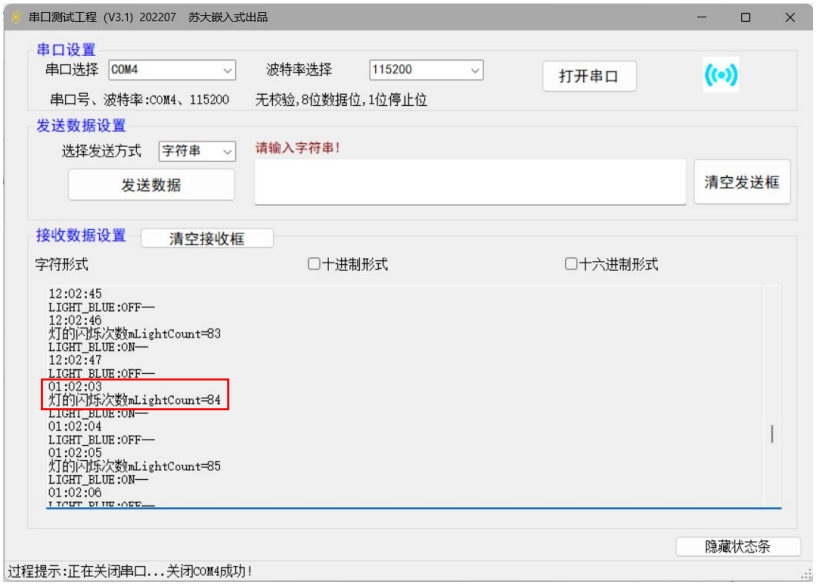
教务处制

实验结果：

将编译好的 hex 文件通过串口更新进设备中，结果如下：

同时打开两个串口传输工具，分别连接至设备的两个串口，发送修改时间的指令，结果如下：





可以发现，指令成功改变了 MCU 的时间，实验成功。

教务处制

（2）完整给出定时器操作的代码片段

//设计性实验中对定时器操作的代码片段

//isr.s

```
//=========================================================================
//程序名称：UARTA_Handler（UARTA 接收中断处理程序）
//触发条件：UART_User 串口收到一个字节触发
//程序功能：接收 PC 通过串口通信传输的时间
//=========================================================================
USART2_IRQHandler:
    cpsid  i           //屏蔽中断
    push {r0-r7,lr}    //保护现场
    sub sp,#4
    mov r7,sp        //r7 存储由 sp 指针偏移获取到的数据
//接收一个字节
    mov r1,r7
    mov r0,#UARTA
    bl  uart_re1      //调用函数接收一个字节
    ldr r1,=received_data//分析接收到的数据
    str r0,[r1]
    //根据 DFA 进行状态转换??
    ldr r3,=recv_state
    ldr r2,[r3]
    cmp r2,#0
    beq USART2_IRQHandler_state0
    cmp r2,#1
    beq USART2_IRQHandler_state1
    cmp r2,#2
    beq USART2_IRQHandler_state2
    cmp r2,#3
    beq USART2_IRQHandler_state3
    cmp r2,#4
    beq USART2_IRQHandler_state4
    cmp r2,#5
    beq USART2_IRQHandler_state5
    cmp r2,#6
    beq USART2_IRQHandler_state6
    cmp r2,#7
    beq USART2_IRQHandler_state7
    cmp r2,#8
    beq USART2_IRQHandler_state8
    cmp r2,#9
    beq USART2_IRQHandler_state9
    cmp r2,#10
    beq USART2_IRQHandler_state10
```

```
    cmp r2,#11
    beq USART2_IRQHandler_state11
    bl  USART2_IRQHandler_exit
//获取小时
USART2_IRQHandler_state0:
//0--num-->1
//0--other-->8
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state0_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state0_other
    sub r0,r0,#'0'
    ldr r1,=new_hour    //将接收到的小时数字的十位数存入 new_hour
    str r0,[r1]
    mov r0,#1           //0--num-->1
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state0_other:
    mov r0,#8           //0--other-->8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state1:
//1--num-->2
//1--other-->8
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state1_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state1_other
    //收到字符 0-9
    sub r0,r0,#'0'      //获取小时数的个位数
    ldr r3,=new_hour    //获取完整的小时
    ldr r1,[r3]
    mov r2,#10
    mul r1,r2,r1
    add r0,r0,r1
    str r0,[r3]
    mov r0,#2           //1--num-->2
    ldr r1,=recv_state
    str r0,[r1]
