第 5 章 基于构件的汇编程序设计方法

本章主要分析系统构件化设计的重要性和必要性，给出软件构件基本概念及构件设计中所需遵循的基本原则；给出了程序流程控制基本方法，包括顺序结构、分支结构、循环结构等条件下的执行实例；给出了几个典型汇编程序设计实例；给出了基于构件方法的汇编程序设计实验等基本内容。

5.1 构件及其设计方法

机械、建筑等传统产业的运作模式是先生产符合标准的构件（零部件），然后将标准构件按照规则组装成实际产品。其中，构件（Component）是核心和基础，复用是必需的手段。传统产业的成功充分证明了这种模式的可行性和正确性。软件产业的发展借鉴了这种模式，为标准软件构件的生产和复用确立了举足轻重的地位。

随着微控制器及应用处理器内部 Flash 存储器可靠性提高及擦写方式的变化，内部 RAM 及 Flash 存储器容量的增大，以及外部模块内置化程度的提高，嵌入式系统的设计复杂性、设计规模及开发手段已经发生了根本变化。在嵌入式系统发展的最初阶段，嵌入式系统硬件和软件设计通常是由一个工程师来承担，软件在整个工作中的比例很小。随着时间的推移，硬件设计变得越来越复杂，软件的份量也急剧增长，嵌入式开发人员也由一人发展为由若干人组成的开发团队。为此，希望提高软硬件设计可重用性与可移植性，构件的设计与应用是重用与移植的基础与保障。

5.1.1 软件构件基本概念

软件构件（Software component）广义上的理解是：可复用的软件成分。截至目前有多种多样关于软件构件的定义，但本质是相同的。这里给出了面向构件程序设计工作组（Szyperski 和 Pfister，1997）在 1996 年的面向对象程序设计欧洲会议上（European Conference On Object-Oriented Programming，ECOOP）给出的软件构件的定义：**软件构件是一种组装单元，它具有规范的接口规约和显式的语境依赖。软件构件可以被独立地部署并由第三方任意地组装。**它既包括了技术因素，例如独立性、合约接口、组装，也包括了市场因素，例如第三方和部署。就技术和市场两方面的因素融为一体而言，即使是超出软件范围来评价，构件也是独一无二的。而从当前的角度上看，上述定义仍然需要进一步澄清。这是因为一个可部署构件的合约内容远远不只是接口和语境依赖，它还要规定构件应该如何部署、一旦部署（和启动）了应该如何被实例化、实例如何通过规定的接口工作等。

再列举其他文献给出的定义，以便了解对软件构件定义的不同表达方式，也可看作从不同角度定义软件构件。

美国卡内基梅隆大学软件工程研究所（Carnegie-Mellon University/Software Engineering Institute，CMU/SEI）给出的软件构件的定义：**构件是一个不透明的功能实体，能够被第三方组织，且符合一个构件模型。**

国际上第一部软件构件专著的作者 Szyperski 给出的软件构件的定义：可单独生产、获取、部署的二进制单元，它们之间可以相互作用构成一个功能系统。

软件构件技术的出现，为实现软件构件的工业化生产提供了理论与技术基石。将软件构件技术应用到嵌入式软件开发中，可以大大提高嵌入式开发的开发效率与稳定性。软件构件的封装性、可移植性与可复用性

是软件构件的基本特性，采用构件技术设计软件，可以使软件具有更好的开放性、通用性和适应性。特别是对于底层硬件的驱动编程，只有封装成底层驱动构件，才能减少重复劳动，使广大 MCU 应用开发者专注于应用软件稳定性与功能设计上。因此，必须把底层硬件驱动设计好、封装好。

国内外对于软件构件的定义曾进行过广泛讨论，有许多不同说法。但是到目前为止依然没有形成一个能够被广泛接受的定义，不同的研究人员对构件有着不同的理解。一般可以将软件构件定义为：**在语义完整、语法正确情况下，具有可复用价值的单位软件是软件复用过程中可以明确辨别的成分；从程序角度上可以将构件看作是有一定功能、能够独立工作或协同其他构件共同完成的程序体。**

5.1.2 构件设计基本原则

1．构件设计的基本思想

底层构件是与硬件直接打交道的软件，它被组织成具有一定独立性的功能模块，由头文件和源程序文件两部分组成。构件的头文件名和源程序文件名一致，且为构件名。

构件的头文件中，主要包含必要的引用文件、描述构件功能特性的宏定义语句以及声明对外接口函数。良好的构件头文件应该成为构件使用说明，不需要使用者查看源程序。

构件的源程序文件中包含构件的头文件、内部函数的声明、对外接口函数的实现。

将构件分为头文件与源程序文件两个独立的部分，意义在于，头文件中包含对构件的使用信息的完整描述，为用户使用构件提供充分必要的说明，构件提供服务的实现细节被封装在源程序文件中；调用者通过构件对外接口获取服务，而不必关心服务函数的具体实现细节。这就是构件设计的基本内容。

在设计底层构件时，最关键的工作是要对构件的共性和个性进行分析，设计出合理的、必要的对外接口函数及其形参。**尽量做到：当一个底层构件应用到不同系统中时，仅需修改构件的头文件，对于构件的源程序文件则不必修改或改动很小。**

2．构件设计的基本原则

在嵌入式软件领域中，由于软件与硬件紧密联系的特性，使得与硬件紧密相连的底层驱动构件的生产成为嵌入式软件开发的重要内容之一。良好的底层驱动构件具备如下特性：

（1）**封装性**。在内部封装实现细节，采用独立的内部结构以减少对外部环境的依赖。调用者只通过构件接口获得相应功能，内部实现的调整将不会影响构件调用者的使用。

（2）**描述性**。构件必须提供规范的函数名称、清晰的接口信息、参数含义与范围、必要的注意事项等描述，为调用者提供统一、规范的使用信息。

（3）**可移植性**。底层构件的可移植性是指同样功能的构件，如何做到不改动或少改动，而方便地移植到同系列及不同系列芯片内，减少重复劳动。

（4）**可复用性**。在满足一定使用要求时，构件不经过任何修改就可以直接使用。特别是使用同一芯片开发不同项目，底层驱动构件应该做到复用。可复用性使得高层调用者对构件的使用不因底层实现的变化而有所改变，可复用性提高了嵌入式软件的开发效率、可靠性与可维护性。不同芯片的底层驱动构件复用需在可移植性基础上进行。

为了使构件设计满足**封装性、描述性、可移植性、可复用性**的基本要求，嵌入式底层驱动构件的开发，应遵循**层次化、易用性、鲁棒性及对内存的可靠使用原则**。

（1）层次化原则

层次化设计要求清晰地组织构件之间的关联关系。底层驱动构件与底层硬件打交道，在应用系统中位于最底层。遵循层次化原则设计底层驱动构件需要做到：

① 针对应用场景和服务对象，分层组织构件。设计底层驱动构件的过程中，有一些与处理器相关的、描述了芯片寄存器映射的内容，这些是所有底层驱动构件都需要使用的，将这些内容组织成底层驱动构件的公共内容，作为底层驱动构件的基础。在底层驱动构件的基础上，还可使用高级的扩展构件调用底层驱动构件功能，从而实现更加复杂的服务。

