实验一　STM32 微处理器指令与程序结构

【实验目的】

1、 熟悉 KEIL 开发环境；

2、 掌握 STM32 微处理器指令与程序结构；

3、 熟悉 STM32 程序调试方法。

【实验内容 1】项目建立与调试方法。

程序要求：计算一个数据的绝对值。

程序名：LX1

程序清单：

```
                AREA    LXX, CODE, READONLY     ;定义程序段
                EXPORT  __Vectors               ;矢量表头,供链接
__Vectors       DCD     0x20000400              ;栈顶=0x20000400
                DCD   Reset_Handler             ;复位中断程序首地址
                                                ;矢量表结束
Reset_Handler
                EXPORT  Reset_Handler           ;供链接
LOOP                                            ;循环运行
                LDR     R0, =TBL                ;取数据表头
                LDR     R1, [R0]                ;取数据
                MOVS    R0, R1                  ;根据 PSR 判别正负
                RSBMI   R0, R0, #0              ;负则取补
                B       LOOP                    ;循环
                ALIGN                           ;数据对齐
TBL
                DCD     -10                     ;数据表
                END                             ;停止编译
```

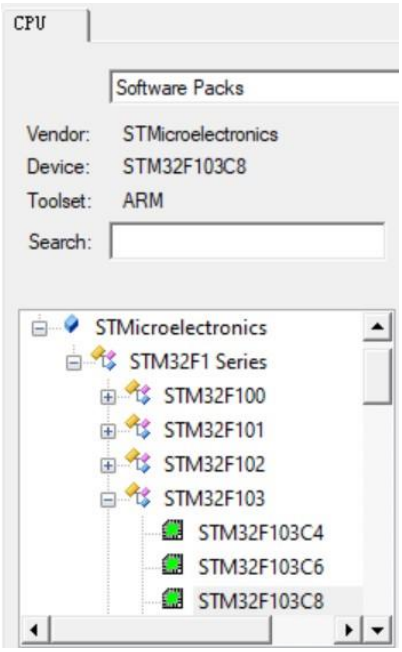
1） 新建一个文件夹，建议用学号做文件夹名；

2） 打开 KEIL 软件；

3） 新建一个项目：点击 Project—New uvision project；

4） 给新建项目命名；

5） 选择 CPU：STM32F103C8



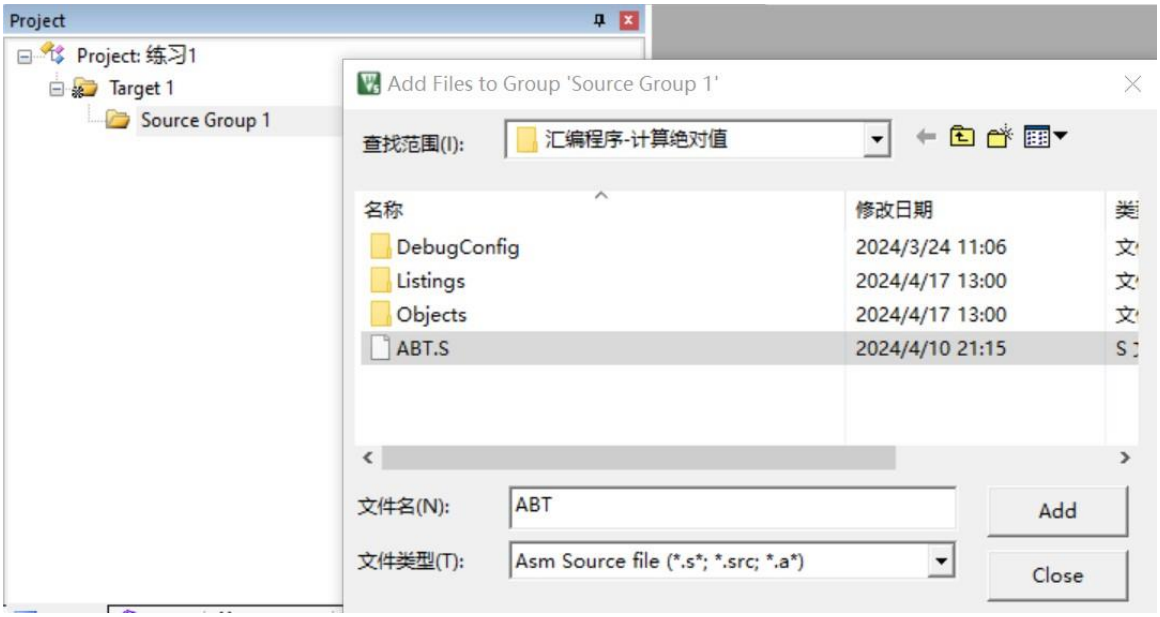
6） 在跳出的环境配置界面，点击取消（不使用标准文件）

7） 新建一段程序：点击 File—New；

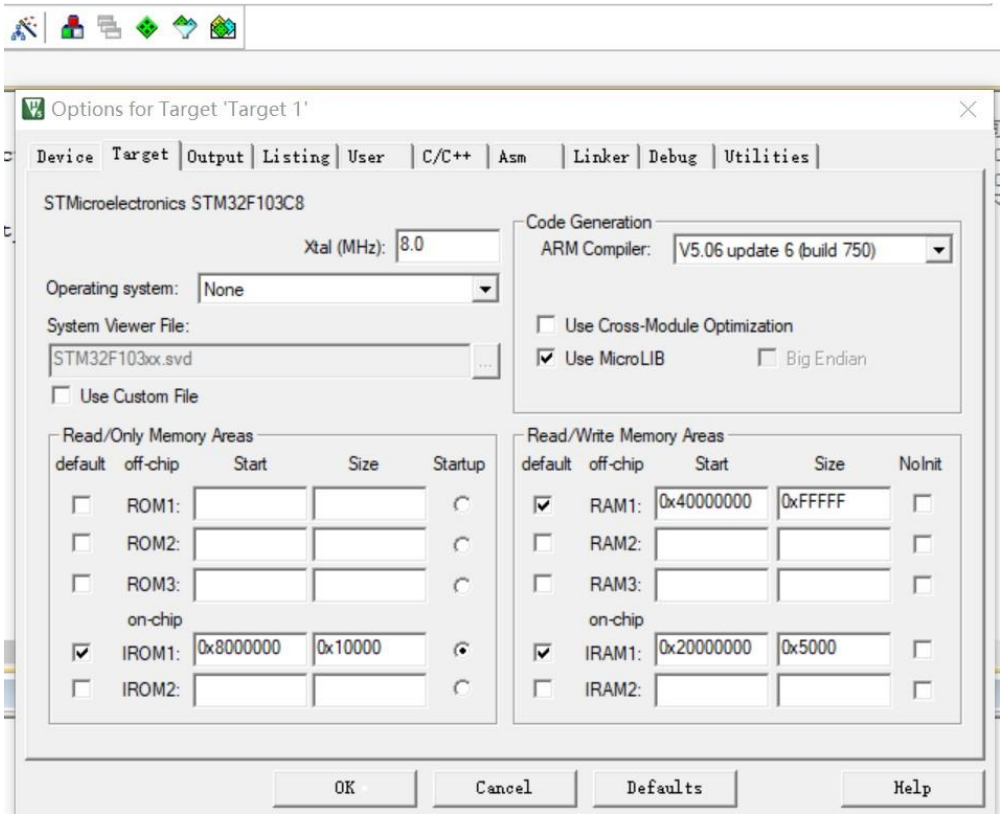
8） 在弹出的文本框内输入程序，点击保存，保存文件名如：ABT.s。注意文件可以任意设置，后缀必须是.s(汇编程序)

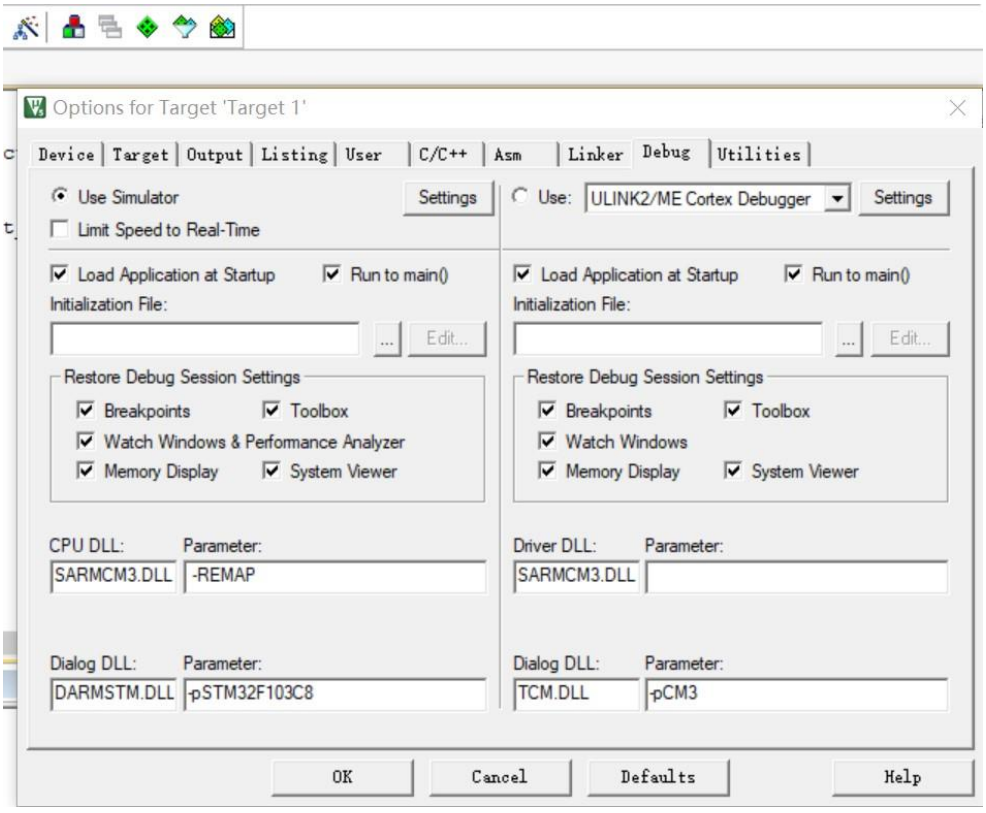
9） 在项目栏，右击“Source Group 1”—add existing files

