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	张昊	学号	1927405160
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	2022 年 4 月 26 日	

实验名称：实验一：理解汇编程序框架及运行

一. 实验目的

本实验通过编程控制 LED 小灯，体会 GPIO 输出作用，可扩展控制蜂鸣器、继电器等；通过编程获取引脚状态，体会 GPIO 输入作用，可用于获取开关的状态。主要目的如下：

（1）了解集成开发环境的安装与基本使用方法。

（2）掌握 GPIO 构件基本应用方法，理解第一个汇编程序框架结构。

（3）掌握硬件系统的软件测试方法，初步理解 printf 输出调试的基本方法。

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

在验证性实验的基础上，自行编程实现开发板上的红灯、蓝灯和绿灯交替闪烁。LED 三色灯电路原理如图 4-8 所示，对应三个控制端接 MCU 的三个 GPIO 引脚，在本书采用的 STM32L4 芯片上，红灯接 PTB.7 引脚、绿灯接 PTB.8 引脚、蓝灯接 PTB.9 引脚。可以通过程序，测试你使用的开发套件中的发光二极管是否与图中接法一致。

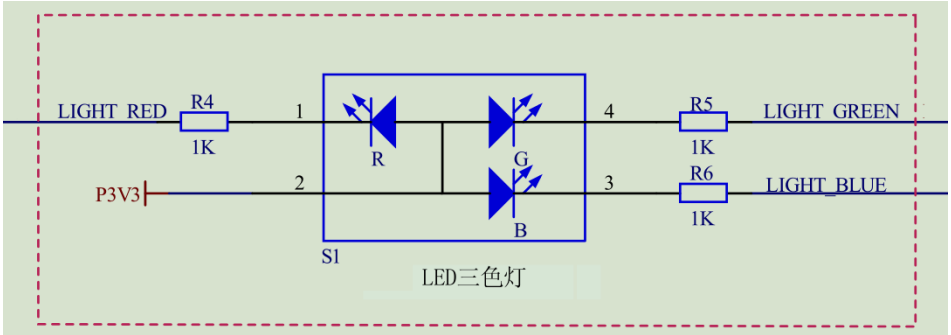

图 4-8 LED 三色灯电路原理图

（3）进阶实验★

对目标板上的三色灯进行编程，通过三色灯的不同组合，实现红、蓝、绿、青、紫、黄和白等灯的亮暗控制。灯颜色提示：青色为绿蓝混合，黄色为红绿混合，紫色为红蓝混合，白色为红蓝绿混合。

四、实验结果

（1）验证性实验

请问实验结果与实验过程描述是否一致？在实验过程中是否遇到过问题？如何解决的？

实验结果与实验过程描述一致。串口更新截图：

（2）设计性实验

```
//验证红灯的实验代码
//主函数开始时先初始化红灯，r0、r1、r2是gpio_init的入口参数（使用寄存器传递）
ldr r0,=LIGHT_RED      //r0指明端口和引脚(端口号|(引脚号))
```

```
mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
mov r2,#LIGHT_OFF      //r2指明引脚的初始状态为暗
bl  gpio_init          //调用gpio初始化函数（gpio.s）
//亮红灯代码
ldr r0,=LIGHT_RED      //亮红灯
ldr r1,=LIGHT_ON
bl  gpio_set
ldr r0, =light_show_red_on    //显示红灯亮提示
bl  printf
//暗红灯代码
ldr r0,=LIGHT_RED      //灭红灯
ldr r1,=LIGHT_OFF
bl  gpio_set
ldr r0, =light_show_off    //显示灯灭提示
bl  printf
```

```
//验证蓝灯的实验代码
//主函数开始时先初始化蓝灯，r0、r1、r2是gpio_init的入口参数（使用寄存器传递）
ldr r0,=LIGHT_BLUE     //r0指明端口和引脚(端口号|(引脚号))
mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
mov r2,#LIGHT_OFF      //r2指明引脚的初始状态为暗
bl  gpio_init          //调用gpio初始化函数（gpio.s）
//亮蓝灯代码
ldr r0,=LIGHT_BLUE      //亮蓝灯
ldr r1,=LIGHT_ON
bl  gpio_set
ldr r0, =light_show_blue_on    //显示红灯亮提示
bl  printf
//暗蓝灯代码
ldr r0,=LIGHT_BLUE      //灭蓝灯
ldr r1,=LIGHT_OFF
bl  gpio_set
ldr r0, =light_show_off    //显示灯灭提示
bl  printf
```

```
//验证绿灯的实验代码
//主函数开始时先初始化绿灯，r0、r1、r2是gpio_init的入口参数（使用寄存器传递）
ldr r0,=LIGHT_GREEN     //r0指明端口和引脚(端口号|(引脚号))
mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
mov r2,#LIGHT_OFF      //r2指明引脚的初始状态为暗
bl  gpio_init          //调用gpio初始化函数（gpio.s）
//亮绿灯代码
ldr r0,=LIGHT_GREEN      //亮绿灯
ldr r1,=LIGHT_ON
bl  gpio_set
ldr r0, =light_show_green_on    //显示红灯亮提示
bl  printf
```

```
//暗绿灯代码
ldr r0,=LIGHT_GREEN      //灭绿灯
ldr r1,=LIGHT_OFF
bl  gpio_set
ldr r0, =light_show_off    //显示灯灭提示
bl  printf
```