② 在构件的层次模型中，**上层构件可以调用下层构件提供的服务，同一层次的构件不存在相互依赖关系，不能相互调用**。例如，Flash 模块与 UART 模块是平级模块，不能在编写 Flash 构件时，调用 UART 驱动构件。即使要通过 UART 驱动构件函数的调用在 PC 机屏幕上显示 Flash 构件测试信息，也不能在 Flash 构件内含有调用 UART 驱动构件函数的语句，应该编写上一层次的程序调用。平级构件是相互不可见的，只有深入理解这一点，并遵守之，才能更好地设计出规范的底层驱动构件。在操作系统下，平级构件不可见特性尤为重要。

（2）易用性原则

易用性在于让调用者能够快速理解的构件提供服务的功能并进行使用。遵循易用性原则设计底层驱动构件需要做到：**函数名简洁且达意；接口参数清晰，范围明确；使用说明语言精炼规范，避免二义性**。此外，在函数的实现方面，避免编写代码量过多。函数的代码量过多会难以理解与维护，并且容易出错。若一个函数的功能比较复杂，可将其“化整为零”，通过编写多个规模较小功能单一的子函数，再进行组合，实现最终的功能。

（3）鲁棒性原则

鲁棒性在于为调用者提供安全的服务，避免在程序运行过程中出现异常状况。遵循鲁棒性原则设计底层驱动构件需要做到：**在明确函数输入输出的取值范围、提供清晰接口描述的同时，在函数实现的内部要有对输入参数的检测，对超出合法范围的输入参数进行必要的处理**；使用分支判断时，确保对分支条件判断的完整性，对缺省分支进行处理。例如，对 if 结构中的“else”分支和 switch 结构中的“default”安排合理的处理程序。**同时，不能忽视编译警告错误**。

（4）内存可靠使用原则

对内存的可靠使用是保证系统安全、稳定运行的一个重要的考虑因素。遵循内存可靠使用原则设计底层驱动构件需要做到：

① 优先使用静态分配内存。相比于人工参与的动态分配内存，静态分配内存由编译器维护，更为可靠。

② 谨慎地使用变量。可以直接读写硬件寄存器时，不使用变量替代；避免使用变量暂存简单计算所产生的中间结果；使用变量暂存数据将会影响到数据的时效性。

③ 检测空指针。定义指针变量时必须初始化，防止产生“野指针”。

④ 检测缓冲区溢出，并为内存中的缓冲区预留不小于 20%的冗余。使用缓冲区时，对填充数据长度进行检测，不允许向缓冲区中填充超出容量的数据。

⑤ 对内存的使用情况进行评估。

5.1.3 三类构件

提高代码质量和生产力的唯一最佳方法就是复用好的代码，软件构件技术是软件复用实现的重要方法。**为了便于理解与应用，可以把嵌入式软件构件分为基础构件、应用构件与软件构件三种类型**。

1．基础构件

基础构件的定义。基础构件是根据 MCU 内部功能模块的基本知识要素，针对 MCU 引脚功能或 MCU 内部功能，利用 MCU 内部寄存器，所制作的直接干预硬件的构件。常用的基础构件主要有：GPIO 构件、UART 构件、ADC 构件、Flash 构件等。

2．应用构件

应用构件的定义。应用构件是使用基础构件并面向对象编程的构件。例如 printf 构件，它调用串口构件完成输出显示功能。printf 函数调用的一般形式为：printf(“格式控制字符串”，输出表列)，本书使用的 printf 函数可通过 uart 串口向外传输数据。

3．软件构件

软件构件的定义。软件构件是一个面向对象的具有规范接口和确定的上下文依赖的组装单元，它能够被独立使用或被其他构件调用，如数字类型转换构件 valueType 等。人工智能的一些算法若制作成构件，也可纳入软件构件范畴。

5.1.4 基于构件的软件设计步骤

在构件制作完成基础上，软件设计分为构件测试及应用程序设计两大步骤。

1．构件测试

要进行具有相对完整功能的应用程序设计，必然要使用到构件。在构件使用之前，必须编写构件测试程序对构件进行测试，其方法参见 4.3.3 节。

2．应用程序设计

汇编语言的应用程序设计可分为主程序及中断服务程序两个部分，一般过程有：分析问题、建立数学模型、确定算法、绘制程序流程图、内存空间分配、编写程序及程序整体测试。实际上在编码过程中，每撰写几句，均可利用 printf 输出显示功能进行打桩调试。

（1）分析问题。分析问题就是将解决问题所需条件、原始数据、输入/输出信息、运行速度、运算精度和结果形式等搞清楚。对于较大问题的程序设计，一般还要用某种形式描绘一个“工艺”流程，以便于对整个问题的讨论和程序设计。“工艺”流程指的是用表格或流程图等去描述问题的物理过程。

（2）建立数学模型。就是把问题数学化、公式化。有些问题比较直观，可不去讨论数学模型的问题；有些问题符合某些公式或某些数学模型，可以直接利用；但是有些问题没有对应的数学模型可以利用，需要建立一些近似数学模型模拟问题。由于微机的运算速度很快、运算精度很高，从而近似运算也可以达到理想精度。

（3）确定算法。建立数学模型后，许多情况下还不能直接进行程序设计，需要确定符合微控制器运算的算法。因为微控制器的字长是一定的，它表示的数也是有一定范围的。如果选择的算法不合适，可能会造成运算结果与实际完全相反或误差很大。一般情况下，优先选择逻辑简单、运算速度快且精度高的算法用于

程序设计。

（4）绘制程序流程图。程序流程图是用箭头线段、框图及菱形图等绘制的一种图，它直接描述程序内容。因此，在程序设计中被广泛应用。

（5）内存空间分配。汇编语言的重要特点之一是能够直接用汇编指令或伪指令为数据或代码程序分配内存空间，当程序中没有指定分配存储空间时，系统会按约定方式分配存储空间。

（6）编写程序。汇编语言编程应按指令和伪指令的语法规则进行，编写程序首先关心的还是程序结构，任何一个复杂的程序都是由简单的基本程序构成的，汇编语言源程序的基本结构形式有顺序结构、条件转移、循环结构等。另外，程序设计通常采用模块化结构，程序的结构要具有层次简单、清楚、易读及易维护的特点。