10） 在弹出框内，选择文件后缀为.s，点击文件 ABT.s，将程序段加入到项目内。



11） 检查编译与仿真设置：

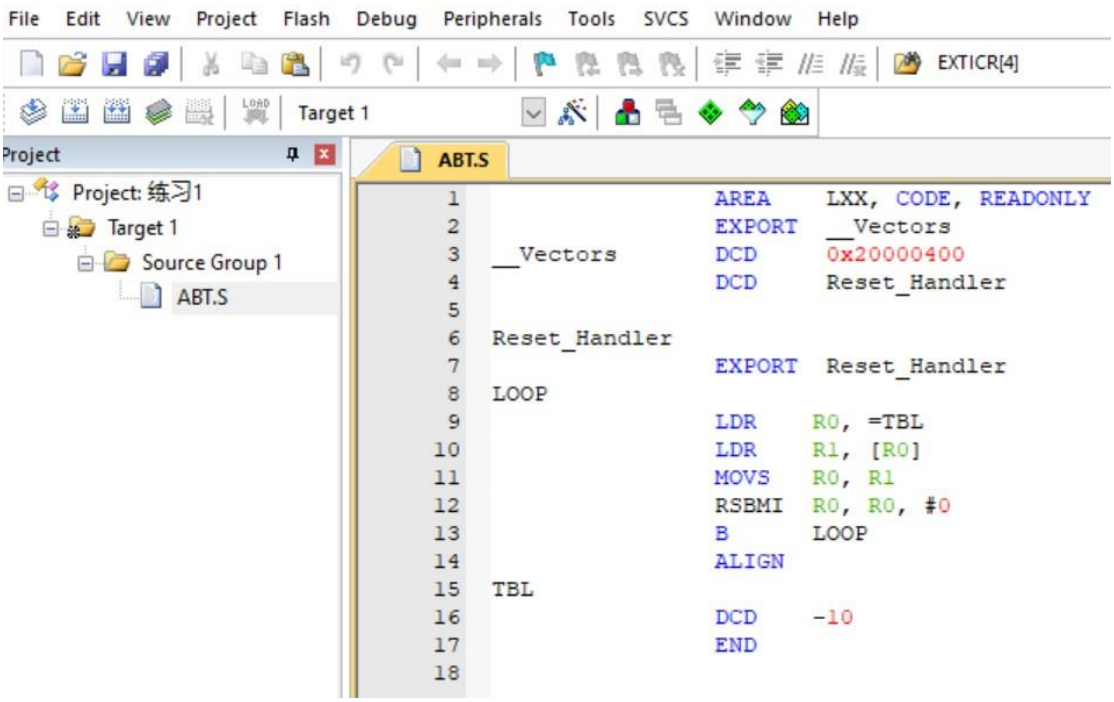




12） 点击编译按钮，检查指令和语法；

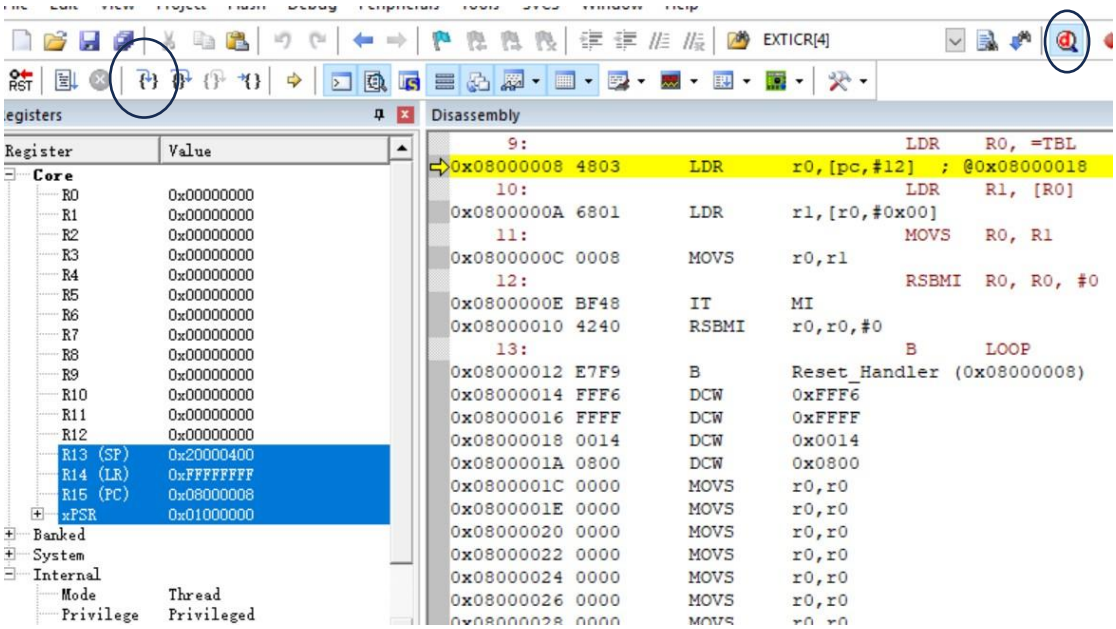
13） 点击 Build 按钮，生成目标文件、16 进制文件，检查链接；

14） 点击 Rebuild 按钮，完整编译；

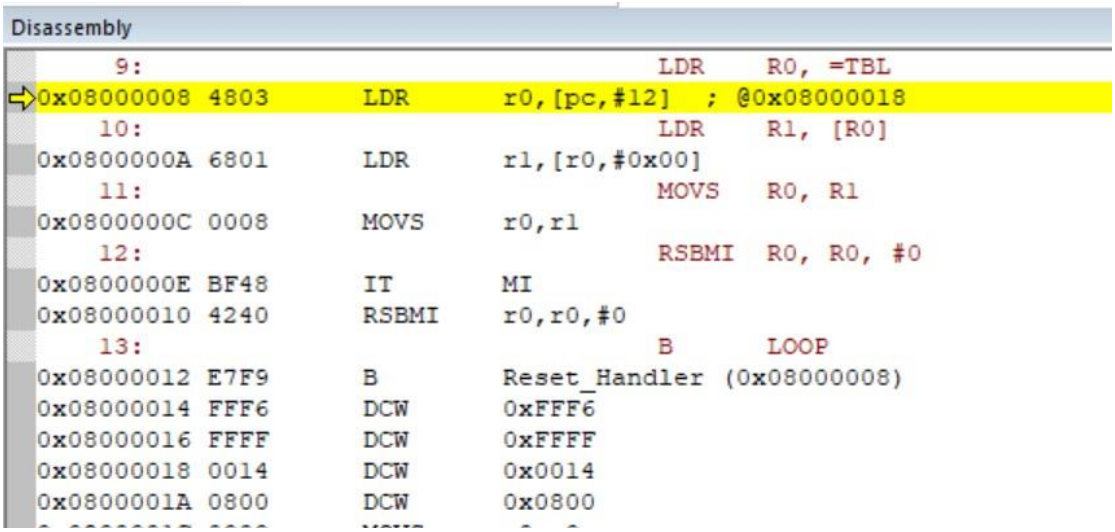


15） 点击仿真按钮，进入仿真界面；

16） 点击单步运行，观察程序运行过程，观察各寄存器值的变化过程；

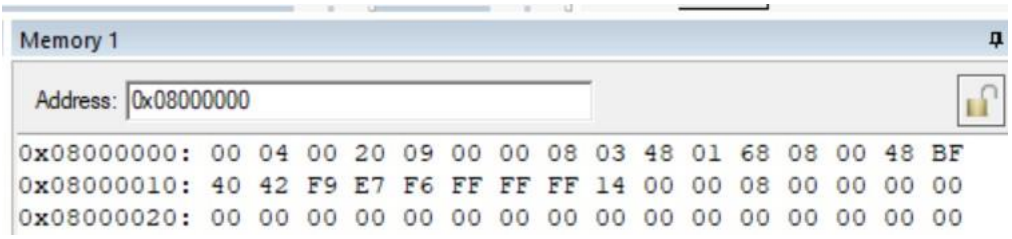


17） 观察反汇编窗口，查看指令的机器码；



18） 点击 View—Memony windows 开启一个内存观察窗口；

19） 观察 ROM 地址 0x80000000 开始的内容，这些单元内存放了程序的机器码。



【实验内容 2】寻址方式与汇编指令

1）编写项目，程序段如下。观察程序的运行结果，并给出说明。

```
MOVS    R0,   #0
SBC     R3,   R2,   R1      ;R3=_____
BIC     R4,   R3,   #7      ;R4=_____
MVN     R0,   #255
EOR     R5,   R0,   #7      ;R5=_____
MOV     R0, #1
RSB     R6, R0, R1          ;R0=_____
CMPS    R7, #1              ;N=_____ C=_____ Z=_____ V=_____
```

【实验小结】

1、STM32 微处理器的程序结构。ROM 地址 0x0800 0000~0x0800 0007 这 8 个字节内的内容是什么？为什么？

2、总结立即数的规范：立即数赋值什么情况下用 LDR 指令、什么情况下用 MOV 指令？

3、对于不存在的汇编指令，可以通过其他指令编程实现。编写一个小程序，实现带进位标记的循环左移功能。

实验二　程序设置与调试

【实验目的】

1、熟悉 STM32 程序设计流程；

2、掌握分支程序与循环程序的编程技巧；

3、熟悉 KEIL 程序调试方法。

【实验内容】编写程序（汇编语言、C 语言均可）

1、 将 R0 内的数值转换为十进制数，十进制数的每个位存入 1 个字节中，要求存放十进制位的字节地址为 0X2000 0000~0x20000009

