第 3 章　程序设计基础

GNU 集成编译环境(GNU Compiler Collection,GCC)是一种面向嵌入式领域,支持多种编程语言,支持多种 CPU 的交叉编译工具。

本章主要内容如下:

(1) GCC 编译过程;

(2) C/C++ 交叉编译器 arm-elf-gcc;

(3) 交叉汇编器 arm-elf-as;

(4) 交叉连接器 arm-elf-ld;

(5) 工程管理器 make;

(6) 汇编语言编程;

(7) 混合编程。

3.1　GCC 简介

GCC 是一套面向嵌入式领域的交叉编译工具,支持多种编程语言、多种优化选项并且能够支持分步编译、支持多种反汇编方式、支持多种调试信息格式,目前支持 x86、ARM7、StrongARM、PPC4xx、MPC8xx 和 MIPS R3000 等多种 CPU。它的基本功能包括以下几点:

(1) 输出预处理后的 C/C++ 源程序(展开头文件和替换宏);

(2) 输出 C/C++ 源程序的汇编代码;

(3) 输出二进制目标文件;

(4) 生成静态库;

(5) 生成可执行程序;

(6) 转换文件格式。

3.1.1　GCC 组成

1. C/C++ 交叉编译器 arm-elf-gcc

arm-elf-gcc 是编译的前端程序,它通过调用其他程序来实现将程序源文件编译成目标文件的功能。

编译时,它首先调用预处理程序(cpp)对输入的源程序进行处理,然后调用 cc1 将预处理后的程序编译成汇编代码,最后由 arm-elf-as 将汇编代码编译成目标代码。

arm-elf-gcc 具有丰富的命令选项,可以控制编译的各个阶段,满足用户的各种编译需求。

2. 汇编器 arm-elf-as

arm-elf-as 将汇编语言程序转换为 ELF(Executable and Linking Format,执行时链接

文件格式)格式的可重定位目标代码,这些目标代码同其他目标模块或函数库易于定位和链接。

arm-elf-as 产生一个交叉参考表和一个标准的符号表,产生的代码和数据能够放在多个区 (Section)中。

3. 连接器 arm-elf-ld

arm-elf-ld 根据链接定位文件 Linkcmds 中的代码区、数据区、BSS 区和栈区等定位信息,将可重定位的目标模块链接成一个单一的、绝对定位的目标程序。该目标程序是 ELF 格式,并且可以包含调试信息。

arm-elf-ld 会产生一个内存映像文件 Map. txt,该文件显示所有目标模块、区和符号的绝对定位地址。它也产生交叉参考列表,显示参考每个全局符号的目标模块。

arm-elf-ld 支持将多个目标模块链接成一个单一的、绝对定位的目标程序,也能够依此对目标模块进行链接,这个特性称为增量链接(Incremental Linking)。假如输入文件是一个函数库,arm-elf-ld 会自动从函数库装载被其他目标模块参考的函数模块。

arm-elf-ld 与其他链接程序相比,能提供更有帮助的诊断信息。许多链接器遇到第一个错误即放弃链接,而 arm-elf-ld 只要有可能都继续执行,帮助用户识别其他错误,有时甚至能获得输出代码。

4. 库管理器 arm-elf-ar

arm-elf-ar 将多个可重定位的目标模块归档为一个函数库文件。采用函数库文件,应用程序能够从该文件中自动装载要参考的函数模块,同时将应用程序中频繁调用的函数放入函数库文件中,易于应用程序的开发管理。arm-elf-ar 支持 ELF 格式的函数库文件。

5. 工程管理器 make

make 是用于自动编译、链接程序的实用工具,使用 make 后就不需要手工地编译每个程序文件。要使用 make,首先要编写 makefile。makefile 描述程序文件之间的依赖关系,并提供更新文件的命令。在一个程序中,可执行文件依赖于目标文件,而目标文件依赖于源文件。如果 makefile 文件存在,每次修改完源程序后,用户通常所需要做的事情就是在命令行输入“make”,然后所有的事情都由 make 来完成。