```

```
88.      bl  USART2_IRQHandler_exit
89.  USART2_IRQHandler_state1_other:
90.      mov r0,#8           //1--other-->8
91.      ldr r1,=recv_state
92.      str r0,[r1]
93.      bl  USART2_IRQHandler_exit
94.  USART2_IRQHandler_state2:
95.  //2--:-->3
96.  //2--other-->8
97.      ldr r0,=received_data
98.      ldr r0,[r0]
99.      cmp r0,#':'
100.     bne USART2_IRQHandler_state2_other
101.     mov r0,#3           //2--:-->3
102.     ldr r1,=recv_state
103.     str r0,[r1]
104.     bl  USART2_IRQHandler_exit
105. USART2_IRQHandler_state2_other:
106.     mov r0,#8           //2--other-->8
107.     ldr r1,=recv_state
108.     str r0,[r1]
109.     bl  USART2_IRQHandler_exit
110. //获取分钟
111. USART2_IRQHandler_state3:
112. //3--num-->4
113. //3--other-->8
114.     ldr r0,=received_data
115.     ldr r0,[r0]
116.     cmp r0,#'0'
117.     blo USART2_IRQHandler_state3_other
118.     cmp r0,#'9'
119.     bhi USART2_IRQHandler_state3_other
120.     sub r0,r0,#'0'
121.     ldr r1,=new_minute  //存放分钟数的十位数
122.     str r0,[r1]
123.     mov r0,#4           //3--num-->4
124.     ldr r1,=recv_state
125.     str r0,[r1]
126.     bl  USART2_IRQHandler_exit
127. USART2_IRQHandler_state3_other:
128.     mov r0,#8           //3--other-->8
129.     ldr r1,=recv_state
130.     str r0,[r1]
131.     bl  USART2_IRQHandler_exit
132. USART2_IRQHandler_state4:
133. //4--num-->5
```

```
//4--other-->8
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state4_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state4_other
    sub r0,r0,#'0'
    ldr r3,=new_minute  //存放完整的分钟数
    ldr r1,[r3]
    mov r2,#10
    mul r1,r2,r1
    add r0,r0,r1
    str r0,[r3]
    mov r0,#5           //4--num-->5
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state4_other:
    mov r0,#8           //4--other-->8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state5:
//5--:-->6
//5--other-->8
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#':'
    bne USART2_IRQHandler_state5_other
    //收到冒号
    mov r0,#6           //5--:-->6
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state5_other:
    mov r0,#8           //5--other-->8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
//获取秒数
USART2_IRQHandler_state6:
//6--num-->7
//6--other-->8
```