2、 比较两个有符号数值（存放地址 0X2000 0000、0x2000 0004）的大小，比较结果>、=、<分别用 1、0、-1 表示，存入地址 0x2000 0008。

【参考步骤】

1. 将 R0 内的数值转换为十进制数，并存入指定地址：

步骤：

1. **定义存储十进制数的数组：**
   - 定义一个长度为 10 的字节数组（如 `uint8_t decimal_digits[10]`），用于存储 R0 内数值转换得到的十进制数每位数字。
2. **转换数值为十进制：**
   - 初始化一个变量（如 `uint32_t value`）赋值为 R0 中的数值。
   - 从低位到高位，依次计算 `value` 除以 10 的余数，将其存入 `decimal_digits` 数组的相应位置。
   - 对 `value` 执行整除 10 的操作，直到 `value` 变为 0，此时 `decimal_digits` 数组已按顺序填充了十进制数的每位数字。
3. **将十进制数存入指定地址：**
   - 使用 `volatile uint8_t *destination = (volatile uint8_t *)0x20000000;`声明指向指定地址的指针。
   - 从 `decimal_digits` 数组的末尾开始，逐个将数组元素的值通过指针赋值给指定地址的内存单元。

2. 比较两个有符号数值的大小，并存入比较结果：

步骤：

1. **读取比较数值：**
   - 定义两个 `int32_t` 变量（如 `num1` 和 `num2`）。
   - 通过指针（如 `volatile int32_t *ptr1 = (volatile int32_t *)0x20000000;`和 `volatile int32_t *ptr2 = (volatile int32_t *)0x20000004;`）读取指定地址的两个有符号数值，分别赋值给 `num1` 和 `num2`。
2. **比较数值大小：**

- 使用条件语句（如 `if...else if...else`）比较 `num1` 和 `num2` 的大小。
- 如果 `num1 > num2`，将结果 1 存入指定地址（如`*(volatile int8_t *)0x20000008 = 1;`）。
- 如果 `num1 == num2`，将结果 0 存入指定地址（如`*(volatile int8_t *)0x20000008 = 0;`）。
- 如果 `num1 < num2`，将结果-1 存入指定地址（如`*(volatile int8_t *)0x20000008 = -1;`）。