（7）程序整体测试。程序整体测试是程序设计的最后一步，也是非常重要的一步。通过整体测试，可以纠正程序的语法错误和语义错误，最后实现程序正确的运行。

5.2 程序流程控制

程序流程控制主要有顺序结构、分支结构、循环结构。

5.2.1 顺序结构

顺序结构程序的执行方式是“从头到尾”，逐条执行指令语句，直到程序结束，这是程序的最基本形式。

【例 5-1】编程实现将两个寄存器相加得到的结果通过串口输出，假设两个寄存器存放的数据分别是 15 和 24，参考程序见“Exam5_1”工程。

分析：假定 r0 寄存器存放数据 15，r1 寄存器存放数据 24，可通过 add 指令实现两数相加，再调用串口构件函数输出结果即可，这里在打印结果前先输出字符串“Serial port information start!”和“Result：”，方便直观的看到结果现象，流程图如图 5-1，执行效果如图 5-2。

```
//（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 xfgi 结束标志
hello_information:                          //字符串标号
    .ascii "------------------------------------------------------------------\n"
    .ascii "金葫芦提示：\n"
    .ascii "      本工程实现两数相加，运行结果如下所示！      \n"
    .ascii "------------------------------------------------------------------\n\0"
data_format:
    .ascii "Result : %d\n\0"                //printf 使用的数据格式控制符
string_first_1:                             //字符串标签
    .string "Serial port information start!\n"
……
main:
……
//（2）======主循环部分（开头）=========================================
main_loop:                                  //主循环标签（开头）
……
//（2.4）两数相加函数
```

```
//（2.4.1）串口输出结果前的提示信息
        ldr r0, =string_first_1          //r0 指明串口输出提示字符串
        bl   printf                      //调用 printf
//（2.4.2）计算结果并用串口输出
        mov r0,#15
        mov r1,#24
        add r0,r0,r1
        mov r1,r0
        ldr r0,=data_format
        bl printf
        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）
```

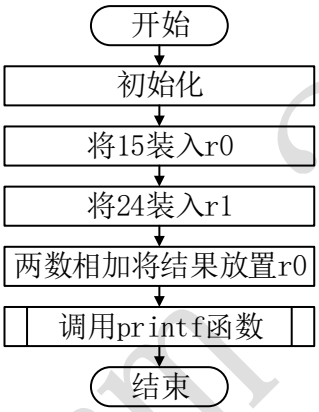

图 5-1 顺序执行程序流程图

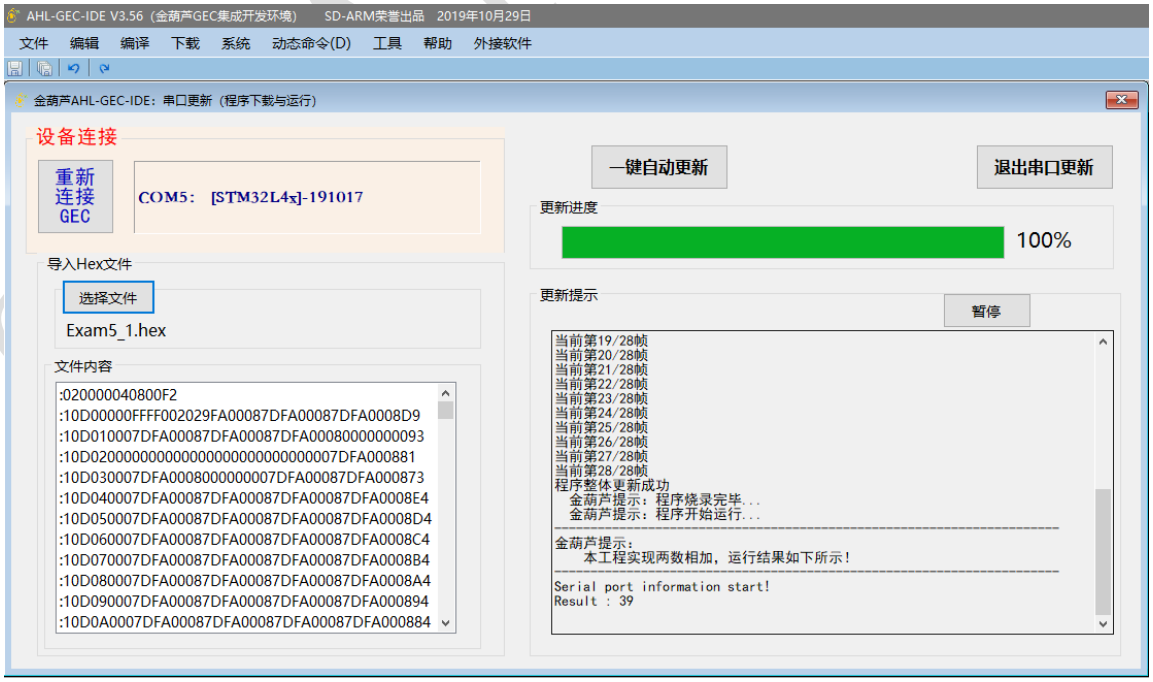

图 5-2 顺序执行结果

【练习 5-1】打开“Exam5_1”工程，将两个数分别存放在 r2 和 r3 寄存器中，然后通过 printf 输出这两

个数相减的结果。

5.2.2 分支结构

分支结构程序是利用条件转移指令，使程序执行到某一指令后，根据条件是否满足，来改变程序执行的次序，这类程序使计算机有了判断作用。分支结构分为单分支结构和多分支结构，程序流程图如图 5-3、图 5-4 所示。

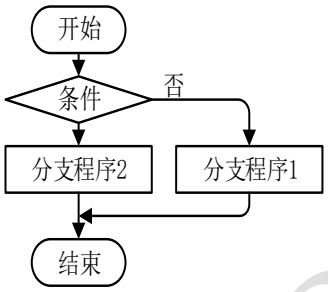

图 5-3 单分支程序流程图

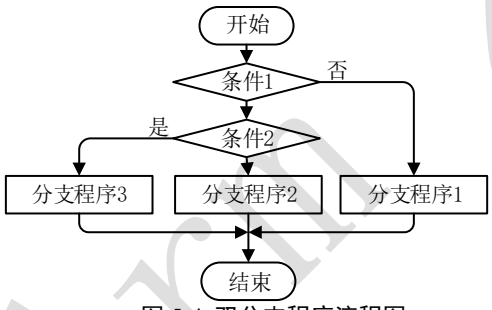

图 5-4 双分支程序流程图

【例 5-2】编程实现寄存器 A 和寄存器 B 两个无符号数之差的绝对值，将结果用串口输出，假设两寄存器存放的数据分别是 15 和 24，参考程序见“Exam5_2”工程。