6. 其他实用程序

目标文件格式转换工具 arm-elf-objcopy 支持的文件格式有 H-record、S-record、ABS、BIN、COFF 和 ELF。例如,它能够将 ELF 格式文件转换为其他格式的文件,如 Intel H-record 格式和 Motorola S-record 等。arm-elf-nm 程序用于显示文件中的符号信息。

3.1.2 GCC 编译程序的基本过程

GCC 编译程序的基本过程如下。

(1) arm-elf-gcc 根据输入文件的后缀来确定文件的类型,然后根据用户的编译选项(包括优化选项、调试信息选项等)将其编译成相应的汇编临时文件(后缀为. s)。

(2) arm-elf-as 将该汇编文件编译成目标文件(后缀为. o)。

(3) arm-elf-ld 根据用户的链接选项(包括指定链接命令文件等)将目标文件和各种库链接起来生成可执行文件。图 3-1 展示了该编译过程。

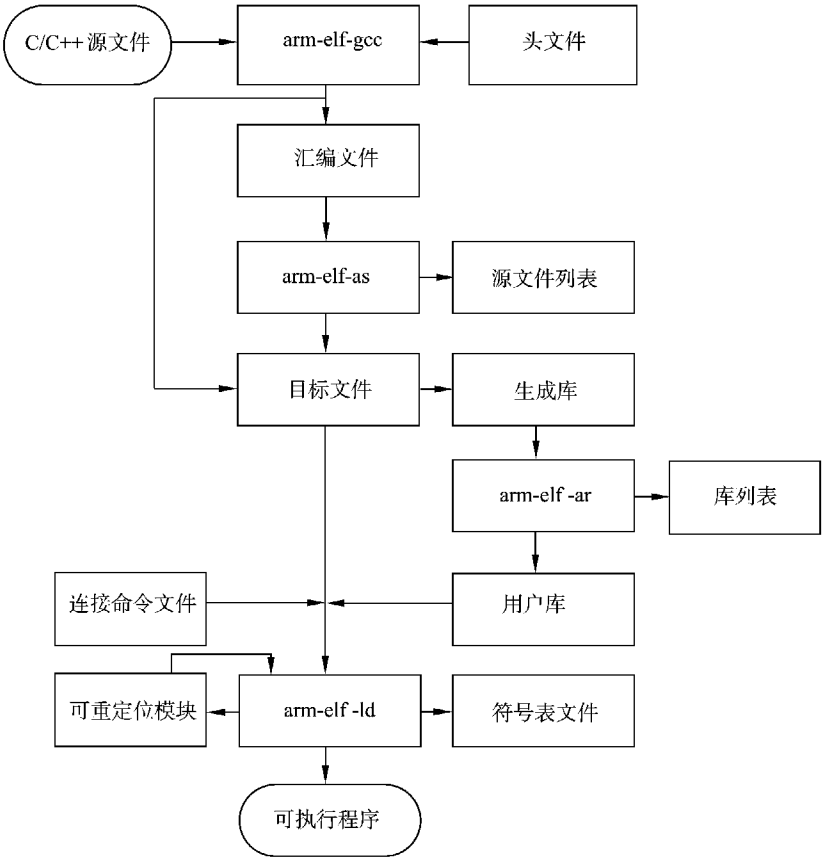

图 3-1　程序的编译过程

3.2　C/C++ 交叉编译器 arm-elf-gcc

3.2.1　概述

arm-elf-gcc 是编译的前端程序。它通过调用其他程序来实现将程序源文件编译成目标文件。编译时它首先调用预处理程序(cpp)对输入的源程序进行处理,然后调用 cc1 将预处理后的程序编译成汇编代码,最后由 arm-elf-as 将汇编代码编译成目标代码。

arm-elf-gcc 具有丰富的命令选项,控制编译的各个阶段,满足用户的各种编译需求。

1. 命令格式

```
arm-elf-gcc [options] file …
```

在命令 arm-elf-gcc 后面跟一个或多个选项,选项间用空格隔开,然后跟一个或多个目标文件。

例如,将 test.c 编译成目标文件 test.o 并且生成调试信息:

```
arm-elf-gcc  -g  -c  -o test.o test.c
```