【实验小结】

1、算法说明，程序流程；

2、程序清单；

3、编译错误及其处理方法总结；

4、断点设置与使用方法，程序调试结果。

实验三　指示灯闪烁

【实验目的】

1、掌握 GPIO 的设置方法；

2、掌握 Systick 的使用方法；

3、熟悉 KEIL 程序调试方法。

【实验内容】编写程序（汇编语言、C 语言均可）

1、利用软件延时，使指示灯 PC13 闪烁（0.5 秒亮，0.5 秒灭）；

2、 使用 Systick 做延时，使指示灯 PC13 闪烁（0.5 秒亮，0.5 秒灭）；

【参考步骤】

1. 使用软件延时实现闪烁：

步骤：

1. **初始化 GPIO：**
   - 使能 GPIOC 的时钟。
   - 配置 PC13 为推挽输出模式，设置速度为 50MHz。
2. **定义延时函数：**
   - 编写一个名为 `delay_ms` 的函数，接受一个整数参数 `ms`，表示延时的毫秒数。
   - 函数内部使用嵌套循环实现延时，外层循环控制延时的总毫秒数，内层循环利用`__asm__("nop")`指令（或等效的空操作）实现微秒级别的延时。
3. **主循环闪烁指示灯：**
   - 在无限循环中，首先调用 `LED_ON` 宏（对应 GPIO_SetBits 函数）点亮 PC13。
   - 调用 `delay_ms(500)`延时 0.5 秒。
   - 然后调用 `LED_OFF` 宏（对应 GPIO_ResetBits 函数）熄灭 PC13。
   - 再次调用 `delay_ms(500)`延时 0.5 秒。

2. 使用 Systick 定时器实现闪烁：

步骤：

1. **初始化 GPIO：**
   - 使能 GPIOC 的时钟。
   - 配置 PC13 为推挽输出模式，设置速度为 50MHz。
2. **初始化 Systick 定时器：**
   - 设置 Systick 定时器的时钟源（通常为系统时钟），使其每 1 毫秒产生一次中断。
   - 注册 Systick 中断服务函数 `SysTick_Handler`。
3. **定义延时函数：**
   - 编写一个名为 `delay_ms` 的函数，接受一个整数参数 `ms`，表示延时的毫秒数。

- 函数内部使用全局变量 `tick_count` 记录 Systick 中断次数，通过比较 `tick_count` 的初始值和当前值，实现精确延时。

4. **主循环闪烁指示灯：**
   - 在无限循环中，首先调用 `LED_ON` 宏（对应 GPIO_SetBits 函数）点亮 PC13。
   - 调用 `delay_ms(500)`延时 0.5 秒。
   - 然后调用 `LED_OFF` 宏（对应 GPIO_ResetBits 函数）熄灭 PC13。
   - 再次调用 `delay_ms(500)`延时 0.5 秒。

【实验小结】

1、接线电路图、算法说明，程序流程；

2、程序清单；

3、编译错误及其处理方法总结；

4、断点设置与使用方法，程序调试结果。

实验四　指示灯控制

【实验目的】

1、掌握 GPIO 的设置方法；

2、掌握 Systick 的使用方法；

3、掌握边沿信号捕捉的编程方法。

【实验内容】（2、3 任选、汇编语言、C 语言均可）

1、按钮 PA00 闭合 1 秒后，灯 PC13 亮；按钮断开，灯 PC13 灭；

2、每按一次按钮 PA00，灯 PC13 的状态切换一次（按钮做去抖动处理）；

3、每按两次按钮 PA00，灯 PC13 的状态切换一次（按钮做去抖动处理）。

【参考步骤】

1. 按钮 PA00 闭合 1 秒后，灯 PC13 亮；按钮断开，灯 PC13 灭：

步骤：

1. **初始化 GPIO：**
   - 使能 GPIOA 和 GPIOC 的时钟。
   - 配置 PA0 为输入模式（带上下拉），PC13 为推挽输出模式。
2. **定义延时函数：**
   - 编写一个名为 `delay_ms` 的函数，接受一个整数参数 `ms`，表示延时的毫秒数。
   - 函数内部可以使用 Systick 定时器或软件延时方法实现指定毫秒数的延时。
3. **主循环检测按钮状态：**
   - 在无限循环中，读取 PA0 的电平状态。
   - 如果 PA0 为高电平（按钮闭合），调用 `delay_ms(1000)`延时 1 秒。
   - 在延时结束后，检查 PA0 是否仍然为高电平（防止短暂触碰）。如果是，调用 `LED_ON` 宏（对应 GPIO_SetBits 函数）点亮 PC13。
   - 如果 PA0 变为低电平（按钮断开），调用 `LED_OFF` 宏（对应 GPIO_ResetBits 函数）熄灭 PC13。

2. 每按一次按钮 PA00，灯 PC13 的状态切换一次（按钮做去抖动处理）：

步骤：

1. **初始化 GPIO：**
   - 使能 GPIOA 和 GPIOC 的时钟。
   - 配置 PA0 为输入模式（带上下拉），PC13 为推挽输出模式。
2. **定义延时函数：**
   - 同上，编写一个名为 `delay_ms` 的函数实现延时。
3. **定义去抖动函数：**
   - 编写一个名为 `debounce_button` 的函数，接受一个 GPIO 端口和引脚编号作为参数。

