第4章 汇编语言框架（一）

4.1初识程序运行

4.2汇编工程框架及执行工程分析

汇编语言通常被应用在诸如芯片启动过程、硬件底层驱动程序、操作系统及程序运行实时性较高场合。若能够从底层透明理解微型计算机运行的基本原理，利用汇编语言进行基本程序设计是基本功。不少人认为汇编语言难以掌握，实际上只要掌握基本方法，规范编程，勤于实践，自然就不难了。

4.1初识程序运行

为了降低学习难度，这里首先运行一个汇编语言模板，了解运行过程，随后再阐述主要知识点。

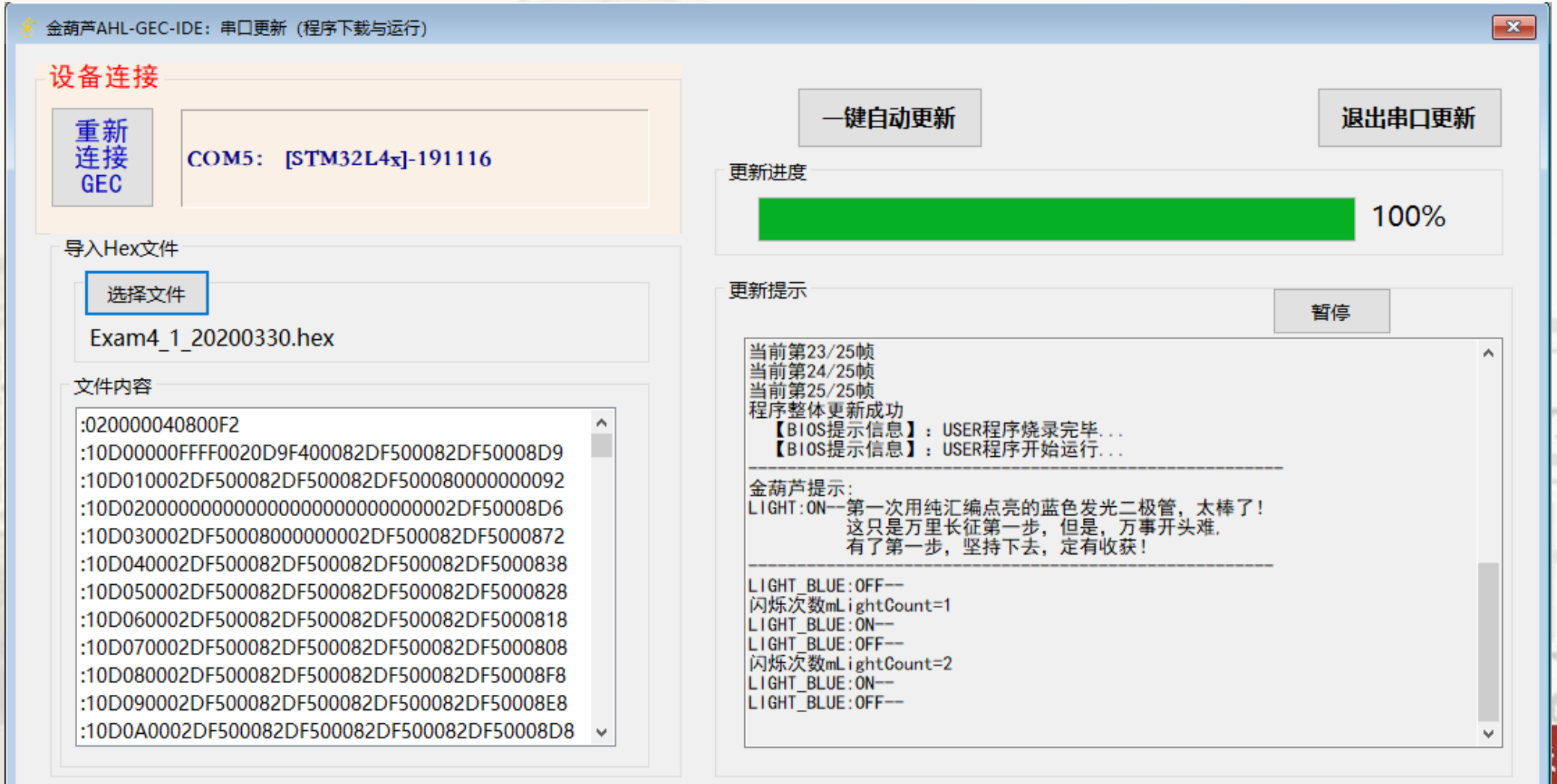
1. 工程框架实现的方法

电子资源下载：http://sumcu.suda.edu.cn/wjyl/main.htm
开发环境下载：http://sumcu.suda.edu.cn/AHLwGECwIDE/list.htm
（使用指南见本书附录B)