2. 命令选项列表

(1) 输出控制选项如下。

-c：将输入的源文件编译成目标文件。

-S：将 C/C++ 文件生成汇编文件。

-o file：将输出内容存于文件 file。

-pipe：在编译的不同阶段之间采用管道通信方式。

-v：打印出编译过程中执行的命令。

-x language：说明文件的输入类型为 language。

(2) C 语言选项如下。

-ansi：支持所有 ANSI C 程序。

(3) 警告选项如下。

-w：关闭所有警告。

-Wall：打开所有警告。

-Wimplicit：如果有隐含申明，显示警告信息。

-Wno-implicit：不显示对隐含申明的警告。

(4) 调试选项如下。

-g：在文件中产生调试信息(调试信息的文件格式有 stabs、COFF、XCOFF、DWARF)。

(5) 优化选项如下。

-O0：不优化。

-O1：一级优化。

-O2：二级优化。

-O3：三级优化。

(6) 预处理选项如下。

-E：运行 C 的预处理器。

-C：在运用-E 进行预处理时不去掉注释。

-D macro：定义宏 macro 为 1。

-D macro=defn：定义宏 macro 为 defn。

(7) 汇编选项如下。

-Wa,option：将选项 option 传递给汇编器。

(8) 搜索路径选项如下。

-I dir：设置搜索路径为 dir。

-I-：指定只对 #include "file"有效的头文件搜索目录。

3. 源文件类型的识别

arm-elf-gcc 能够自动根据文件名后缀识别文件类型。

文件名后缀和文件类型的对应关系如下。

*.c：C 源文件。

*.i：经过预处理后的 C 源文件。

*.h：C 头文件。

*.ii：经过预处理后的 C++ 源文件。

*.cc：C++ 源文件。

*.cxx：C++ 源文件。

*.cpp：C++ 源文件。

*.C：C++ 源文件。

*.s：不需要预处理的汇编文件。

*.S：需要预处理的汇编文件。

此外，用户可通过-x language 说明文件的输入类型，此时可以不用以上的后缀规则。例如：

```
-x language
```

其中的 language 可为以下参数。

c：C 源文件。

C++ ：C++ 源文件。

c-header：C 头文件。

cpp-output：经过预处理后的 C 源文件。

C++ -cpp-output：经过预处理后的 C++ 源文件。

assembler：不需要预处理的汇编文件。

assembler-with-cpp：需要预处理的汇编文件。

例如，编译一个不需要预处理的 C 程序：

```
arm-elf-gcc  -c  -g  -x  cpp-output test.c
-x none
```

如果-x 后面未跟任何参数，则按照文件的后缀名做相应处理。

3.2.2 命令使用

1. 输出文件名的指定

-o file。将输出内容存于文件 file，仅适用于只有一个输出文件时。例如，将 test.c 编译成汇编程序并存放于文件 test.txt：

```
arm-elf-gcc  -S  -o test.txt test.c
```

2. 目标文件的生成

-c。将输入的源文件编译成目标文件。例如，将 test.c 编译成 test.o：

```
arm-elf-gcc  -c  -o test.o test.c
```

3. 将 C/C++ 文件生成汇编文件

-S。将 C/C++ 文件生成汇编文件。例如，将 test.c 编译生成汇编文件 test.s：

```
arm-elf-gcc  -S  -o test.s test.c
```

4. 预处理文件的生成

-E。只对源文件进行预处理并且默认输出到标准输出。例如，对 test.c 进行预处理并将结果输出到屏幕：

```
arm-elf-gcc  -E test.c
```

例如，对 test.c 进行预处理并将结果输出到文件 test.txt：

```
arm-elf-gcc  -E  -o test.txt test.c
```