- 函数内部，连续读取指定引脚电平若干次（如 10 次），并计算高电平出现的次数。
- 如果高电平出现次数超过阈值（如 8 次），则认为按钮稳定按下，返回真；否则返回假。

4. **主循环检测按钮状态并切换灯状态：**
   - 在无限循环中，调用 `debounce_button(GPIOA, GPIO_Pin_0)`判断 PA0 是否稳定按下。
   - 如果按钮稳定按下，调用 `LED_TOGGLE` 宏（对应 GPIO_ToggleBits 函数）切换 PC13 的电平状态。

【实验小结】

1、接线电路图、算法说明，程序流程；

2、程序清单；

3、编译错误及其处理方法总结；

4、断点设置与使用方法，程序调试结果。

实验五　中断控制

【实验目的】

1、掌握 EXIT 中断的配置方法；

2、掌握 AFIO 的使用方法；

3、熟悉 KEIL 程序调试方法。

【实验内容】编写程序（汇编语言、C 语言均可）

1、GPIO 初始化；

2、EXTI 中断配置及初始化；

3、编写中断服务程序。

【参考步骤】

1、初始化 GPIO：

- 使能 GPIOB、AFIO 的时钟；
- 配置外部中断输入模式(浮空、上拉或下拉)；
- 配置 AFIO 和 EXTI。

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);

GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource14);

EXTI_InitTypeDef EXTI_InitStructure;
EXTI_InitStructure.EXTI_Line = EXTI_Line14;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_Init(&EXTI_InitStructure);
```

2、配置 NVIC：

- 中断优先级分组；
- 指定中断通道、配置抢占式优先级、配置响应式优先级、中断使能。

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

NVIC_InitTypeDef NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
NVIC_Init(&NVIC_InitStructure);
```

3、中断服务程序：

- 调用中断函数，设置指定的中断线；
- 每次中断结束计数一次，并清除标志位；
- 程序调试。

```
void EXTI15_10_IRQHandler(void)
{
  if (EXTI_GetITStatus(EXTI_Line14) == SET)
  {
    /*如果出现数据乱跳的现象，可再次判断引脚电平，以避免抖动*/
    if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_14) == 0)
    {
      CountSensor_Count ++;
    }
    EXTI_ClearITPendingBit(EXTI_Line14);
  }
}
```

【实验小结】

1、接线电路图、算法说明，程序流程；

2、程序清单；

3、编译错误及其处理方法总结；

4、断点设置与使用方法，程序调试结果。

实验六　ADC

【实验目的】

掌握 ADC 的使用方法。

【实验内容】编写程序（汇编语言、C 语言均可）

1、GPIO 初始化；

2、ADC 初始化；

3、ADC 校准；

4、ADC 转换。

【参考步骤】

1、开启时钟及分频、设置 GPIO 模式、ADC 初始化、ADC 使能、复位 ADC 校准、开启 ADC 校准：

.H | dsgshow.c | main.c | public.h | ADC.C

```
#include "ADC.H"
void adc1_init()
{
  GPIO_InitTypeDef GPIO_InitStructure;
  ADC_InitTypeDef ADC_InitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_AFIO , ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);
  RCC_ADCCLKConfig(RCC_PCLK2_Div6);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0| GPIO_Pin_1| GPIO_Pin_2| GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOA, &GPIO_InitStructure);


  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);
  ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_55Cycles5 );
  ADC_Cmd(ADC1,ENABLE);
  ADC_ResetCalibration(ADC1);
  while(ADC_GetResetCalibrationStatus(ADC1));
  ADC_StartCalibration(ADC1);
```

2、启动 ADC 获取转换结果：

- 软件触发转换；
- 判断标志位，即是否完成转换；
- 返回转换结果。

```
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
return ADC_GetConversionValue(ADC1);
```

【实验小结】

1、接线电路图、算法说明，程序流程；

2、程序清单；

3、编译错误及其处理方法总结；

4、断点设置与使用方法，程序调试结果。

实验七　串口通讯

【实验目的】

掌握 USART 的使用方法；

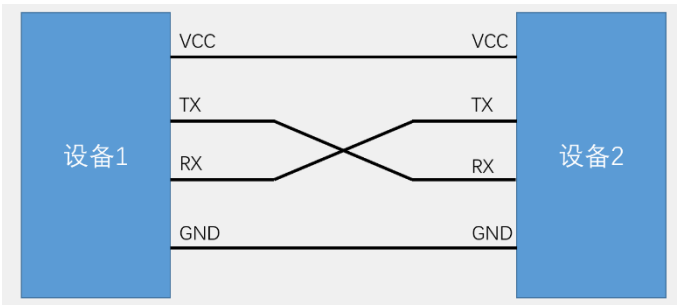
【实验内容】

利用 USART 接口实现微处理器与电脑的通讯。按钮 PA00 闭合时，微处理器向电脑串口发送字符串“ON”，电脑收到信息后，输出指令使指示灯 PC13 点亮；按钮 PA00 断开时，微处理器向电脑串口发送字符串“OFF”，电脑收到信息后，输出指令使指示灯 PC13 灭。

课程预习指导：

1. 关键概念：USART 功能，引脚为 TX/RX 脚，全双工模式，异步时钟，单端电平，点对点通讯模式。
2. 发送数据：3.3V 电平=逻辑信号 1；0V 电平=逻辑电平 0.
3. 异步时钟：没有时钟线，需要提前约定采用频率。
4. 单端电平：各自的电平都针对 GND（大地电平）而言，所以通讯设备需要共地，保证 0V 电压基准一致。
5. 点对点：通讯只有 2 个设备参加，无法广播，无需寻址。
6. 波特率：串口的通讯速率，常见数值 9600 Baud/s，波特=码元。在二进制设备中，一个码元=一个 bit 位。
7. 特别注意：USART 对应有 UART（同步通讯），UART 多了一个对外的时钟输出引脚。

