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		2023.04.18	

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。

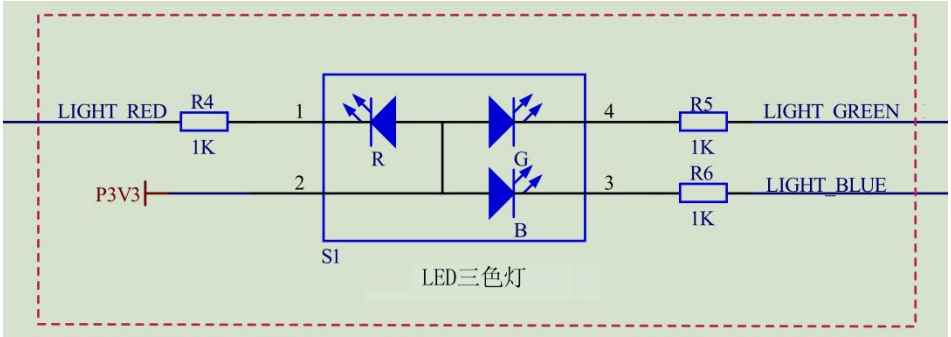

图 4-8 LED 三色灯电路原理图

（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，PC 界面显示结果如下：

教务处制

（2）设计性实验

//验证红灯的实验代码

```
1.  //宏定义
2.  .equ LIGHT_RED,(PTB_NUM|7)  //红色 RUN 灯使用的端口/引脚
3.  .equ LIGHT_ON,0  //灯亮
4.  .equ LIGHT_OFF,1  //灯暗
5.  //使用 gpio_init 初始化
6.  ldr r0,=LIGHT_RED  //r0<-端口和引脚（用=是因为宏常数>=256，且用 ldr）
7.  mov r2,#GPIO_OUTPUT  //r2<-引脚的初始状态为输出
8.  mov r3,#LIGHT_ON  //r3<-引脚的默认状态，默认点亮
9.  bl gpio_init  //调用 gpio 初始化函数
10. //点亮红灯
11. ldr r0,=LIGHT_RED
12. ldr r1,=LIGHT_ON
13. bl gpio_set  //调用函数设置小灯为亮
14. //熄灭红灯
15. ldr r0,=LIGHT_RED
16. ldr r1,=LIGHT_OFF
17. bl gpio_set  //调用函数设置小灯为亮
```

//验证蓝灯的实验代码

```
1.  //宏定义
2.  .equ LIGHT_BLUE,(PTB_NUM|9)  //蓝色 RUN 灯使用的端口/引脚
3.  .equ LIGHT_ON,0  //灯亮
4.  .equ LIGHT_OFF,1  //灯暗
5.  //使用 gpio_init 初始化
6.  ldr r0,=LIGHT_BLUE  //r0<-端口和引脚（用=是因为宏常数>=256，且用 ldr）
7.  mov r2,#GPIO_OUTPUT  //r2<-引脚的初始状态为输出
8.  mov r3,#LIGHT_ON  //r3<-引脚的默认状态，默认点亮
9.  bl gpio_init  //调用 gpio 初始化函数
10. //点亮蓝灯
11. ldr r0,=LIGHT_BLUE
12. ldr r1,=LIGHT_ON
13. bl gpio_set  //调用函数设置小灯为亮
14. //熄灭蓝灯
15. ldr r0,=LIGHT_BLUE
16. ldr r1,=LIGHT_OFF
17. bl gpio_set  //调用函数设置小灯为亮
```