分析：在此题目中，因为绝对值永远是大于等于 0，故应先判定哪一个值稍大些，再用大的数减去小的数，方可求出绝对值。该程序的流程图设计如图 5-5 所示，执行效果如图 5-6。

```
//（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 xfgi 结束标志
hello_information:                                  //字符串标号
    .ascii "------------------------------------------------------------------\n"
    .ascii "金葫芦提示：\n"
    .ascii "      本工程实现两数相减得到绝对值，运行结果如下所示！      \n"
    .ascii "------------------------------------------------------------------\n\0"
data_format:
    .ascii "Result : %d\n\0"                        //printf 使用的数据格式控制符
string_first_1:                                     //字符串标签
    .string "Serial port information start!\n"
string_control_1:
    .string "%d>%d,Result is:%d\n"                  //输出原数
```

```
string_control_2:
      .string "%d<%d,Result is:%d\n"
……
main:
……
//（2）======主循环部分（开头）=========================================
main_loop:                                          //主循环标签（开头）
……
//（2.4）计算两数的绝对值并通过串口输出结果
//（2.4.1）串口输出结果前的提示信息
            ldr r0, =string_first_1                 //r0 指明字符串
            bl    printf                            //调用 printf
//（2.4.2）计算结果并用串口输出
            mov r2,#15
            mov r3,#24
            mov r1,r2
//（2.4.3）比较 r2、r3 的值大小
            cmp r2,r3
            bls low                                 //若 r2≤r3，跳转到 low 执行
            sub r4,r2,r3
            mov r2,r3
            mov r3,r4
            ldr r0, =string_control_1
            bl    printf
 low:
            sub r4,r3,r2                            //r4=r3-r2
            mov r2,r3
            mov r3,r4
            ldr r0, =string_control_2
            bl    printf
            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）
```

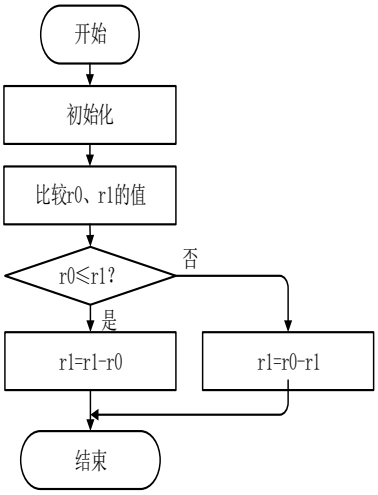

图 5-5 求绝对值程序流程图

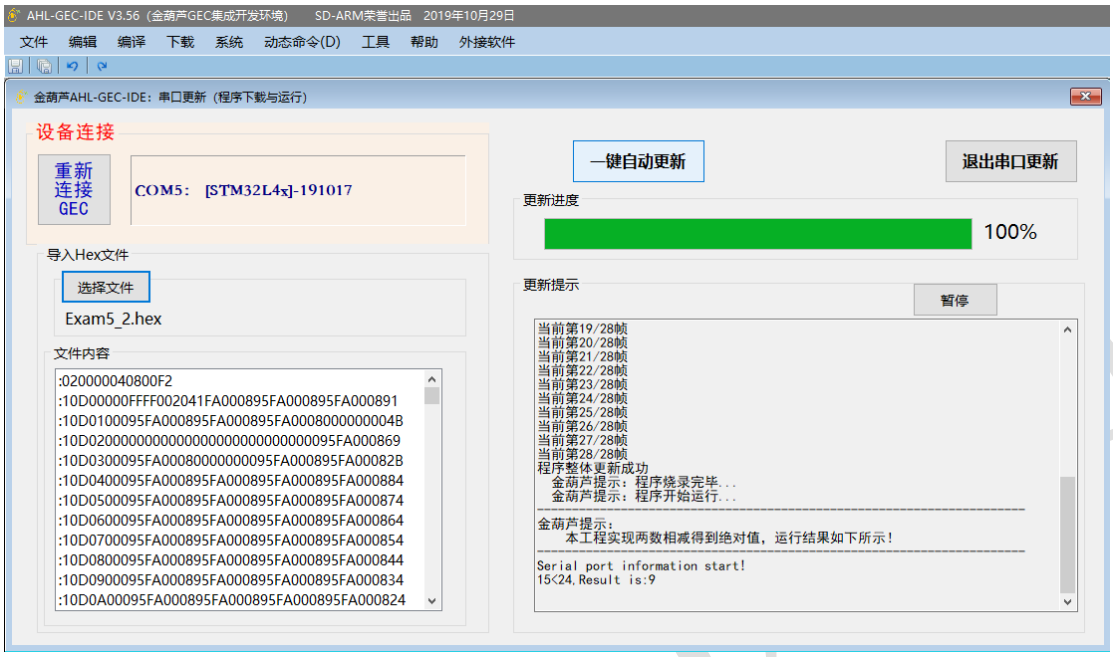

图 5-6 求绝对值执行结果

5.2.3 循环结构

循环结构程序是强制 CPU 重复执行某一指令系列（程序段）的一种程序结构形式，凡是要重复执行的程序段都可以按循环结构设计。循环结构程序不仅简化了程序清单书写形式，而且减少了内存空间。值得注意的是循环程序并不简化程序执行过程，相反，还需增加一些循环控制等环节，总的程序执行语句和时间会有所增加。循环程序一般由四部分组成：初始化、循环体、循环控制和循环结束处理，它的程序结构流程图如图 5-7、图 5-8 所示。

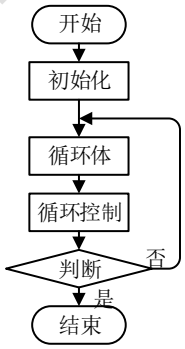

图 5-7 循环程序结构流程图

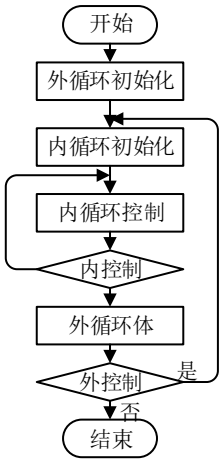

图 5-8 双重循环程序结构流程图

（1）初始化。主要功能是完成建立循环次数计数器，设定变量的初值，装入暂存单元的初值等。

（2）循环体。循环体是 CPU 执行某一指令系列（程序段）具体组成部分，在循环体内需要设定终止循环条件，否则，程序容易进入死循环。

（3）循环控制。包括修改变量，为下一次循环做准备，以及修改循环计数器（计数器减 1），判断循环次数是否到达。达到循环次数则结束循环；否则继续循环（即跳转回去，再执行一次循环）。

循环程序分为单循环和多重循环，两重以上的循环称为多重循环，如图 5-8 所示。

循环控制方式有多种，如计数控制、条件控制、状态控制等。计数控制事先已知循环次数，每次循环都要加或减 1 来表示一次计数，并通过判定循环总次数来控制循环。条件循环事先不知循环次数，在执行循环时通过判定某种条件真假来达到控制循环的目的。状态控制可事先设定二进制位的状态，或由外界干预、测试来得到开关状态，进而决定是否执行循环操作。