样例程序：电子资源“..\\04-Software\Exam4_1”
功能：（1）主程序：（理解该部分程序）
红灯闪烁，红灯闪烁次数通过printf输出到PC机屏幕
（2）串口中断处理程序：（仅演示功能，第7章讲解）

工程Exam4_1具体运行结果（利用实际工程分析）

主循环中的屏幕输出（通过调试串口UART_Debug）



用户串口（UART_User）的接收中断处理程序：收到一个字节，回送一个字节

【硬件接线： 板上UART_User，黑线-地、白线-TX、绿线-RX】（仅演示功能）

2. main函数完整代码注释（随后边运行、边打桩，重点理解）

include.inc文件：

```
.include "user.inc"
//宏定义常数
.equ MainLoopNUM,1122338  //主循环次数设定值（常量）
```

```
//=====================================================================

//文件名称：main.s
//功能概要：汇编编程调用GPIO构件控制小灯闪烁（利用printf输出提示信息）
//版权所有：SD-ARM(sumcu.suda.edu.cn)
//版本更新：20180810-20191018
//=====================================================================
.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
```

2. main函数完整代码注释（边运行、边打桩，重点理解）

```
//（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 "金葫芦提示:                                      \n"
    .ascii "LIGHT:ON--第一次用纯汇编点亮的蓝色发光二极管，太棒了！  \n"
    .ascii "          这只是万里长征第一步，但是，万事开头难,      \n"
    .ascii "          有了第一步，坚持下去，定有收获！            \n"
    .ascii "-------------------------------------------------\n\0"
data_format:
    .ascii "%d\n\0"           //printf使用的数据格式控制符
light_show1:
    .ascii "LIGHT_BLUE:ON--\n\0"   //灯亮状态提示
light_show2:
    .ascii "LIGHT_BLUE:OFF--\n\0"  //灯暗状态提示
light_show3:
    .ascii "闪烁次数mLightCount=\0"  //闪烁次数提示
```

2. main函数完整代码注释（边运行、边打桩，重点理解）

```
//（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
        .word  0
mFlag:                          //定义灯的状态标志,1为亮，0为暗
        .byte 'A'
.align 4
mLightCount:
        .word  0
```

```
//（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
```

```
//_____________________________________________
//主函数，一般情况下可以认为程序从此开始运行（实际上有启动过程，参见书稿）
main:
//（1）======启动部分（开头）主循环前的初始化工作======================
//（1.1）声明main函数使用的局部变量

//（1.2）【不变】关总中断
        cpsid i
//（1.3）给主函数使用的局部变量赋初值

//（1.4）给全局变量赋初值
```

```
//（1.5）用户外设模块初始化
// 初始化蓝灯, r0、r1、r2是gpio_init的入口参数
        ldr r0,=LIGHT_BLUE     //r0指明端口和引脚（用=,因常量>=256,需用ldr）
        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亮
        bl  gpio_init          //调用gpio初始化函数
// 初始化串口UART_User1
        mov r0,#UART_User      //串口号
        ldr r1,=UART_BAUD      //波特率
        bl uart_init           //调用uart初始化函数
```

```
//（1.6）使能模块中断
    mov r0,#UART_User       //串口号
    bl  uart_enable_re_int  //调用uart中断使能函数

//（1.7）【不变】开总中断
        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

        //bl .  //在此打桩(.表示当前地址)，理解发光二极管为何亮起来了？

//（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]
        add r1,#1
        str r1,[r2]
//（2.2）未达到主循环次数设定值，继续循环
        ldr r2,=MainLoopNUM
        cmp r1,r2
        blO  main_loop                  //未达到，继续循环
//（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
        ldr r2,=mMainLoopCount     //r2←mMainLoopCount的地址
        mov r1,#0
        str r1,[r2]
```

```
//（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]
        add r1,#1
        str r1,[r2]
//（2.2）未达到主循环次数设定值，继续循环
        ldr r2,=MainLoopNUM
        cmp r1,r2
        blO  main_loop                  //未达到，继续循环
//（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
        ldr r2,=mMainLoopCount     //r2←mMainLoopCount的地址
        mov r1,#0
        str r1,[r2]
```

```
//（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
//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
//mFlag等于'L'情况
        ldr r3,=mLightCount             //灯的闪烁次数mLightCount+1
        ldr r1,[r3]
        add r1,#1
        str r1,[r3]
        ldr r0,=light_show3       //显示“灯的闪烁次数mLightCount=”
        bl  printf
        ldr r0,=data_format       //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'
        mov r7,#'A'
```

```
        str r7,[r2]
        ldr r0,=LIGHT_BLUE      //亮灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show1                //显示灯亮提示
        bl  printf
//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
        main_light_off:
        ldr r2,=mFlag                       //灯的状态标志改为'L'
        mov r7,#'L'
        str r7,[r2]
        ldr r0,=LIGHT_BLUE          //暗灯
        ldr r1,=LIGHT_OFF
        bl  gpio_set
        ldr r0, =light_show2            //显示灯暗提示
        bl  printf
```