//验证绿灯的实验代码

```
1.  //宏定义
2.  .equ LIGHT_GREEN,(PTB_NUM|8)  //绿色 RUN 灯使用的端口/引脚
3.  .equ LIGHT_ON,0  //灯亮
```

```
.equ LIGHT_OFF,1  //灯暗
//使用 gpio_init 初始化
ldr r0,=LIGHT_GREEN  //r0<-端口和引脚（用=是因为宏常数>=256，且用 ldr）
mov r2,#GPIO_OUTPUT  //r2<-引脚的初始状态为输出
mov r3,#LIGHT_ON  //r3<-引脚的默认状态，默认点亮
bl gpio_init  //调用 gpio 初始化函数
//点亮绿灯
ldr r0,=LIGHT_GREEN
ldr r1,=LIGHT_ON
bl gpio_set  //调用函数设置小灯为亮
//熄灭绿灯
ldr r0,=LIGHT_GREEN
ldr r1,=LIGHT_OFF
bl gpio_set  //调用函数设置小灯为亮
```

（3）进阶实验★

定义一个枚举变量，用于判断当前应该亮何种颜色的灯，使它在 0123456 中依次循环，其中，0123456 分别代表“红绿蓝黄紫青白”色的灯。

//实验各色灯的实验代码

```
    .include "include.inc" //总头文件，只包含 user.inc
//（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 " 使用 GPIO 点亮七种颜色的二极管\n"
    .ascii "------------------------------------------------------\n\0"
light_off:
    .asciz "LIGHT:OFF--\n" //灯暗状态提示
light_red_on:
    .asciz "LIGHT_RED:ON--\n" //红灯亮状态提示
light_green_on:
    .asciz "LIGHT_GREEN:ON--\n" //绿灯亮状态提示
light_blue_on:
    .asciz "LIGHT_BLUE:ON--\n" //蓝灯亮状态提示
light_count:
    .asciz "灯的闪烁次数 mLightCount=%d\n" //闪烁次数提示
light_type:
    .asciz "灯的闪烁类型 mLightType=%d\n" //闪烁类型提示
//（0.1.2）定义变量
    .align 4 //.word 格式 4 字节对齐
    mMainLoopCount: //定义主循环次数变量
```

教务处制

```
25.      .word 0
26.      mFlag: //定义灯的状态标志
27.      .byte 'L'
28.      .align 4
29.      mLightCount: //定义灯的闪烁次数
30.      .word 0
31.      .align 4
32.      mLightType: //定义灯的类型,数据格式为字,初始值为 0,按照 0123456 顺序循环
33.      .word 0
34.      .equ MainLoopNUM,5566770 //主循环次数设定值（常量）
35.  //（0.2）定义代码存储 text 段开始，实际代码存储在 Flash 中
36.      .section .text
37.      .syntax unified      //指示下方指令为 ARM 和 thumb 通用格式
38.      .thumb               //Thumb 指令集
39.      .type main function //声明 main 为函数类型
40.      .global main         //将 main 定义成全局函数，便于芯片初始化之后调用
41.      .align 2             //指令和数据采用 2 字节对齐，兼容 Thumb 指令集
42.  //--------------------------------------------------------------------
43.  //main.c 使用的内部函数声明处
44.  //--------------------------------------------------------------------
45.      main: //主函数
46.  //（1）======启动部分（开头）主循环前的初始化工作======================
47.  //（1.1）声明 main 函数使用的局部变量
48.  //（1.2）【不变】关总中断
49.      cpsid i
50.  //（1.3）给主函数使用的局部变量赋初值
51.  //（1.4）给全局变量赋初值
52.  //（1.5）用户外设模块初始化
53.  //初始化红灯, r0、r1、r2 是 gpio_init 的入口参数
54.      ldr r0,=LIGHT_RED //r0<-端口和引脚
55.      mov r1,#GPIO_OUTPUT //r1<-引脚方向为输出
56.      mov r2,#LIGHT_OFF //r2<-引脚的初始状态为暗
57.      bl gpio_init //调用 gpio 初始化函数
58.  //初始化绿灯, r0、r1、r2 是 gpio_init 的入口参数
59.      ldr r0,=LIGHT_GREEN //r0<-端口和引脚
60.      mov r1,#GPIO_OUTPUT //r1<-引脚方向为输出
61.      mov r2,#LIGHT_OFF //r2<-引脚的初始状态为暗
62.      bl gpio_init //调用 gpio 初始化函数
63.  //初始化蓝灯, r0、r1、r2 是 gpio_init 的入口参数
64.      ldr r0,=LIGHT_BLUE //r0<-端口和引脚
65.      mov r1,#GPIO_OUTPUT //r1<-引脚方向为输出
66.      mov r2,#LIGHT_OFF //r2<-引脚的初始状态为暗
67.      bl gpio_init //调用 gpio 初始化函数
68.  //（1.6）使能模块中断
```