教务处制

```
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state6_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state6_other
    //收到字符 0-9
    sub r0,r0,#'0'
    ldr r1,=new_second  //秒数的十位数
    str r0,[r1]
    mov r0,#7           //6--num-->7
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state6_other:
    mov r0,#8           //6--other-->8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state7:
//7--num-->10
//7--other-->8
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state7_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state7_other
    //收到字符 0-9
    sub r0,r0,#'0'
    ldr r3,=new_second  //存放完整的秒数
    ldr r1,[r3]
    mov r2,#10
    mul r1,r2,r1
    add r0,r0,r1
    str r0,[r3]
    mov r0,#10          //7--num-->10
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state7_other:
    mov r0,#8           //7--other-->8
    ldr r1,=recv_state
    str r0,[r1]
```

```
226.     bl  USART2_IRQHandler_exit
227. USART2_IRQHandler_state8:
228. //8--$-->9
229. //8--other-->8
230.     //重置接收到的时间
231.     mov r0,#0
232.     ldr r1,=new_hour
233.     str r0,[r1]
234.     ldr r1,=new_minute
235.     str r0,[r1]
236.     ldr r1,=new_second
237.     str r0,[r1]
238.     ldr r0,=received_data
239.     ldr r0,[r0]
240.     cmp r0,#'$'
241.     bne USART2_IRQHandler_state8_other
242.     mov r0,#9           //8--$-->9
243.     ldr r1,=recv_state
244.     str r0,[r1]
245.     bl  USART2_IRQHandler_exit
246. USART2_IRQHandler_state8_other:
247.     mov r0,#8           //8--other-->8
248.     ldr r1,=recv_state
249.     str r0,[r1]
250.     bl  USART2_IRQHandler_exit
251. USART2_IRQHandler_state9:
252. //9--$-->0
253. //9--other-->8
254.     //处理收到的字节
255.     ldr r0,=received_data
256.     ldr r0,[r0]
257.     //收到$
258.     cmp r0,#'$'
259.     bne USART2_IRQHandler_state9_other
260.     mov r0,#0           //9--$-->0
261.     ldr r1,=recv_state
262.     str r0,[r1]
263.     bl  USART2_IRQHandler_exit
264. USART2_IRQHandler_state9_other:
265.     mov r0,#8           //9--other-->8
266.     ldr r1,=recv_state
267.     str r0,[r1]
268.     bl  USART2_IRQHandler_exit
269. USART2_IRQHandler_state10:
270.  //10--#-->11
271.  //10--other-->8
```

```
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'#'
    bne USART2_IRQHandler_state10_other
    mov r0,#11          //10--#-->11
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state10_other:
    mov r0,#8           //10--other-->8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state11:
//11--#--set_new_time-->8
//11--other-->8
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'#'//11--#--set_new_time-->8
    bne USART2_IRQHandler_state11_other
    //set_time:
    ldr r0,=new_hour
    ldr r0,[r0]
    //判断读取到的时间是否合法
    cmp r0,#23
    bhi USART2_IRQHandler_state11_other
    ldr r1,=new_minute
    ldr r1,[r1]
    cmp r1,#59
    bhi USART2_IRQHandler_state11_other
    ldr r2,=new_second
    ldr r2,[r2]
    cmp r2,#59
    bhi USART2_IRQHandler_state11_other
    //时间合法
    ldr r3,=hour
    str r0,[r3]
    ldr r3,=minute
    str r1,[r3]
    ldr r3,=second
    str r2,[r3]
    mov r0,#60
    ldr r1,=last_second
    str r0,[r1]
    ldr r1,=gcount
    mov r0,#0
```

```
    str r0,[r1]
USART2_IRQHandler_state11_other:
    mov r0,#8           //-->8
    ldr r1,=recv_state
    str r0,[r1]

USART2_IRQHandler_exit:
//exit，解除屏蔽中断，恢复现场
    cpsie   i
    add r7,#4
    mov sp,r7
    pop {r0-r7,pc}



//=====================================================================
//函数名称：SysTick_Handler
//功能概要：SysTick 定时器中断服务例程
//=====================================================================
SysTick_Handler:
    push {r0,r1,r2,lr}
    ldr r0,=gcount
    ldr r1,[r0]
    cmp r1,#50
    bcs SysTick_Handler_1s
    add r1,#1
    str r1,[r0]
    b   SysTick_Handler_Exit
SysTick_Handler_1s:
    mov r1,#0
    str r1,[r0]
    b   add_sec
SysTick_Handler_Exit:
    pop {r0,r1,r2,pc}

add_sec:
    ldr r0,=second
    ldr r1,[r0]
    cmp r1,#59
    bcs sec60
    add r1,#1
    str r1,[r0]
    b   add_sec_exit
sec60:
    mov r1,#0
    str r1,[r0]
```

```
364.     ldr r0,=minute
365.     ldr r1,[r0]
366.     cmp r1,#59
367.     bcs min60
368.     add r1,#1
369.     str r1,[r0]
370.     b   add_sec_exit
371. min60:
372.     mov r1,#0
373.     str r1,[r0]
374.     ldr r0,=hour
375.     ldr r1,[r0]
376.     cmp r1,#23
377.     bcs hour23
378.     add r1,#1
379.     str r1,[r0]
380.     b   add_sec_exit
381. hour23:
382.     mov r1,#0
383.     str r1,[r0]
384. add_sec_exit:
385.     b   SysTick_Handler_Exit
```