【例 5-3】编程实现 sum=1+2+3+4+…+100，并将 sum 的值通过串口输出，参考程序见“Exam5_3”工程。

分析：这是一个循环已知的程序设计，可将寄存器 A，B 赋值 1，然后寄存器 A 数据不断自加 1 再逐一与寄存器 B 的数据相加，得到的结果放到寄存器 B，到寄存器 A 的值累加到 100 的时候，寄存器 B 存放的数据即为 sum 的值，该程序的流程图设计如图 5-9 所示，执行效果如图 5-10。

```
//（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 xfgi 结束标志
hello_information:                                  //字符串标号
    .ascii "------------------------------------------------------------------------\n"
    .ascii "金葫芦提示：\n"
    .ascii "      本工程实现 1~100 这 100 个数相加，运行结果如下所示！       \n"
    .ascii "------------------------------------------------------------------------\n\0"
string_loop:                                        //串口输出结果前的提示信息
    .string "sum:%d\n"
string_first_1:                                     //字符串标签
    .string "Serial port information start!\n"
string_control_1:
    .string "%d>%d,Result is:%d\n"                  //输出原数字符串信息
string_control_2:
```

```
    .string "%d<%d,Result is:%d\n"
main:
……
//（2）======主循环部分（开头）=======================================
main_loop:                                  //主循环标签（开头）
……
//（2.4）计算 1 到 100 的和，并用串口输出结果
//（2.4.1）串口输出结果前的提示信息
        ldr r0, =string_first_1             //r0 指明字符串
         bl   printf                        //调用 printf
//（2.4.2）计算结果并用串口输出
         mov r0,#1
         mov r1,r0
         bl   loop                          //调用 loop 函数
 loop:
         add r0,r0,#1
         add r1,r1,r0
         mov r2,#100
         cmp r0,r2                          //比较 r0 和 r2 的值
         bne loop                           //不相等则继续跳转到 loop
         ldr r0, =string_loop
         bl   printf
         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）
```