（3）进阶实验★

使用 mLightType 作为亮灯的类型，0 为红色，1 为绿色，2 为蓝色，3 为黄色（红色+绿色），4 为紫色（红色+蓝色），5 为青色（蓝色+绿色），6 为白色（红色+蓝色+绿色）。每次亮灯后对其加 1，当大于等于 7 时变为 1。

```
//实验各色灯的实验代码
.include "include.inc"    //头文件中主要定义了程序中需要使用到的一些常量
//（0）数据段与代码段的定义
//（0.1）定义数据存储data段开始，实际数据存储在RAM中
.section .data
//（0.1.1）定义需要输出的字符串，标号即为字符串首地址,\0为字符串结束标志
hello_information:          //字符串标号
    .ascii "-----------------------------------------------------\n"
    .ascii " 用汇编点亮的三色发光二极管。\n"
    .ascii "-----------------------------------------------------\n\0"
light_show_red_on:
    .asciz "LIGHT_RED:ON--\n"    //红灯亮状态提示
light_show_green_on:
    .asciz "LIGHT_GREEN:ON--\n"    //绿灯亮状态提示
light_show_blue_on:
    .asciz "LIGHT_BLUE:ON--\n"    //蓝灯亮状态提示
light_show_off:
    .asciz "LIGHT:OFF--\n"     //灯暗状态提示
light_show_count:
    .asciz "闪烁次数mLightCount=%d\n"  //闪烁次数提示
light_show_type:
    .asciz "闪烁类型mLightType=%d\n"  //闪烁次数提示
//（0.1.2）定义变量
.align 4              //.word格式四字节对齐
mMainLoopCount:          //定义主循环次数变量，数据格式为字，初始值为0
    .word 0
mFlag:                //定义灯的状态标志，'L'为亮，'A'为暗初始值为亮
    .byte 'L'
.align 4
mLightCount:       //定义灯的闪烁次数，数据格式为字，初始值为0
    .word 0
.align 4
mLightType:           //定义灯的类型，数据格式为字，初始值为0，大于7时重置为0
    .word 0
```

```
.equ MainLoopNUM,6122338  //主循环次数设定值（常量）

//（0.2）定义代码存储text段开始，实际代码存储在Flash中
.section   .text
.syntax unified        //指示下方指令为ARM和thumb通用格式
.thumb                 //Thumb指令集
.type main function    //声明main为函数类型
.global main           //将main定义成全局函数，便于芯片初始化之后调用
.align 2               //指令和数据采用2字节对齐，兼容Thumb指令集

//-------------------------------------------------------------------
//main.c使用的内部函数声明处
//-------------------------------------------------------------------
main:        //主函数
//（1）======启动部分（开头）主循环前的初始化工作======================
//（1.1）【不变】关总中断
    cpsid i
//（1.2）用户外设模块初始化
    //初始化红灯，r0、r1、r2是gpio_init的入口参数（使用寄存器传递）
    ldr r0,=LIGHT_RED     //r0指明端口和引脚(端口号|(引脚号))
    mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
    mov r2,#LIGHT_OFF      //r2指明引脚的初始状态为暗
    bl  gpio_init          //调用gpio初始化函数（gpio.s）
    //初始化绿灯，r0、r1、r2是gpio_init的入口参数（使用寄存器传递）
    ldr r0,=LIGHT_GREEN     //r0指明端口和引脚(端口号|(引脚号))
    mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
    mov r2,#LIGHT_OFF      //r2指明引脚的初始状态为暗
    bl  gpio_init          //调用gpio初始化函数（gpio.s）
    //初始化蓝灯，r0、r1、r2是gpio_init的入口参数（使用寄存器传递）
    ldr r0,=LIGHT_BLUE     //r0指明端口和引脚(端口号|(引脚号))
    mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
    mov r2,#LIGHT_OFF      //r2指明引脚的初始状态为暗
    bl  gpio_init          //调用gpio初始化函数（gpio.s）
//（1.3）【不变】开总中断（初始化结束）
    cpsie  i
    //显示hello_information定义的字符串
    ldr r0,=hello_information   //待显示字符串首地址
    bl  printf                 //调用printf显示字符串
//（1）======启动部分（结尾）=======================================
//（2）======主循环部分（开头）=======================================
    main_loop:                      //主循环标签（开头）
//（2.1）主循环次数变量mMainLoopCount+1
        ldr r2,=mMainLoopCount   //r2←mMainLoopCount的地址
        ldr r1, [r2]            //将r2保存的地址中存储的值存到r1中
        add r1,#1               //r1=r1+1
```