3．有关main函数中相关指令的说明

1）字符串的定义方法

.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”；

.asciz：这种方式定义的字符串会自动在末尾添加“\0”；

.string：这种方式定义的字符串会自动在末尾添加“\0”。

为了便于下一次显示的内容能从新的一行开始，一般在定义字符串时会在其末尾加上“\n”，起到回车换行的作用。第一种方法常用于一次性输出多行字符串的场合，而第二、三种方法一般用于输入一行字符串的情况。

```
light_show1:
        .ascii  "LIGHT_BLUE:ON--\n\0"      //第一种方法定义字符串，要自行添加\0
light_show2:
        .asciz  "LIGHT_BLUE:ON--\n"        //与第一种方法的效果一样
light_show3:
        .string  "LIGHT_BLUE:ON--\n"       //与第一种方法的效果一样
```

【练习4-1】上机练习将工程中的light_show2标号对应的字符串改为用asciz定义，将light_show3标号对应的字符串改为用string定义。

2）变量的定义

在程序中不可避免的会使用到变量，在汇编中变量名实际上是一个标号，它代表了变量的地址，变量的类型有字、半字、字节等。因此，当要同时定义多种不同类型的变量时，要在变量定义前指明对齐格式，否则会影响变量的存储。

如何获得变量的值或将新的值存入变量中呢？一般是先将的变量的地址存入寄存器中，然后通过寄存器间接寻址方式取出变量的值或将值存入变量中。

```
ldr  r2,=mMainLoopCount    //r2←变量mMainLoopCount的地址
ldr  r1, [r2]              //取变量的值
add  r1,#1
str  r1,[r2]               //新的值存入变量中
```

【练习4-2】上机练习自行定义一个半字类型的数据，并在主程序中将它输出。

3）常量的定义

为了使程序能有更好的可移植性，有时会在汇编中定义一些常量，它相当于C语言的宏定义，可以使用“.equ”或“.set”来定义常量。常量有两种使用方法：一种是当常量小于或等于256时，在常量前加“#”；另一种是当常量大于256时，在常量前加“=”。

4）函数调用方法

在汇编语言中，对参数有如下规定：当参数个数小于或等于4个时，是通过寄存器r0～r3来传递参数；当参数超过4个时，超过的参数可以使用堆栈来传递参数，但入栈的顺序要与参数的顺序相反；所有参数被看作是存放在连续的内存字单元的字数据。

（1）printf函数的调用用法

对于字符串的输出，只需将待显示字符串的首地址作为参数存入r0，然后再通过bl指令调用printf函数，就可以显示提示信息了；对于需要显示数值或地址值，则需要将数据显示格式控制符%d（十进制格式）或%x（十六进制格式）作为第一个参数存入r0，把待显示的数值或地址值作为第二个参数存入r1，然后再通过bl指令调用printf函数，就可以显示值了。