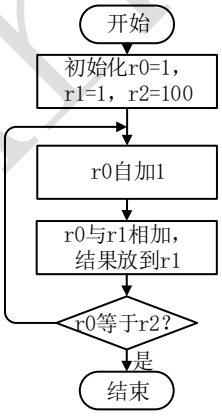

图 5-9 循环控制流程图

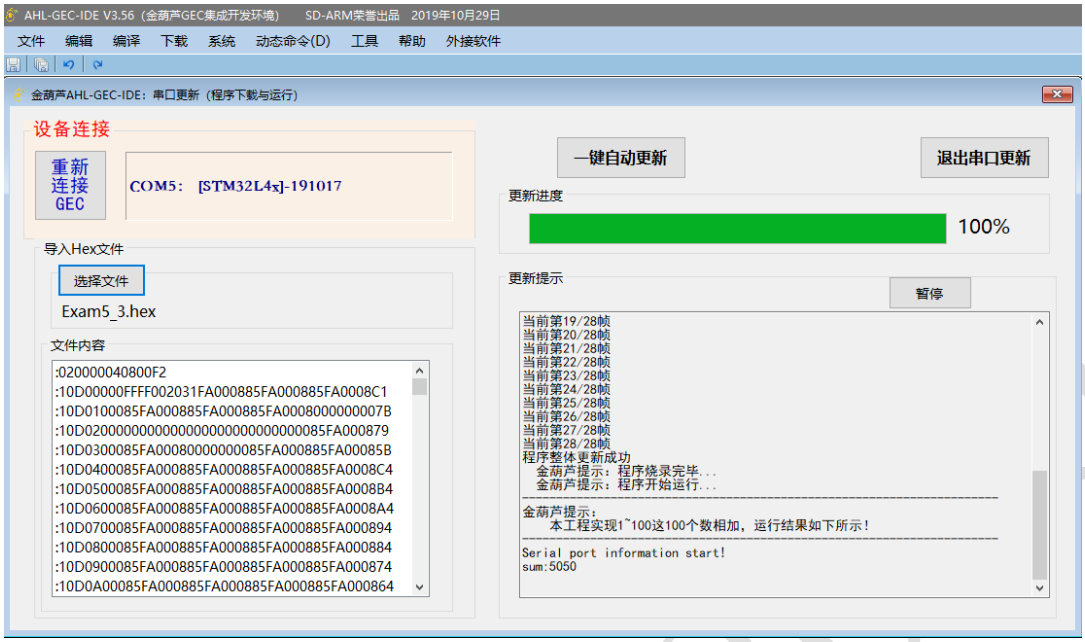

图 5-10 循环控制执行结果

【练习 5-2】打开“Exam5_3”工程，修改程序实现求 5!，并通过 printf 输出结果。

5.3 汇编程序设计实例

本节以数制转换、冒泡排序为例，并通过调用 printf 构件、LCD 构件进行过程及结果显示。

5.3.1 数制转换程序设计

【例 5-4】编写程序将十进制数转换成二进制、八进制、十六进制，结果通过串口输出。假设需要转换的十进制数是 95，参考程序见“Exam5_4”工程。

分析：十进制转换成二进制可通过每次移动一位数字再与“1”进行逻辑“与”运算，依次得到存储在寄存器中的最后一位数据，最后通过串口输出每一位即可。十进制转化为八进制同上。十进制转化成十六进制可通过每次移动四位数字再与“1”进行逻辑“与”运算，依次得到存储在寄存器中的后四位数据，若数据大于 10，则参照 ASCII 表可知字符“10”和“A”相差 55，在“10”的 ASCII 值的基础上加上 55 输出即可，执行结果如图 5-11。

```
//=====================================================================
//文件名称：system_convert.s
//功能概要：STM32L432RC 进制转换（汇编）程序文件
//版权所有：SD-Arm(sumcu.suda.edu.cn)
//版本更新：2019-10-21 V3.0
//=====================================================================
string_system_convert:
    .string "system_convert : "          //转换之前的提示信息
string_system_mesg:
    .string "%d"                         //输出十进制数
string_system_mesg2:
    .string "%c"                         //输出字符形式
```

```
//start  函数定义区域
.type    convert_binary, function
.type    convert_hexade, function
.type    convert_octal, function
//end    函数定义区域
//----------------------以下为内部函数存放处--------------------------------
.section .text
//=========================================================================
//函数名称：convert_binary
//函数返回：无
//参数说明：r5：通过将所要操作的数一次移动 1 位，再与“1”进行逻辑”与“运算后，再存放到 r5 中
//功能概要：将十进制数最后用二进制表示并输出。
//=========================================================================
convert_binary:
//（1.1）保存现场
        push {r0-r7,lr}                    //保存现场，pc(lr)入栈
//（1.2）循环得到寄存器每次移位后的数据，并用串口输出
        mov r2,#8
        mov r4,#1
  loop:
        mov r5,r7                          //r5=r7←逻辑“与”之后的数
        sub r2,r2,#1
        lsr r5,r5,r2
        and r5,r5,r4                       //r5←移位得到数据最后一位数
        mov r6,r2
        mov r1,r5
        ldr r0,=string_system_mesg         //r0←进制转换提示信息
        bl printf                          //调用 printf 函数
        mov r2,r6
        cmp r2,#0
        bne loop                           //r2 不等于 0 则跳转到 loop
//（1.3）恢复现场
        pop {r0-r7,pc}                     //恢复现场，lr 出栈到 pc
//=========================================================================
//函数名称：convert_hexade
//函数返回：无
//参数说明：r5：通过将所要操作的数一次移动 1 位，再与“1”进行逻辑”与“运算后，再存放到 r5 中
//功能概要：将十进制数最后用十六进制表示并输出。
//=========================================================================
convert_hexade:
//（2.1）保存现场
        push {r0-r7,lr}                    //保存现场，pc(lr)入栈
//（2.2）循环得到寄存器每次移位后的数据，并用串口输出
        mov r2,#8                          //r2 指明移位次数
        mov r4,#15                         //与移位后的数据进行逻辑”与“运算
  loop_hexade:
        mov r5,r7
        sub r2,r2,#4
        lsr r5,r5,r2
```

```
        and r5,r5,r4                    //r5←移位得到数据最后四位数
        mov r6,r2
//（2.3）r5 与 10 进行比较，用于输出类似字符 a(十六进制用 10 表示)
        cmp r5,#10
        bcs display_string              //r5≥10 则跳转到 display_string
        mov r1,r5
        cmp r1,#0
        beq loop_hexade                 //r1 不等于 0 则跳转到 loop_hexade
        ldr r0,=string_system_mesg
        bl printf
        mov r2,r6
        cmp r2,#0
        bne loop_hexade                 //r2 不等于 0 则跳转到 loop_hexade
  display_string:                       //转换成字符显示标签
        mov r3,r5
//（2.4）将 r3 得到的 ASCII 码加 55，进而可用 ABCDEF 类似字符表示
        add r3,r3,#55                   //r3←转换成该字母的 ASCII
        mov r1,r3
        ldr r0,=string_system_mesg2
        bl printf
        mov r2,r6
        cmp r2,#0
        bne loop_hexade                 //r2 不等于 0 则跳转到 loop_hexade
//（2.5）恢复现场
        pop {r0-r7,pc}                  //恢复现场，lr 出栈到 pc
//=============================================================================
//函数名称：convert_octal
//函数返回：无
//参数说明：r5：通过将所要操作的数一次移动 3 位，再与“7”进行逻辑”与“运算后，再存放到 r5 中
//功能概要：将十进制数最后用八进制表示并输出。
//=============================================================================
convert_octal:
//（3.1）保存现场
        push {r0-r7,lr}                 //保存现场，pc(lr)入栈
//（3.2）循环得到寄存器每移 3 位后的数据，并用串口输出
        mov r2,#9                       //r2←移位次数
        mov r4,#7
  loop_octal:
        mov r5,r7
        sub r2,r2,#3
        lsr r5,r5,r2
        and r5,r5,r4                    //r5←移位得到数据最后三位数
        mov r1,r5
        mov r6,r2
        cmp r1,#0
        beq loop_octal                  //r1 不等于 0 则跳转到 loop_octal
        ldr r0,=string_system_mesg
        bl printf
        mov r2,r6
```

```
        cmp r2,#0
        bne loop_octal                  //r2 不等于 0 则跳转到 loop_octal
    //（3.3）恢复现场
        pop {r0-r7,pc}                  //恢复现场，lr 出栈到 pc
```

在 main 函数的调用函数的主要程序代码如下：

```
    //（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 xfgi 结束标志
    hello_information:                  //字符串标号
        .ascii "------------------------------------------------------------------\n"
        .ascii "金葫芦提示：\n"
        .ascii "      本工程实现将一个十进制数分别转换为二进制、八进制、十六进制输出！      \n"
        .ascii "      运行结果如下所示！      \n"
        .ascii "------------------------------------------------------------------\n\0"
    data_format:
        .ascii "%d\n\0"                 //printf 使用的数据格式控制符
    string_first_2:                     //字符串标签
        .string "Serial port information start!\ndata : %d\n"
    string_to_binary:                   //串口输出二进制形式提示信息
         .string "binary : "
    string_to_hexade:                   //串口输出十六进制形式提示信息
         .string "\nhexade :    "
    string_to_octal:
         .string "\noctal : "          //输出八进制形式提示信息
    main:
    ……
    //（1.8）将需要转换的数串口输出
         ldr r0,=string_system_convert  //r0←进制转换提示信息
         bl printf                      //调用 printf 函数
         ldr r0,=string_first_2
         mov r7,#95                     //r7←要转换的数
         mov r1,r7
         bl printf
    //（1.8.1）二进制形式输出需要转换的数
         ldr r0,=string_to_binary       //r0←进制转换提示信息
         bl printf
         bl convert_binary              //调用转换成二进制的函数
    //（1.8.2）八进制形式输出需要转换的数
         ldr r0,=string_to_octal
         bl printf
         bl convert_octal               //调用转换成八进制的函数
    //（1.8.3）十六进制形式输出需要转换的数
         ldr r0,=string_to_hexade
         bl printf
         bl convert_hexade              //调用转换成十六进制的函数
    ……
    //（2）======主循环部分（开头）=========================================
    main_loop:                          //主循环标签（开头）
```

```
……
        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）
```

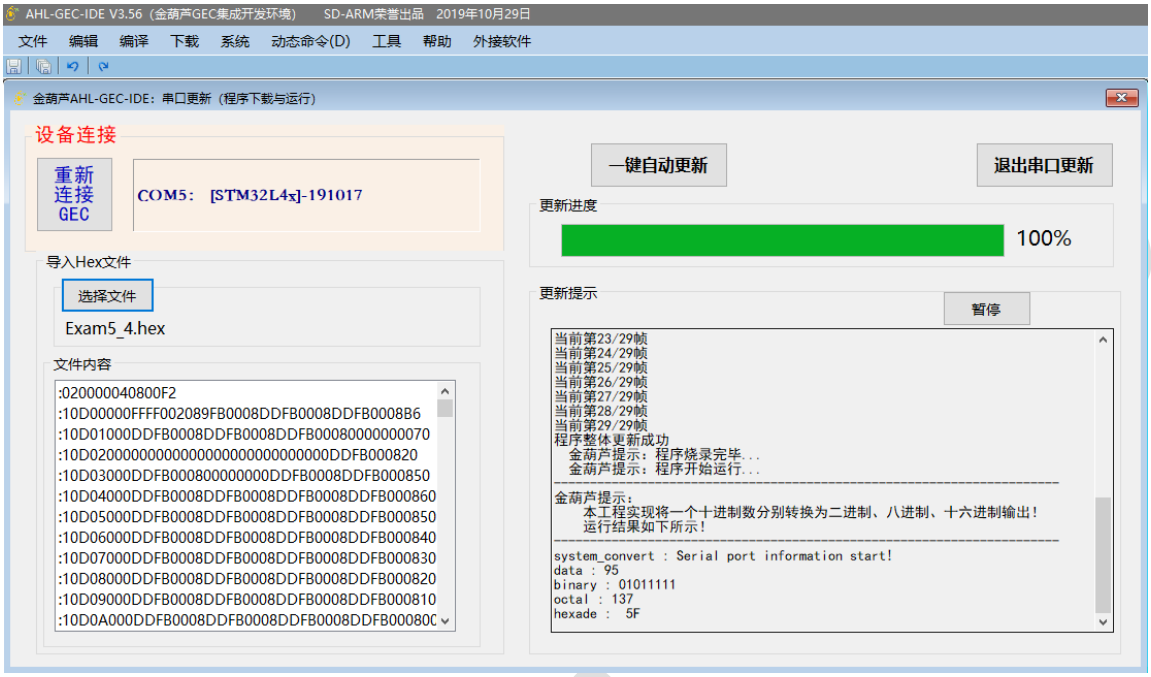

图 5-11 进制转换程序执行结果

5.3.2 冒泡排序程序设计

【例 5-5】输入一组数据，编程实现冒泡递减排序效果，并通过 LCD 屏幕显示输出结果。假设待排序的一组数是：12，15，8，14，16，10，2，30 参考程序见“Exam5_5”工程。