教务处制

```
        str r1,[r2]                 //将r1中的值存到r2保存的地址中去
//（2.2）未达到主循环次数设定值，继续循环
        ldr r2,=MainLoopNUM    //从MainLoopNUM所在的地址中取数据至r2
        cmp r1,r2              //比较r1和r2
        blO  main_loop        //未达到，继续循环（lo：无符号数小于）
        //bne  main_loop      //未达到，继续循环（ne：不等于）
//（2.3）达到主循环次数设定值，执行下列语句，进行灯的亮暗处理
//（2.3.1）清除循环次数变量 mMainLoopCount=0
        ldr r2,=mMainLoopCount     //r2←mMainLoopCount的地址
        mov r1,#0                  //若达到循环次数，将循环变量的值清零
        str r1,[r2]                  //将r1中的值存到r2所在的内存地址中
//（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，根据灯的类型亮相应的灯，修改灯的类型
        //判断灯的状态标志
        ldr r2,=mFlag
        ldr r6,[r2]
        cmp r6,#'L'
        bne main_light_off    //mFlag不等于'L'转关灯
        //mFlag等于'L'情况：开灯
        ldr r3,=mLightCount      //灯的闪烁次数mLightCount+1
        ldr r1,[r3]
        add r1,#1
        str r1,[r3]
        ldr r0,=light_show_count  //显示灯的闪烁次数值
        ldr r2,=mLightCount
        ldr r1,[r2]
        bl  printf
        ldr r2,=mFlag           //灯的状态标志改为'A'（下一步是灭灯）
        mov r7,#'A'
        str r7,[r2]
        //根据mLightType亮不同颜色的灯
        ldr r0,=light_show_type  //显示灯的类型mLightType
        ldr r2,=mLightType
        ldr r1,[r2]
        bl  printf
        ldr r2,=mLightType    //读取灯的类型mLightType
        ldr r1,[r2]
        cmp r1,#0
        bne case_type_1           //不等于0转判断1
        //等于0的情况:红灯
        ldr r0,=LIGHT_RED      //亮红灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show_red_on    //显示红灯亮提示
        bl  printf
```

```
    b end_case_type           //break
case_type_1:
    ldr r2,=mLightType    //读取灯的类型mLightType
    ldr r1,[r2]
    cmp r1,#1
    bne case_type_2           //不等于1转判断2
    //等于1的情况:绿灯
    ldr r0,=LIGHT_GREEN      //亮绿灯
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r0, =light_show_green_on    //显示绿灯亮提示
    bl  printf
    b end_case_type           //break
case_type_2:
    ldr r2,=mLightType    //读取灯的类型mLightType
    ldr r1,[r2]
    cmp r1,#2
    bne case_type_3           //不等于2转判断3
    //等于2的情况:蓝灯
    ldr r0,=LIGHT_BLUE      //亮蓝灯
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r0, =light_show_blue_on    //显示蓝灯亮提示
    bl  printf
    b end_case_type           //break
case_type_3:
    ldr r2,=mLightType    //读取灯的类型mLightType
    ldr r1,[r2]
    cmp r1,#3
    bne case_type_4           //不等于3转判断4
    //等于3的情况:红灯+绿灯=黄
    ldr r0,=LIGHT_RED      //亮红灯
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r0, =light_show_red_on    //显示红灯亮提示
    bl  printf
    ldr r0,=LIGHT_GREEN      //亮绿灯
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r0, =light_show_green_on    //显示绿灯亮提示
    bl  printf
    b end_case_type           //break
case_type_4:
    ldr r2,=mLightType    //读取灯的类型mLightType
    ldr r1,[r2]
```

```
        cmp r1,#4
        bne case_type_5           //不等于4转判断5