//main.s

```
1.  main:
2.  //（1）======启动部分（开头）主循环前的初始化工作============================
3.  //（1.1）声明 main 函数使用的局部变量
4.
5.  //（1.2）【不变】关总中断
6.      cpsid i
7.  //（1.3）给主函数使用的局部变量赋初值
8.  //（1.4）给全局变量赋初值
9.  //（1.5）用户外设模块初始化
10. //  使用 gpio_init 函数初始化 LED 灯
11.     ldr r0,=LIGHT_BLUE
12.     mov r1,#GPIO_OUTPUT
13.     mov r2,#LIGHT_ON
14.     bl  gpio_init
15. //  使用 systick_init 函数初始化定时器
16.     mov r0,#20
17.     bl systick_init
18. //  使用 uart_init 函数初始化串口 UARTA
19.     mov r0,#UARTA
20.     ldr r1,=UART_BAUD
21.     bl  uart_init
22. //（1.6）使用 uart_enable_re_int 函数使能模块中断
```

```
    mov r0,#UARTA
    bl  uart_enable_re_int
//（1.7）【不变】开总中断
    cpsie  i
//（1）======启动部分（结尾）=========================================
//（2）======主循环部分（开头）=========================================
main_loop:
        ldr r0,=second
        ldr r0,[r0]
        ldr r1,=last_second
        ldr r1,[r1]
        cmp r0,r1               //判断是否到达了新的一秒
        beq main_loop
        ldr r1,=last_second
        str r0,[r1]
        //到达了新的一秒：向串口发送数据
        ldr r2,=hour
        ldr r1,[r2]
        mov r0,#UARTA
        bl  uart_send1
        ldr r2,=minute
        ldr r1,[r2]
        mov r0,#UARTA
        bl  uart_send1
        ldr r2,=second
        ldr r1,[r2]
        mov r0,#UARTA
        bl  uart_send1
        ldr r2,[r2]
        ldr r1,=hour
        ldr r1,[r1]
        cmp r1,#10
        bcs hour_two_bits
        ldr r0,=time_show0
        bl  printf
hour_two_bits:
        ldr r0,=time_data_format
        ldr r1,=hour
        ldr r1,[r1]
        bl  printf            //输出 hour
        ldr r0,=time_show1
        bl  printf
        ldr r1,=minute
        ldr r1,[r1]
        cmp r1,#10
        //处理格式
```

```
69.          bcs minute_two_bits
70.          ldr r0,=time_show0
71.          bl  printf
72.
73.  minute_two_bits:
74.          ldr r0,=time_data_format
75.          ldr r1,=minute
76.          ldr r1,[r1]
77.          bl  printf              //输出 minute
78.          ldr r0,=time_show1
79.          bl  printf
80.          ldr r1,=second
81.          ldr r1,[r1]
82.          cmp r1,#10
83.          //处理格式
84.          bcs second_two_bits
85.          ldr r0,=time_show0
86.          bl  printf
87.
88.  second_two_bits:
89.          ldr r0,=time_data_format
90.          ldr r1,=second
91.          ldr r1,[r1]
92.          bl  printf             //输出 second
93.          ldr r0,=time_show2
94.          bl  printf
95.  //（2.3.2）如灯状态标志 mFlag 为'L'，灯的闪烁次数+1 并显示，改变灯状态及标志
96.          //判断灯的状态标志
97.          ldr r2,=mFlag
98.          ldr r6,[r2]
99.          cmp r6,#'L'
100.         bne main_light_off
101.         ldr r3,=mLightCount
102.         ldr r1,[r3]
103.         add r1,#1
104.         str r1,[r3]
105.         ldr r0,=light_show3
106.         bl  printf
107.         ldr r0,=data_format
108.         ldr r2,=mLightCount
109.         ldr r1,[r2]
110.         bl  printf
111.         ldr r2,=mFlag
112.         mov r7,#'A'
113.         str r7,[r2]
114.         ldr r0,=LIGHT_BLUE
```

```
115.        ldr r1,=LIGHT_ON
116.        bl  gpio_set
117.        ldr r0, =light_show1
118.        bl  printf
119.        b main_exit
120. //（2.3.3）如灯状态标志 mFlag 为'A'，改变灯状态及标志
121. main_light_off:
122.        ldr r2,=mFlag          //灯的状态标志改为'L'
123.        mov r7,#'L'
124.        str r7,[r2]
125.        ldr r0,=LIGHT_BLUE      //暗灯
126.        ldr r1,=LIGHT_OFF
127.        bl  gpio_set
128.        ldr r0, =light_show2    //显示灯暗提示
129.        bl  printf
130. main_exit:
131.    b main_loop                 //继续循环
132. //（2）======主循环部分（结尾）=======================================
133. .end         //整个程序结束标志（结尾）
```