分析：（1）硬件接法：用 SWD 连接目标套件和 PC 机的 USB 端口；LCD 屏幕的 8 个针脚与底板标有“彩色 LCD”字样处相连。

（2）这里需要定义一个数据段来存储数据，可以设定存取每个数据都占用一个字节；排序过程主要分为外循环和内循环两个程序段，每执行完一次内循环，外循环需要循环的次数减 1，当外循环需要循环的次数为 0 时，说明排序完成，此时串口输出提示信息如图 5-12 所示， LCD 屏幕显示输出结果如图 5-13。

```
//=====================================================================================================
//文件名称：bubbleSort.s
//功能概要：STM32L432RC 冒泡排序（汇编）程序文件
//版权所有：SD-Arm(sumcu.suda.edu.cn)
//版本更新：2019-10-21 V3.0
//=====================================================================================================
string_first_2:                    //串口输出前的提示信息
    bubble_uart_bef:
.string "before Sort:"             //冒泡排序前的数的提示信息
    bubble_uart_aft:
.string "\nafter Sort:"            //冒泡排序后的数的提示信息
    string_control:
```

```
.string "%d,"                          //输出十进制数
//start  函数定义区域
.type    bubbleSort, function
//end    函数定义区域
//-----------------------以下为内部函数存放处--------------------------------
.section .text
//=============================================================================================
//函数名称：bubbleSort
//函数返回：无
//参数说明：r2:用于存储数据的首地址，r6:用于控制冒泡排序外循环的次数
//功能概要：用串口的方法输出冒泡排序前的数据和冒泡排序后的数据。
//备注：这里需要用户自己手动输入数据，先获取存储首地址，在按数据所需存储单元一步步
//              存入数据。
//=============================================================================================
bubbleSort:
//（1）保存现场
      push {r0-r7,lr}                  //保存现场，pc(lr)入栈
//（2）排序前，先在 LCD 屏幕上显示字符串信息
      mov r0,#6                        //r0←需要显示在 LCD 屏幕上的 x 坐标初始值
      mov r1,#45                       //r2←需要显示在 LCD 屏幕上的 y 坐标
      ldr r2,=bubble_uart_bef          //r2 指明排序前的字符串信息
      bl LCD_showScreen_string         //调用 LCD 屏幕显示字符串的函数
//（3）依次输出冒泡排序前的数据
      mov r7,#0                        //r7←需要移动的相对地址数
      loop_bub_bef:
//（3.1）获取数组的首地址
      ldr r2,=array                    //r2←获取数组的首地址
      mov r0,#65
      ldr r1,[r2,r7]                   //r1=r2←首地址+r7 后的数据
      ldr r3,=0x000000FF
      and r1,r1,r3                     //依次取出数据
      bl LCD_showScreen_digital        //调用 LCD 显示数字的函数
//（3.2）每次取完数后，地址+1
      add r7,r7,#1
      ldr r6,=count
      cmp r7,r6
      bcc loop_bub_bef                  //若 r7<数组长度，则跳转 loop_bub_bef
//（4）冒泡排序一趟，外层循环次数减 1
      loop_outer:                      //外层循环控制标签
      ldr r2,=array
      sub r6,r6,#1
      cmp r6,#0
      beq end                          //跳转到程序执行处
      mov r5,#0                        //r5←记录当前外层循环的次数
//（5）内层循环用于比较两数并确定是否需要交换位置
      loop_inner:                      //内层循环控制标签
      ldr r3,[r2]                      //r3←所要比较的第一个数据
      ldr r0,=0x000000FF
      and r3,r3,r0                     //从存储地址取第一个数
```

```
        add r4,r2,#1
        ldr r4,[r4]                          //r4←所要比较的第二个数据
        ldr r0,=0x000000FF
        and r4,r4,r0                         //从存储地址取第二个数
        cmp r3,r4
    //（5.1）若 r3≥r4 则跳转到 noSwap
        bcs noSwap
    //（5.2）若 r3<r4 则无需交换存储位置
        add r0,r2,#1
        strb r3,[r0]                         //以一个字节方式存储
        strb r4,[r2]                         //r3＜r4 则交换存储位置
        noSwap:                              //无需交换存储位置标签
        add r2,r2,#1
        add r5,r5,#1                         //r5←外层循环次数+1
        cmp r5,r6                            //比较记录的外层循环次数与实际所需的外层循环次数
        bcc loop_inner                       //小于实际所需的外层循环次数则跳转到 loop_inner
        b   loop_outer                       //跳转到外层循环
        end:
     //（6）排序后，在 LCD 屏幕上显示字符串信息
        ldr r0,=bubble_uart_aft              //r0←排序后的提示信息
        mov r0,#6
        mov r1,#85
        ldr r2,=bubble_uart_aft
        bl LCD_showScreen_string
    //（7）依次输出冒泡排序后的数据
        mov r7,#0                            //r7←需要移动的相对地址数
        loop_bub_aft:                        //串口输出数据提示信息前的显示标签
        ldr r2,=array
        ldr r1,[r2,r7]                       // r1=r2←首地址+r7
        ldr r3,=0x000000FF
        and r1,r1,r3
        mov r0,#105
        bl LCD_showScreen_digital
        add r7,r7,#1
        ldr r6,=count
        cmp r7,r6
        bcc loop_bub_aft                     //跳转 loop_bub_aft
    //（8）恢复现场
        pop {r0-r7,pc}                       //恢复现场，lr 出栈到 pc
```