        //等于4的情况:红灯+蓝灯=紫
        ldr r0,=LIGHT_RED      //亮红灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show_red_on    //显示红灯亮提示
        bl  printf
        ldr r0,=LIGHT_BLUE      //亮蓝灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show_blue_on    //显示蓝灯亮提示
        bl  printf
        b end_case_type           //break
    case_type_5:
        ldr r2,=mLightType    //读取灯的类型mLightType
        ldr r1,[r2]
        cmp r1,#5
        bne case_type_6           //不等于5转6
        //等于5的情况:绿灯+蓝灯=青
        ldr r0,=LIGHT_GREEN      //亮绿灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show_green_on    //显示绿灯亮提示
        bl  printf
        ldr r0,=LIGHT_BLUE      //亮蓝灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show_blue_on    //显示蓝灯亮提示
        bl  printf
        b end_case_type           //break
    case_type_6:
        //等于6的情况:红灯+绿灯+蓝灯=白
        ldr r0,=LIGHT_RED      //亮红灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show_red_on    //显示红灯亮提示
        bl  printf
        ldr r0,=LIGHT_GREEN      //亮绿灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show_green_on    //显示绿灯亮提示
        bl  printf
        ldr r0,=LIGHT_BLUE      //亮蓝灯
        ldr r1,=LIGHT_ON
```

```
        bl  gpio_set
        ldr r0, =light_show_blue_on    //显示蓝灯亮提示
        bl  printf
    end_case_type:
        //修改灯的类型mLightType+1，若mLightType>=7，mLightType=0
        ldr r3,=mLightType    //灯的类型mLightType
        ldr r1,[r3]
        add r1,#1             //mLightType++
        cmp r1,#7
        blo type_not_to_zero    //若mLightType>=7，mLightType=0
        mov r1,#0
    type_not_to_zero:                 //若mLightType<7，mLightType++后不改变
        str r1,[r3]
        //mFlag等于'L'情况处理完毕，转
        b main_exit
//（2.3.3）如灯状态标志mFlag为'A'，改变灯状态及标志，灭所有的灯
    main_light_off:
        ldr r2,=mFlag        //灯的状态标志改为'L'（下一步是亮灯）
        mov r7,#'L'
        str r7,[r2]
        ldr r0,=LIGHT_RED      //灭红灯
        ldr r1,=LIGHT_OFF
        bl  gpio_set
        ldr r0,=LIGHT_GREEN      //灭绿灯
        ldr r1,=LIGHT_OFF
        bl  gpio_set
        ldr r0,=LIGHT_BLUE      //灭蓝灯
        ldr r1,=LIGHT_OFF
        bl  gpio_set
        ldr r0, =light_show_off    //显示灯灭提示
        bl  printf
    main_exit:
        b main_loop                //继续循环
//（2）======主循环部分（结尾）=====================================
.end     //整个程序结束标志（结尾）
```

五. 实践性问答题

（1）比较 ascii、asciz、string 这三种字符串定义格式的区别。

- .ascii：这种方式定义的字符串不会自动在末尾添加“\0”，因此为了让字符串结束必须手动添加“\0”， 常用于一次性输出多行字符串的场合；
- .asciz 或.string：这种方式定义的字符串会自动在末尾添加“\0”，一般用于输入一行字符串的情况。

（2）比较立即数的“#”和“=”这两个前缀的区别

使用常量时，当常量小于等于 256 时，使用前缀“#”，当常量大于 256 时，使用前缀“=”。如：

```
ldr r0,=LIGHT_RED     // LIGHT_RED>256
```

```
   mov r1,#GPIO_OUTPUT    // GPIO_OUTPUT=1
```

（3）编写程序输出参考样例中mMainLoopCount变量的地址。

```
   LDR R1, =mMainLoopCount
   LDR R0, =data_format
   BL printf
```