（2）GPIO构件的调用方法

在本程序中涉及到GPIO构件中的初始化函数gpio_init和灯的设置函数gpio_set。

```
ldr   r0,=LIGHT_BLUE      //r0←端口和引脚（用=，是因为常量>=256，且要用ldr指令）
mov  r1,#GPIO_OUTPUT   //r1←引脚方向为输出
mov  r2,#LIGHT_ON        //r2←引脚的初始状态为亮
bl    gpio_init                 //调用gpio初始化函数
ldr   r0,=LIGHT_BLUE      //亮灯
ldr   r1,=LIGHT_ON
bl    gpio_set
```

5）打桩调试

在汇编语言程序编写过程中，难免会出现一些语法和语义上的错误，使得程序在汇编时通不过。为了能快速找到错误，常用采用打桩调试的方法来定位排除，即在可能出错的语句前加上“bl .”指令（“.”表示当前指令的地址，该指令相当于高级语言中的无限死循环）进行打桩。当执行到该指令时，程序就会停止下来，此时就可以观察程序的执行情况，以此来判断程序的错误。

bl . //根据需要可以在不同位置进行打桩(.表示当前地址)，打桩调试后要将该指令删除

4. 有关头文件的说明

1）总头文件

总头文件include.inc位于07_NosPrg文件夹下，它只包含了user.inc一个头文件。

2）用户头文件

用户头文件 user.inc位于05_UserBoard文件夹下，它包含了各类与引脚有关、外设相关参数有关的常定义，是为了主程序和中断服务程序的可移植性而设置的。

3）其他头文件

其它头文件包括与MCU相关的底层硬件构件的头文件、与目标板相关的应用构件的头文件以扩与应用相关的软件构件头文件的，各类构件头文件要按照“分门别类，各有归处”的原则进行存放。

4.2汇编工程框架及执行工程分析

本节以“Exam4_1”工程为例，阐述工程的组织及执行过程，该工程组织与芯片及开发环境无关。

嵌入式系统工程包含若干文件，包括程序文件、头文件、与调试相关的文件、工程说明文件、开发环境生成文件等，文件众多，合理组织这些文件，规范工程组织，可以提高项目的开发效率、提高阅读清晰度、提高可维护性、降低维护难度。工程组织应体现嵌入式软件工程的基本原则与基本思想。这个工程框架也可被称为软件最小系统框架，因为它包含的工程的最基本要素。软件最小系统框架是一个能够点亮一个发光二管的，甚至带有串口调试构件的，包含工程规范完整要素的可移植与可复用的工程模板。

4.2.1汇编工程框架的基本内容

下图给出以“Exam4_1”工程为例的树形工程结构模板，物理组织与逻辑组织一致。该模板是苏州大学嵌入式中心为在STM32CubeIDE1.0.0环境下开发，采用Arm Cortex-M4F系列MCU为应用工程而设计的。

01_Doc	<文档文件夹>
02_CPU	<内核相关文件>
03_MCU	<MCU 相关文件夹>
04_GEC	<GEC 芯片相关文件>
05_UserBoard	<应用构件文件夹>
06_SoftComponent	<软件构件文件夹>
07_NosPrg	<无操作系统工程主程序文件夹>

1. 工程名与新建工程

不必在意工程名，而使用工程文件夹来标识工程，不同工程文件夹就区别不同工程。这样，工程文件夹内的文件中所含的工程名字就不再具有标识意义，可以修改，也可以不修改。建议新工程文件夹使用手动复制标准模板工程文件夹的方法来建立，这样复用的构件已经存在，框架保留，体系清晰。不推荐使用STM32CubeIDE1.0.0或其他开发环境的新建功能来建立一个新工程。

2. 工程文件夹内的基本内容

工程文件夹内编号的共含7个下级文件夹，除去STM32CubeIDE1.0.0环境保留的文件夹Includes与Debug，分别是01_Doc、02_Core、03_MCU、04_ GEC、05_ UserBoard、06_ SoftComonnt、07_NoPrg，各文件夹的含义、简明功能及特点如下表所示。