5. 设置头文件搜索路径

头文件的引用有两种形式：一种是 #include"filename"，另一种是 #include<filename>。前一种形式的路径搜索顺序是当前目录、指定的搜索路径；后一种形式只搜索指定路径。

(1) -Idir。将目录 dir 添加到头文件搜索目录列表的第一项。通过此选项可以使用户头文件先于系统头文件被搜索到。如果同时用-I 选项添加几个目录，目录被搜索时的优先级顺序为从左到右。

例如，编译 test.c，在当前目录和/include 中搜索 test.c 所包含的头文件：

```
arm-elf-gcc  -I./  -I/include  -c test.c
```

(2) -I-。-I-以前用-I 指定的头文件搜索目录只对 #include"file" 有效，对 #include<file>无效；-I-以后指定的头文件搜索目录对以上两种形式的头文件都有效。此外，-I-会禁止对当前目录的隐含搜索，不过用户可以通过使用“-I.”使能对当前目录的搜索。

例如，在需要编译的 test.c 文件对头文件的引用如下：

```
#include<file1.h>
#include "file2.h"
#include "file3.h"
```

其中，file1.h 在目录/include/test 下，file2.h 在/include 下，file3.h 在当前目录下。在以下命令行中，只能搜索到 file2.h，而不能搜索到 file1.h：

```
arm-elf-gcc  -I./include/test  -I-  -I./include  -c test.c
```

在以下命令行中，可以搜索到需要的两个头文件 file1.h 和 file2.h：

```
arm-elf-gcc  -I-  -I./include  -I./include/test  -c test.c
```

如果要搜索到 file3.h，必须要添加对当前目录的搜索：

```
armelf-gcc  -I-  -I.  -I./include  -I./include/test  -c test.c
```

实质上，上述编译命令等价于

```
arm-elf-gcc  -I.  -I./include  -I./include/test  -c test.c 与 arm-elf-gcc
-I./include  -I./include/test  -c test.c
```

6. 控制警告产生

用户可以使用以-W 开头的不同选项对特定警告进行设定。对于每种警告类型都有相应以-Wno-开始的选项关闭警告。

例如，如果有隐含申明，显示警告信息：

```
arm-elf-gcc  -c  -Wimplicit test.c
```

不显示对隐含申明的警告：

```
arm-elf-gcc  -c  -Wno-implicit test.c
```

常用的警告选项有以下参数。

-w：关闭所有警告信息。

-Wall：打开所有警告信息。

7. 实现优化

优化的主要目的是使编译生成的代码的尺寸更小、运行速度更快，但是在编译过程中随着优化级别的升高，编译器会相应消耗更多时间和内存，而且优化生成代码的执行顺序和源代码有一定出入，因此优化选项更多地用于生成固化代码，而不用于生成调试代码。arm-elf-gcc 支持多种优化选项，总体上划分为三级优化。

(1) -O1。可以部分减小代码尺寸，对运行速度有一定的提高。较多地使用了寄存器变量，提高指令的并行度。

(2) -O2。除了解循环、函数插装和静态变量优化，几乎包含 arm-elf-gcc 所有优化选项。一般在生成固化代码时使用该选项较为适宜。

(3) -O3。包含-O2 的所有优化，并且还包含了解循环、函数插装和静态变量优化。通常情况下，该级优化生成的代码执行速度最快，但是代码尺寸比-O2 大一些。

8. 在命令行定义宏

(1) -D macro。定义宏 macro 为 1。

(2) -D macro=defn。定义宏 macro 为 defn。

例如，编译 test.c 并且预定义宏 RUN_CACHE 值为 1 的命令行如下：

```
arm-elf-gcc  -c  -D  RUN_CACHE test.c
```

编译 test.c 并且预定义宏 RUN_CACHE 值为 0 的命令行如下：

```
arm-elf-gcc  -c  -D  RUN_CACHE= 0 test.c
```