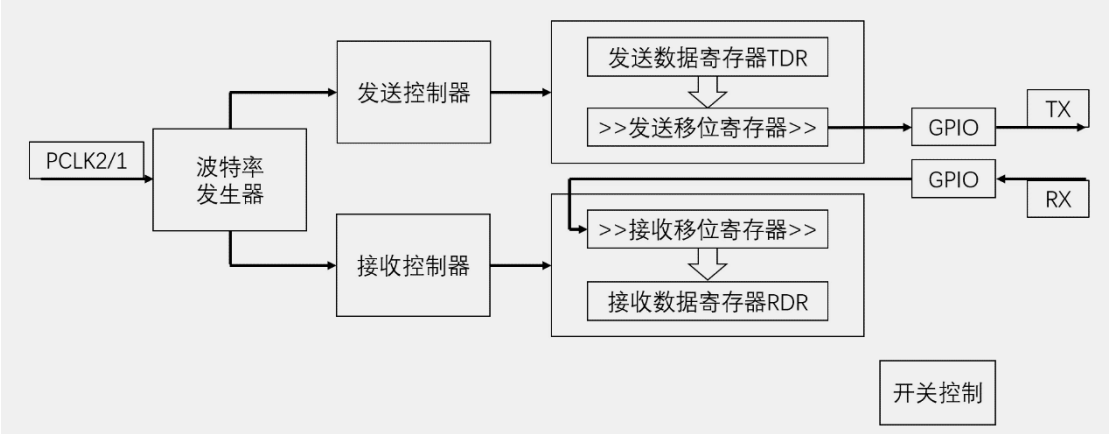
标准串口通讯的连线方式：



注意：设备 1 的 TX 接设备 2 的 RX，反之亦然。

发送过程：TX 管脚发出数据，该数据是一个字节数据转化成相应的 bit 码字输出，可以直接使用，无需考虑具体的转换算法。

关键寄存器解读：



- 关键库：Serial.h/.c
- 关键函数：Serial_SendByte;Serial_SendArray;
- 关键讨论：从 printf 到 Serial_Printf

关键函数都是基于 USART_Init 函数完成初始化功能：

工作模式：USART_Mode_Tx 和 USART_Mode_Rx

基础函数：USART_SendData、USART_ReceiveData、USART_GetFlagStatus

特别注意：“\r\n”字符串表达的意义。ASCII 码字的表达方式。

实验目预习思考要点：

- 如何让 STM32 芯片实现对 PA00 的状态（断开闭合）监听？
  - 提示：使用中断还是看门狗？
- 如何让 STM32 读取判断上位机（电脑）发送的字符？
  - 提示：是否使用 while 或者 if 等循环判定语句？

【参考步骤】

1. 导线连接：

- USB 转串口上 VCC 和 3V3 引脚相连；
- USB 转串口上 TXD 接 STM32 的 PA10
- USB 转串口上 RXD 接 STM32 的 PA9
- 连接 GND，保持电平参考相同

2. 串口模块驱动安装

- USB-SERIAL CH340(COMx)

3. 编写 Serial_Init 函数

- 开启 USART1 的时钟 RCC APB2PeriphClockCmd(RCC APB2Periph USART1ENABLE):
- 开启 GPIOA 的时钟 RCC APB2PeriphClockCmd(RCC APB2Periph GPIOA，ENABLE);
- 定义 PA9 的工作模式（结构体初始化，AF_PP 模式，工作频率 50 MHz）

4. 初始化 USART1

- USART_InitStructure 结构体初始化 USART_Mode = USART_Mode_Tx;
- USART_Cmd 时钟使能

5. USART 发送数据函数 Serial_SendByte 编写，使用 PA9

- 基于 USART_SendData(USART1, Byte);
- 配合 USART_GetFlagStatus(USART1, USART_FLAG_TXE)函数使用

6. printf 函数使用

- #include<stdio.h>调用

7. USART 接收数据函数 Serial_ReceiveByte 编写，使用 PA10

- 定义 PA10 的工作模式（结构体初始化，IPU 模式，工作频率 50 MHz）
- 修改 USART 结构体初始化 USART_Mode = USART_Mode_Tx I USART_Mode_Rx;

8. USART 发送数据函数 Serial_ReceiveByte 编写，使用 PA10

- 基于 USART_ReceiveData(USART1);
- 配合 USART_GetFlagStatus(USART1, USART_FLAG_TXE)函数使用