五. 实践性问答题

（1）不用其他工具，如何测试发送一个字符的真实时间？

可以通过串口发送一个较长的数据，记录下发送数据的首位和末位字符的时间差，用数据的字符数除以记录下的时间差即可得到发送一个字符的真实时间，同时还可以利用统计学技巧，例如多次取平均等来减小误差。

（2）请简述定时器中断的基本原理，定时器是如何实现计时的？

Arm Cortex-M4F 内核中内置了一个名为 SysTick 的简单定时器，常被称为 "滴答" 定时器。SysTick 定时器作为一个功能模块，嵌入在 NVIC（嵌套向量中断控制器）中。它具有 24 位的有效位数，并采用减 1 计数的工作方式。当减 1 计数器达到 0 时，SysTick 定时器会触发 SysTick 中断，中断号为 15。

在实现计时的过程中，可以根据所需的定时时间使用指令来设置定时器的计时常数，并使用指令启动定时器开始计数。当计数器达到指定值时，定时器会自动产生一个定时输出信号或中断信号，通知 CPU。一旦定时器开始工作，CPU 就不必关注它，而可以继续执行其他任务。这种方式允许 CPU 在定时器运行的同时执行并行任务，提高系统的效率和性能。定时器的自动计数和中断功能使得计时操作可以在后台进行，而无需 CPU 持续监控和干预。

（3）Timer 中断最小定时时间是多少？

Timer 的最小定时时间取决于时钟频率和 Timer 的配置。最小定时时间由以下公式确定：

最小定时时间 = (Timer 的分频系数 + 1) × Timer 的重装载值 / Timer 的时钟频率

其中，Timer 的分频系数表示时钟预分频的倍数，Timer 的重装载值表示计数器计数到多少时重新加载，Timer 的时钟频率表示 Timer 的输入时钟频率。

Arm Cortex-M4F 的主频为 72M，计算结果为 0.06us，指令的执行与程序的跳转同样需要大约 0.1us 的时间，所以常使用 1us 作为最小定时时间。

教务处制

教务处制