3.3 交叉连接器 arm-elf-ld

3.3.1 概述

arm-elf-ld 根据链接定位文件 Linkcmds 中代码段、数据段、BSS 段和堆栈段等定位信息，将可重定位的目标模块链接成一个单一的、绝对定位的目标程序，该目标程序是 ELF 格式，并且可以包含调试信息。arm-elf-ld 可以输出一个内存映像文件，该文件显示所有目标模块、段和符号的绝对定位地址，它也产生目标模块对全局符号引用的交叉参考列表。

arm-elf-ld 支持将多个目标模块链接成一个单一的、绝对定位的目标程序，也能够依次对目标模块进行链接，这个特性称为增量链接(Incremental Linking)。

arm-elf-ld 会自动从库中装载被调用函数所在的模块。

1. 命令格式

arm-elf-ld 的命令格式如下：

```
arm-elf-ld [option] file  …
```

命令行后跟选项和可重定位的目标文件名。

例如，若链接的输入文件为 demo.o，输出文件为 demo.elf，链接的库为 libxxx.a，生成内存映像文件 map.txt，链接定位文件为 linkcmds，则命令如下：

```
arm-elf-ld  -Map map.txt  -N  -T linkcmds  -L./lib  -o demo.elf demo.o  -lxxx
```

2. 命令选项列表

arm-elf-ld 的命令选项列表如下。

-e entry：指定程序入口。

-Map：指定输出映像文件。

-M：输出链接信息。

-lar：指定链接库。

-Ldir：添加搜索路径。

-o：设置输出文件名。

-T commandfile：指定链接命令文件。

-v：显示版本信息。

3.3.2 命令使用

1. 程序入口地址

-e entry 是以符号 entry 作为程序执行的入口地址，而不从默认的入口地址开始。默认入口地址的指定方式和其他指定方式的描述，参见 3.3.3 小节。

例如，链接的输入文件为 demo.o，输出文件为 demo.elf，链接定位文件为 linkcmds，将入口地址设为_start，命令如下：

```
arm-elf-ld  -T linkcmds  -e _start  -o demo.elf demo.o
```

2. 输出链接信息

(1) -M。在标准端口打印出符号映像表和内存分布信息。

例如，链接的输入文件为 demo.o，输出文件为 demo.elf，在标准端口打印出符号映像表和内存分布信息，命令如下：

```
arm-elf-ld  -M  -o demo.elf demo.o
```

如果标准输出设置为显示器，运行命令后将在显示器上显示内存映像信息和符号映像表。

(2) -Map mapfile。将链接的符号映像表和内存分布信息输出到文件 mapfile 里。

例如，链接的输入文件为 demo.o，输出文件为 demo.elf，将链接的符号映像表和内存分布信息输出到文件 map.txt 里，命令如下：

```
arm-elf-ld  -Map map.txt  -o demo.elf demo.o
```

3. 指定链接的库

-lar。用于指定库文件 libar.a 为链接的库。可以重复使用-l 来指定多个链接的库。

例如，链接的输入文件为 demo.o，指定 libxxx.a 为链接的库，输出文件为 demo.elf，命令如下：

```
arm-elf-ld  -o demo.elf demo.o  -lxxx
```

注意：库的命名规则为 libxxx.a，在-l 指定库名时使用的格式为-lxxx。

4. 添加库和脚本文件的搜索路径

-Ldir。用于将 dir 添加到搜索路径。搜索顺序按照命令行中输入的顺序，并且优先于默认的搜索路径。所有在-L 添加的目录中找到的-l 指定的库都有效。

例如，链接的输入文件为 demo.o，输出文件为 demo.elf，将/lib 添加到库的搜索路径，命令如下：

```
arm-elf-ld  -L./lib  -o demo.elf demo.o
```

5. 设置输出文件的名字

-o output。用于将输出文件名字设定为 output。如果不指定输出文件名，arm-elf-ld 生成文件名默认为 a.out。例如，链接的输入文件为 demo.o，输出文件为 demo.elf，命令如下：

```
arm-elf-ld  -o demo.elf demo.o
```