其中函数 LCD_showScreen_string、LCD_showScreen_digital 和 main 函数中定义 LCD_showTitle 都是 LCD 构件内部函数，具体请参考“Exam5_5”工程 05_UserBoard 下的 lcd.c 文件。在 main 函数的调用函数的主要程序代码如下：

```
    //（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 xfgi 结束标志
    hello_information:                       //字符串标号
        .ascii "----------------------------------------------------------------\n"
```

```
    .ascii "金葫芦提示：\n"
    .ascii "      本工程实现将一组数进行冒泡递减排序，并将结果显示在 LCD 屏幕中！      \n"
    .ascii "      运行结果请参照 LCD 屏幕显示！      \n"
    .ascii "-------------------------------------------------------------------\n\0"
string_first_2:                          //字符串标签
    .string "Serial port information start\n"
……
//（0.1.3）定义数组
.global array,count                      //声明所定义的数组和数组长度是全局变量
.section .data
.align 1
array:                                   //定义需排序的数组
  .byte 12,15,8,14,16,10,2
.align 1
.equ count,7                             //count=数组长度，用于记录外循环次数
……
main:
……
    bl LCD_Init                          //调用 LCD 初始化函数
    ldr r0,=hello_information            //r0←待显示字符串
    bl   printf                          //调用 printf 显示字符串
//（1.9）调用冒泡排序函数
    ldr r0,=string_first_2               //r0←串口输出数据前的提示信息
    bl LCD_showTitle                     //调用显示头标题的函数
    bl bubbleSort                        //调用冒泡排序函数
//（2）======主循环部分（开头）=========================================
main_loop:                               //主循环标签（开头）
……
        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）
```

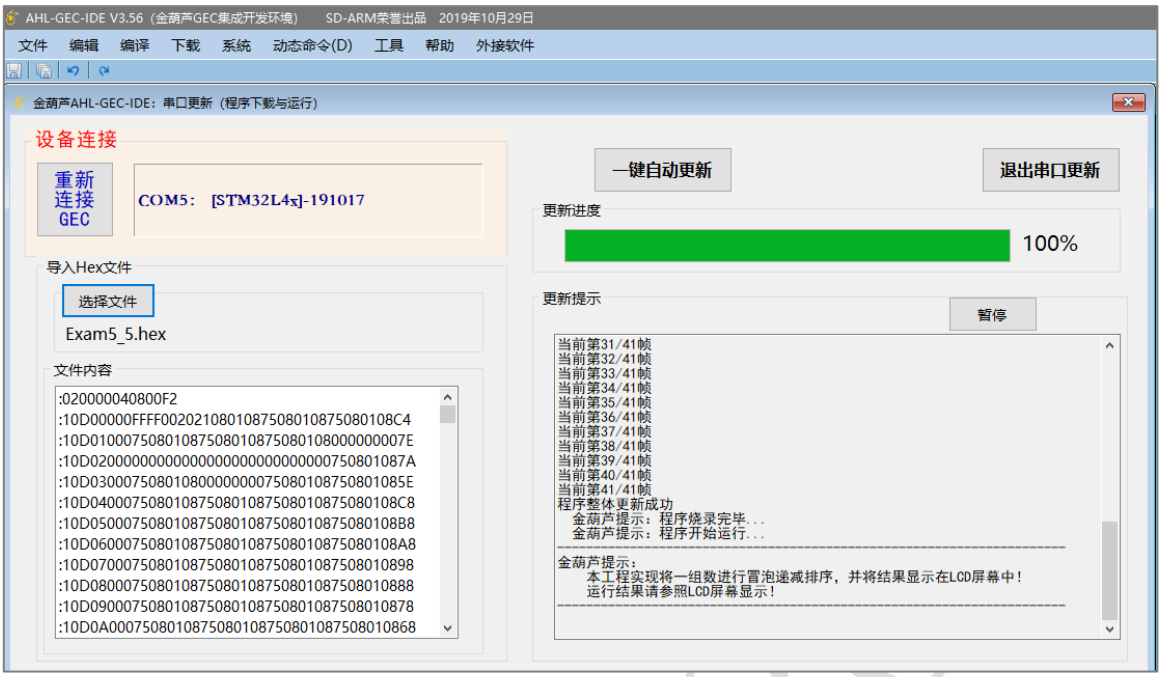

图 5-12 冒泡排序串口输出提示信息

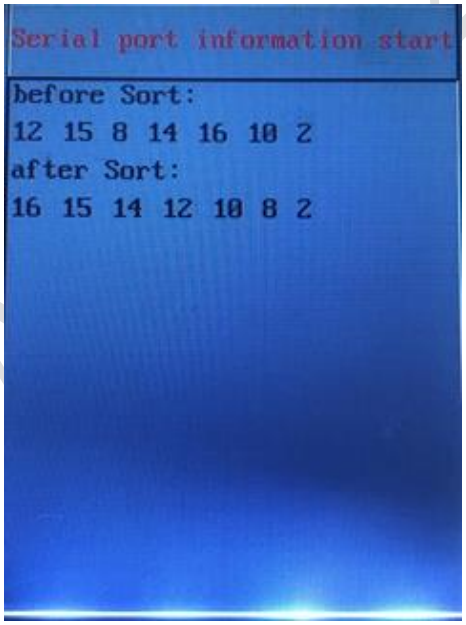

图 5-13 LCD 屏幕显示冒泡排序程序执行结果

5.4 实验二：基于构件方法的汇编程序设计

1．实验目的

（1）对构件基本应用方法有更进一步的认识，初步掌握基于构件的汇编设计与运行。

（2）理解汇编语言中顺序结构、分支结构和循环结构的程序设计方法。

（3）理解和掌握汇编跳转指令的使用方法和场合。

（4）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。

2．实验准备

参见实验一。

3．参考样例

参照“Exam5_5”工程。该程序通过申请一段绝对地址空间用来存储一组数据，并通过编程实现冒泡从大到小的排序，最后用串口输出排序后的结果。

4．实验过程或要求

（1）验证性实验

验证样例程序，具体验证步骤见实验一。

（2）设计性实验

在验证性实验的基础上，自行编程实现 100 以内的奇数相加所得到的和，最后通过串口输出该结果。

（3）进阶实验★

利用“Exam5_5”工程提供的一组数据，也可自行定义一组数据，采用**选择排序**算法对这组数据进行从小到大排序。

5．实验报告要求

（1）用适当文字、图表描述实验过程。

（2）用 200~300 字写出实验体会。

（3）在实验报告中完成实践性问答题。

6．实践性问答题

（1）若要通过串口输出冒泡排序结果，该如何实现。

（2）若要实现冒泡排序从小到大的顺序，则应修改哪些语句。

5.5 习题

1．构件在设计的过程中应遵循哪些基本原则。

2．给出汇编语言主程序文件 main.s 的基本结构。

3．给出汇编语言中断服务程序文件 isr.s 的基本结构。

4．读程序：

```
mov r2,#97
sub r2,r2,#32
mov r1,r2
ldr r0,=string_test
bl printf
```

其中 printf 函数用于输出数据；string_test 的定义如下：

```
string_test:
    .string " %c"
```

请问：上述程序实现了什么功能。

5．编程实现二进制与十六进制之间的转换。

6．查找文件：NATO Communications and Information Systems Agency. NATO Standard for Development of

Reusable Software Components, 1991。认真阅读该文件，从中总结出 500 字左右的要点。