【实验小结】

1、接线电路图、算法说明，程序流程；

2、程序清单；

3、编译错误及其处理方法总结；

4、断点设置与使用方法，程序调试结果。

实验八　PWM 与直流电机调速

【实验目的】

掌握 PWM 的使用方法；

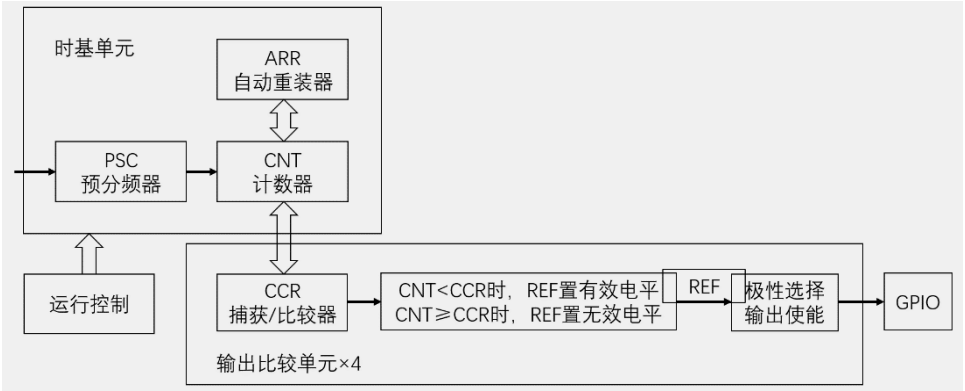
【实验内容】

利用微处理器的定时器，实现对电机的调速控制。按加速按钮 PA00，电机速度增加，按减速按钮 PA01，电机速度降低。

课程预习指导：

1. 关键概念：PWM 功能，引脚选择，定时器 TIM，占空比。
   a) TIM 计数是从 0 开始计算的
2. 发送数据：3.3V 电平=逻辑信号 1；0V 电平=逻辑电平 0.
3. 配置逻辑：时基单元配置，输出比较单元配置。
4. 引脚输出模式：复用推挽输出，AF_PP 模式支持 PWM 输出。
5. 电机控制与电机控制电路板工作原理。
6. 特别注意：GPIO 定义下，输出引脚和 TIM 功能是硬件定义的，AFIO 的引脚重映射可解。

关键配置：时基单元、输出比较单元



- 关键库：stm32f10x_tim.h/.c
- 关键函数：Motor_SetSpeed
  - TIM_OC1Init，输出比较单元;
  - TIM_CtrlPWMOutputs,使能 PWM 输出功能；

- TIM_Cmd，启动定时器。

- 关键讨论：PWM 输出波形参数的计算方程组

PWM 频率：　Freq = CK_PSC / (PSC + 1) / (ARR + 1)

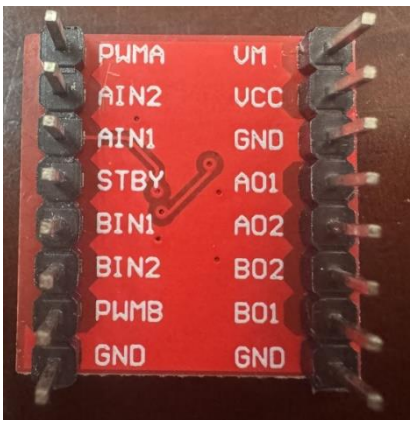
PWM 占空比：Duty = CCR / (ARR + 1)

PWM 分辨率：Reso = 1 / (ARR + 1)

电机硬件结构：

电机管脚：正负级 2 个端口。

电机驱动电路：大功率器件 STM32 输出功率不足，需配置 TB6612 芯片代为驱动。驱动电路关键引脚如下：



VM：5V 供电电源

VCC：电平参考电源

GND：0V 地

AO1，AO2：接电机正负极

PWMA：PWM 信号输入

AIN2，AIN1：电机控制模式输入端，高低配合控制转动方向

STBY：工作/待机控制，3.3V=工作

实验目预习思考要点：

- 如何让生成一个频率为 10 kHz，占空比为 30%，分辨率为 1% 的 PWM 波形？
  - 提示：求解三个方程组即可。

- 电机驱动需要 5V 电源，STM32 上只有 3.3V，如何解决？
- 电机有正负级，接反了会怎么样？

【参考步骤】

1. 导线连接：

- 电机驱动板 PWMA 接 STM32 的 PA2；
- 电机驱动板 AIN2 接 STM32 的 PA5；
- 电机驱动板 AIN1 接 STM32 的 PA4；
- 电机驱动板 STBY 接 3.3V 电平；
- 电机驱动板 VM 接外置 5V 电平；
- 电机驱动板 VCC 接 3.3V 电平；
- 电机驱动板 AO1 和 AO0 接电机的正负极；

2. 编写 PWM_Init 函数

- 开启 TIM2 的时钟 RCC APBlPeriphClockCmd(RCC APBlPeriph TIM2ENABLE):
- 开启 GPIOA 的时钟 RCC APB2PeriphClockCmd(RCC APB2Periph GPIOA，ENABLE);
- 定义 PA2 的工作模式（结构体初始化，AF_PP 模式，工作频率 50 MHz）
- 开启 TIM2 的中断函数 TIM InternalclockConfig(TIM2);

3. 初始化 TIM 的 TimeBase

- TIM Period=100-1;
- TIM Prescaler=720-1;
- TIM RepetitionCounter =0;
-

4. 初始化 TIM 的 OC

- USART_InitStructure 结构体初始化
- TIM_OCMode = TIM_OCMode_PWM1;
- TIM_OC Polarity = TIM_OCPolarity_High;
- TIM_Outputstate = TIM_Outputstate_Enable;
- TIM_Pulse=0;

5. 编写 Motor_SetSpeed 函数

- GPIO_SetBits/ResetBits 函数控制 PA5 和 PA4 的高低电平，控制电机旋转的方向
- PWM_SetCompare 函数控制 PWM 输出占空比，控制电机输出速度。

6. 编写 PWM_SetCompare 函数

- 调用库函数中的 TIM_SetCompare3(TIM2, Compare)函数即可，作用是修改 TIM2 的 CCR 数值，从而实现占空比的调整。

【实验小结】

1、接线电路图、算法说明，程序流程；

2、程序清单；

3、编译错误及其处理方法总结；

4、断点设置与使用方法，程序调试结果。