3.3.3 linkcmds 链接命令文件

arm-elf-ld 的命令语言是一种描述性的脚本语言，它主要用于控制：有哪些输入文件、文件的格式怎样、输出文件中的模块怎样布局、分段的地址空间怎样分布，以及未初始化的数据段怎样处理等。

用命令语言写成的文件(通常称为 linkcmds)具有可重用性，不必每次在命令行输入一大堆命令选项。并且对于不同的应用，只需对 linkcmds 进行简单的修改就可以使用。

1. 调用 linkcmds

首先写一个链接命令文件 linkcmds，然后在 arm-elf-ld 的命令中使用-T linkcmds 参数，就能在链接时自动调用 linkcmds 文件。

例如，链接的输入文件为 demo.o，输出文件为 demo.elf，链接定位文件为 linkcmds，则命令如下：

```
arm-elf-ld  -T linkcmds  -o demo.elf demo.o
```

2. 编写 linkcmds

(1) arm-elf-ld 命令语言。arm-elf-ld 命令语言是一系列语句的集合，包括用简单的关键字设定选项、描述输入文件及其格式、命名输出文件。其中有两种语句对于链接过程起重要作用：SECTIONS 语句和 MEMORY 语句。SECTIONS 语句用于描述输出文件中的模块怎样布局，MEMORY 语句描述目标机中可以用的存储单元。

(2) 表达式。在 linkcmds 中的表达式与 C 语言中的表达式类似，它们具有如下的特征：

- 表达式的值都是 unsigned long 或者 long 类型；
- 常数都是整数；
- 支持 C 语言中的操作符；
- 可以引用或者定义全局变量；

• 可以使用内建的函数。

① 整数。八进制数以“0”开头，例如：0234；十进制数以非 0 的数字开头，例如：567；十六进制数以“0x”或“0X”开头，例如：0x16；负数以运算符“-”开头，例如：-102；以 K，M 为后缀分别表示以 1024，1024 * 1024 为单位，例如：var1＝1K 和 var1＝1024 相等，var2＝1M 和 var2＝1024 * 1024 相等。

② 变量名。以字母、下划线和点开头，可以包含任何字母、下划线、数字、点和连接符。变量名不能和关键字一样，如果变量名和关键字一样，或者变量名中包含空格时，必须将变量名包含在" "中，例如：

```
"SECTION"=9;
"with a space"="also with a space"+10;
```

在 arm-elf-ld 命令语言中，空格用于界定相邻符号，例如：A　-B 表示一个变量名，而A－B 表示一个减法的表达式。

③ 地址计数器点号(.)。点号是一个包含当前输出地址的计数器。因为点号总是表示某个输出段的地址，所以总是和表达式一起在 SECTIONS 命令中出现。点号可以在任何一般符号出现的地方出现，对点号的赋值将引起计数器所指位置的移动，而计数器位置不能反向移动。例如：

```
SECTIONS
{
    output:
    {
        file1(.text)
        . =.+1000;
        file2(.text)
        .+=1000;
        file3(.text)
    }=0x1234;
}
```

在上面的例子中，在 file1(. text)与 file2(. text)之间被空出了 1000B 的空间，file2(. text)与 file3(. text)之间也被空出了 1000B 的空间，而 0x1234 为该分段的空间空隙的填充值。可以将点号(.)赋给变量，也可以对点号(.)赋值，例如：

```
data_start=.;
.=.+2000;
```

(3) linkcmds 的结构。linkcmds 文件主要由以下 4 个部分组成。

① 程序入口说明。用于指定程序运行时所需要执行的第一条指令的地址。

② 程序头说明。生成目标文件类型为 ELF，可以指定详细的程序头信息。

③ 内存布局的说明。用于规划内存的布局，将内存空间划分为不同的部分。

④ 分段的分步说明。详细指明各个分段的构成以及分段的定位地址和装载地址。

其中①和④的部分不能被省略。

(4) 对程序入口的说明。arm-elf-ld 命令语言有一条特定命令用于指定输出文件中第