表4-1 工程文件夹内的基本内容

名称	文件夹		简明功能及特点
文档文件夹	01_Doc		工程改动时，及时记录。
CPU 文件夹	02_CPU		与内核相关的文件。
MCU 文件夹	03_MCU	linker_File	链接文件夹，存放链接文件。
		MCU_drivers	MCU 底层构件文件夹，存放芯片级硬件驱动。
		startup	启动文件夹，存放芯片头文件及芯片初始化文件。
GEC 相关文件夹	04_GEC		GEC 芯片相关文件夹，存放引脚头文件。
用户板文件夹	05_UserBoard		用户板文件夹，存放应用级硬件驱动，即应用构件。
软件构件文件夹	06_SoftComponent		抽象软件构件文件夹，存放硬件不直接相关的软件构件。
无操作系统源程序文件夹	07_NosPrg	include.inc	总头文件，包含各类宏定义。
		isr.s	中断服务程序文件，存放各中断服务程序子函数。
		main.s	主程序文件，存放芯片启动的入口函数 main。

3. CPU（内核）相关文件简介

CPU（内核）相关文件包括内核外设访问层头文件、编译器头文件、Cortex-M SIMD 头文件和微控制器软件接口标准头文件等，位于工程框架的“..\02_CPU”文件夹内。