教务处制

```
69.  //（1.7）【不变】开总中断（初始化结束）
70.      cpsie i
71.      ldr r0,=hello_information //待显示字符串首地址
72.      bl printf //调用 printf 显示字符串
73.      //bl . //在此打桩
74.  //（1）======启动部分（结尾）========================================
75.  //（2）======主循环部分（开头）========================================
76.      main_loop: //主循环标签（开头）
77.  //（2.1）主循环次数变量 mMainLoopCount+1
78.      ldr r2,=mMainLoopCount //r2←mMainLoopCount 的地址
79.      ldr r1, [r2]
80.      add r1,#1
81.      str r1,[r2]
82.  //（2.2）未达到主循环次数设定值，继续循环
83.      ldr r2,=MainLoopNUM
84.      cmp r1,r2
85.      blO main_loop //未达到，继续循环
86.  //（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
87.  //（2.3.1）清除循环次数变量
88.      ldr r2,=mMainLoopCount //r2←mMainLoopCount 的地址
89.      mov r1,#0
90.      str r1,[r2]
91.  //（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
92.  //判断灯的状态标志
93.      ldr r2,=mFlag
94.      ldr r6,[r2]
95.      cmp r6,#'L'
96.      bne main_light_off //mFlag 不等于'L'，这时需要关灯
97.  //mFlag 等于'L'情况，这时需要开灯
98.      ldr r3,=mLightCount //灯的闪烁次数 mLightCount+1
99.      ldr r1,[r3]
100.     add r1,#1
101.     str r1,[r3]
102.     ldr r0,=light_count //显示灯的闪烁次数值
103.     ldr r2,=mLightCount
104.     ldr r1,[r2]
105.     bl printf
106.     ldr r2,=mFlag //灯的状态标志改为'A'（下一步是灭灯）
107.     mov r7,#'A'
108.     str r7,[r2]
109. //输出此时的灯的颜色类型
110.     ldr r0,=light_type //显示灯的类型 mLightType
111.     ldr r2,=mLightType
112.     ldr r1,[r2]
```

教务处制

```
    bl printf
//依次判断 light_type 枚举到哪一种灯的颜色，亮相应的灯
    ldr r2,=mLightType
    ldr r1,[r2]  //r1<-mLightType
    cmp r1,#0
    bne light_type_1 //不等于 0,跳转判断 1
//light_type=0:red
    ldr r0,=LIGHT_RED //亮红灯
    ldr r1,=LIGHT_ON
    bl gpio_set
    ldr r0, =light_red_on //显示红灯亮提示
    bl printf
    b end_light_type
//light_type=1:green
light_type_1:
    ldr r2,=mLightType
    ldr r1,[r2] //r1<-mLightType
    cmp r1,#1
    bne light_type_2 //不等于 1,跳转判断 2
    ldr r0,=LIGHT_GREEN //亮绿灯
    ldr r1,=LIGHT_ON
    bl gpio_set
    ldr r0, =light_green_on //显示绿灯亮提示
    bl printf
    b end_light_type //break
//light_type=2:blue
light_type_2:
    ldr r2,=mLightType
    ldr r1,[r2] //r1<-mLightType
    cmp r1,#2
    bne light_type_3 //不等于 2,跳转判断 3
    ldr r0,=LIGHT_BLUE //亮蓝灯
    ldr r1,=LIGHT_ON
    bl gpio_set
    ldr r0, =light_blue_on //显示蓝灯亮提示
    bl printf
    b end_light_type //break
//light_type=3:yellow=red+green
light_type_3:
    ldr r2,=mLightType
    ldr r1,[r2] //r1<-mLightType
    cmp r1,#3
    bne light_type_4 //不等于 3,跳转判断 4
    ldr r0,=LIGHT_RED //亮红灯
```

教务处制

```
157.     ldr r1,=LIGHT_ON
158.     bl gpio_set
159.     ldr r0, =light_red_on //显示红灯亮提示
160.     bl printf
161.     ldr r0,=LIGHT_GREEN //亮绿灯
162.     ldr r1,=LIGHT_ON
163.     bl gpio_set
164.     ldr r0, =light_green_on //显示绿灯亮提示
165.     bl printf
166.     b end_light_type //break
167. //light_type=4:purple=red+blue
168. light_type_4:
169.     ldr r2,=mLightType
170.     ldr r1,[r2] //r1<-mLightType
171.     cmp r1,#4
172.     bne light_type_5 //不等于 4,跳转判断 5
173.     ldr r0,=LIGHT_RED //亮红灯
174.     ldr r1,=LIGHT_ON
175.     bl gpio_set
176.     ldr r0, =light_red_on //显示红灯亮提示
177.     bl printf
178.     ldr r0,=LIGHT_BLUE //亮蓝灯
179.     ldr r1,=LIGHT_ON
180.     bl gpio_set
181.     ldr r0, =light_blue_on //显示蓝灯亮提示
182.     bl printf
183.     b end_light_type //break
184. //light_type=5:cyan=green+blue
185. light_type_5:
186.     ldr r2,=mLightType
187.     ldr r1,[r2] //r1<-mLightType
188.     cmp r1,#5
189.     bne light_type_6 //不等于 5,跳转 6
190.     ldr r0,=LIGHT_GREEN //亮绿灯
191.     ldr r1,=LIGHT_ON
192.     bl gpio_set
193.     ldr r0, =light_green_on //显示绿灯亮提示
194.     bl printf
195.     ldr r0,=LIGHT_BLUE //亮蓝灯
196.     ldr r1,=LIGHT_ON
197.     bl gpio_set
198.     ldr r0, =light_blue_on //显示蓝灯亮提示
199.     bl printf
200.     b end_light_type //break
```

教务处制

```
//light_type=6:white=red+green+blue
light_type_6:
    ldr r0,=LIGHT_RED //亮红灯
    ldr r1,=LIGHT_ON
    bl gpio_set
    ldr r0, =light_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_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_blue_on //显示蓝灯亮提示
    bl printf
end_light_type:
//mLightType+1，让它在 0123456 循环，若 mLightType>=7，mLightType=0
    ldr r3,=mLightType
    ldr r1,[r3] //r1<-mLightType
    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'
    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_off //显示提示
```

教务处制

```
    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：使用 ASCII 码来定义字符串，不会在字符串的结尾自动添加空字符”\0”

ASCIZ：使用 ASCII 码来定义字符串，会在字符串的结尾自动添加空字符”\0”

STRING：使用的是字符形式来定义字符串，会在字符串的结尾添加空字符”\0”

（2）比较立即数的“#”和“=”这两个前缀的区别

"#"和"="是两种不同的立即数前缀。

"#"前缀表示一个立即数值。立即数是一个固定的数值，可以直接使用，而不需要在程序运行时计算。当常量小于等于 256 时，使用前缀“#”。

"="前缀表示一个立即数的标签，通常用于定义常量或全局变量。该标签指向一个位置，位置中存储的是实际的数值。在使用这个标签时，汇编器会将其替换为一个立即数值。当常量大于 256 时，使用前缀“=”。

（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
```

教务处制