4. MCU（芯片）相关文件简介

MCU文件夹（芯片）相关文件包括芯片启动文件、芯片头文件和系统初始化文件等，位于工程框架的“..\03_MCU\ startup”文件夹内。

芯片头文件给出了芯片专用的寄存器地址映射，设计面向直接硬件操作的底层驱动时，利用该文件使用映射寄存器名，获得对应地址。

启动文件主要完成复位MCU、从Flash中将已初始化的数据复制到RAM中、清零bss段数据、初始化系统时钟、初始化标准库函数，最后转到main函数执行。

系统初始化文件主要完成复位RCC时钟配置、中断向量表的设置、初始化系统时钟等任务。

5. 应用程序源代码文件

在工程框架的“..\ 07_NosPrg”文件夹内放置着总头文件include.inc、main.s及中断服务程序isr.s。

总头文件include.inc是main.s使用的头文件，内含常量、全局变量声明、外部函数及外部变量的引用。

主程序文件main.s是应用程序启动的总入口，main函数即在该文件中实现。在main函数中包含了一个永久循环，对具体功能的实现代码一般都添加在该主循环中。

中断服务程序文件isr.s是中断处理函数编程的地方。

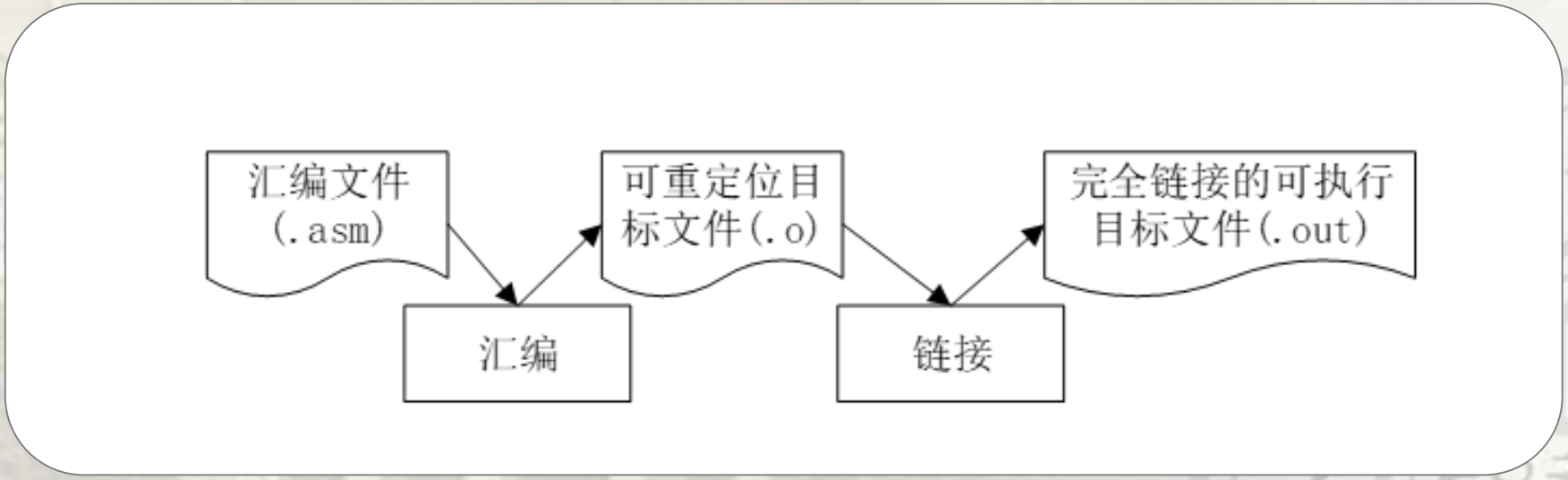
6. 编译链接产生的其他相关文件简介

可执行链接格式文件（.elf）映像文件（.map）与列表文件（.lst）位于“..\Debug”文件夹，由编译链接产生。

4.2.2链接脚本文件的作用

链接脚本文件（简称链接文件）是用于控制链接的过程，规定了如何把输入的中间文件中的section映射到最终目标文件内，并控制目标文件内各部分的地址分配。它为链接器提供链接脚本，是以.ld或.lds为扩展名的文件。

程序编译链接的过程：



4.2.3机器码解析

1. 记录格式

.hex文件中的语句有六种不同类型的语句，但总体格式是一样的。

	字段1	字段2	字段3	字段4	字段5	字段6
名称	记录标记	记录长度	偏移量	记录类型	数据/信息区	校验和
长度	1字节	1字节	2字节	1字节	N字节	1字节
内容	开始标记“:”		数据类型记录有效；非数据类型，该字段为“0000”	00-数据记录； 01-文件结束记录； 02-扩展段地址； 03-开始段地址； 04-扩展线性地址； 05-链接开始地址。	取决于记录类型	开始标记之后字段的所有字节之和的补码。 校验和=0xFF-(记录长度+记录偏移+记录类型+数据段)+0x01

2. 实例分析

行	记录标记	记录长度	偏移量	记录类型	数据/信息区	校验和
1	:	02	0000	04	0800	F2
2	:	10	D000	00	FFFF002071FD0008C5FD0008C5FD0008	F8
626	:	10	FD70	00	DFF838D0002103E00D4B5B5843500431	CD
666	:	00	0000	01		FF

（1）第1行，以“：”开始，02表示长度为2字节，“0000”表示相对地址，紧接着的“04”代表记录类型为扩展线性地址，“0800”与“0000”组成“08000000”代表代码段在Flash的起始地址，“F2”为校验和。

（2）第2行，以“：”开始，长度为“0x10”（16个字节），“D000”表示偏移量，紧接着的“00”代表记录类型为数据类型，接下来的数据段是存放在偏移地址为“D000”的存储区的机器操作码中，也就是说，只有这些数据被写入到Flash存储区。

（3）第666行（最后一行）为文档的结束记录，记录类型为“0x01”；“0xFF”为本记录的校验和字段内容。

【练习4-3】打开“Exam4_1.hex”文件，根据实例分析情况验证一下。

4.2.4执行过程分析

系统程序的运行过程可分为两部分，即main函数之前的运行和main函数的运行。

main函数的运行过程：

首先进入main函数后先对所用到的模块进行初始化，比如小灯端口引脚的初始化，然后进入main_loop函数，在该函数中首先把一个延时数MainLoopNUM存储到寄存器r2中，该延时数用于控制小灯的闪烁频率，可在单步调试中把它改成较小的值。接着使寄存器r1从零开始递增，每次加1并且同时和寄存器r2中的值比较，如果两个寄存器中的值相等，则调用小灯亮暗转变函数，然后继续运行main_loop，否则寄存器r1的值继续递增直到和r2寄存器中的值相等为止。

最后当某个中断发生后，MCU将转到中断服务程序文件isr.s所指定的中断入口地址处，开始运行中断服务程序ISR。例如，在工程的（UARTA接收中断服务程序）中断服务程序，当UARTA收到一个字节，程序便会跳转至UARTA_Handler中断函数服务例程执行，待执行完中断函数服务例程后，再跳转至main函数继续执行。（此处只演示）

本讲小结：

1) 理解汇编程序结构
2) 掌握汇编程序的常量、变量、函数定义方法
3) 掌握汇编程序的函数调用方法，理解参数传递方式
4) 掌握汇编程序的分支、循环构成方式
5) 学习教材的工程结构，学习自己构件工程

作业：1~6