GNU gcc 嵌入式系统开发

董文军 编著

北京航空航天大学出版社

GNU gcc 嵌入式系统开发

董文军　编著

北京航空航天大学出版社

内 容 简 介

本书首先介绍了 GNU gcc 的基本组成，分章节讲述了 GNU gcc、Debian Linux、makefile、gdb、vi、emacs 等开源开发工具的使用，然后以 Atmel 公司的两款颇具代表性的嵌入式芯片，即低端的 8 位 AVR 单片机 ATmega48和中端的 32 位 ARM 芯片 AT91SAM7S64 为代表，全面讲述了 GNU gcc 在嵌入式开发中的应用。可以看到 GNU gcc 在不同硬件下的开发过程与使用方法的确具有高度的一致性，给学习与使用带来了很大的方便。书中还特别列举了非常实用的开源项目 USBASP 以及 usbdrv，使读者既能对开源软件的强大功能留下深刻的印象，又能学到实际有用的东西。

本书可作为高等院校计算机、电子、自动化、机电一体化等相关专业嵌入式系统课程的教学参考书，也可作为从事嵌入式系统应用开发工程师的参考资料。

图书在版编目(CIP)数据

GNU gcc 嵌入式系统开发/董文军编著. —北京：北京航空航天大学出版社，2010.1

ISBN 978-7-81124-814-2

Ⅰ.G… Ⅱ.董… Ⅲ.操作系统(软件)，GNU gcc Ⅳ.TP316.7

中国版本图书馆 CIP 数据核字(2009)第 111045 号

GNU gcc 嵌入式系统开发

董文军 编著

责任编辑 董立娟

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(100191) 发行部电话：010-82317024 传真：010-82328026

www.buaapress.com.cn E-mail：emsbook@gmail.com

涿州市新华印刷有限公司印装 各地书店经销

*

开本：787×960 1/16 印张：26.25 字数：588 千字

2010 年 1 月第 1 版 2010 年 1 月第 1 次印刷 印数：4 000 册

ISBN 978-7-81124-814-2 定价：45.00 元

前　言

笔者从事电子类相关工作近二十年，一直都对此有着深厚的兴趣。自学生时代的家电维修开始，到后来从事计算机硬件教学工作这一路走来，从电子硬件电路、单片机应用，到计算机应用程序、驱动程序开发以及到现在的软硬一体化的嵌入式系统，一步一步地从最底层的电路焊接开始到现在，计算机硬件的多个层次都曾接触过，较长的专业生涯使笔者也积累了较多的经验体会。一直以来也有想法，将这些经验集结成册，为推广计算机知识尽一点绵薄之力，恰逢北京航空航天大学出版社的邀请，于是决定着手此书的编写。

做过技术工作的人都有过这样的经历与体会，大量的时间都是花费在技术资料的阅读与查找上，很多项目常常因为某个技术资料的缺乏而使项目卡壳，有时甚至影响到项目的顺利完成。这一点在技术飞速发展的 IT 行业特别在计算机软件领域表现更为突出，很多的技术细节被隐藏在不公开的源代码中，源码的不公开往往造成技术发展的瓶颈，直到 20 世纪 80 年代出现了一种全新的开源思想：它认为计算机源代码应该像文学艺术作品一样作为人类共同知识财富的一部分，让每个人都有机会阅读与学习，而不应该只作为公司或个人的私有财产，并为此发布了一系列的法律文件来保证开源软件源码的公开性。开源精神得到了广大计算机专业人士及爱好者的大力拥护，包括很多世界著名的大公司在内，行业内的很多精英都投身到这一伟大的事业中来，这极大地促进了计算机产业的发展。在开源精神的指导下，世界有了全新的操作系统 Linux，有了全新的开发工具 gcc，有了全新的文本编辑器 emacs 等。

开源软件对于莘莘学子来说更是一个福音，有了开源代码，他们就能自由地阅读到世界一流的代码，学习的资料极大地丰富，所学的技术与知识也能最快地站在世界的前沿；能迅速有效地将所学的理论用上，理论与实践的距离被迅速拉近，学习变成了探索，变成了一件有趣的活动。开源软件同时也极大地促进了社会经济的发展，IT 产品渐渐地不再昂贵，很多公司的产品都是来源于开源项目，人类智慧的共同合力得到了充分的体现。开源精神在大家的共同努力下不断发展，对世界产生了越来越大的影响。

现在IT产业已进入了后PC时代，传统的桌面PC市场已经趋于饱和，现阶段难以找到新的增长点，但随着人们更热衷于快捷方便、能随身携带的IT服务，低功耗的便携式产品成为了市场的新热点，也成为广大厂商与专业人员追捧的目标，连IT业巨头Intel都高调宣布进军嵌入式领域。开源软件也顺应时代潮流率先进入了嵌入式时代，为广大开发者提供了特性高度一致的嵌入式产品，Linux、gcc等开源软件都能在嵌入式系统中使用，而且目前还是支持硬件最多的开发工具与平台，这正适合于嵌入式硬件平台众多的特点。

为了能全面描述GNU gcc在嵌入式开发中的应用，笔者选择了Atmel出品的两款颇具代表性的嵌入式芯片，一款为低端的8位AVR单片机ATmega48，另一款为中端的32位ARM芯片AT91SAM7S64。本书对这两款芯片都讲述了使用GNU gcc开发工具进行开发的方法与过程，可以看到GNU gcc在不同硬件下的开发过程与使用方法的确是具有高度的一致性，这给学习与使用带来了很大的方便。书中还特别列举了非常实用的开源项目USBASP以及usbdrv，使读者既能对开源软件的强大功能留下深刻的印象，又能学习实际有用的东西。

写书是一项艰苦的工作，为此笔者停下了一些项目的开发工作，专心写书。在此过程中得到了家人以及很多朋友、同事的关心与帮助，在此表示衷心的感谢。首先要感谢笔者的妻子和家人，他们的支持让我能安下心来写书；特别感谢我的合作伙伴刘宾林，他给了我很多专业上的帮助；还有这些人员：董振兴、邹远菊、刘中平、谭蔚芸、陈双妹、刘冬丽、伍向阳、刘新宇等，他们都为本书作出了贡献，这里一并表示感谢。同时衷心感谢北航出版社，使本书得以顺利出版。

有兴趣的读者可以发送电子邮件到：dongwj@gzpyp.edu.cn，与作者进一步交流；也可以发送电子邮件到xdhydcd5@sina.com，与本书策划编辑进行交流。

董文军

2009年11月

目 录

第1章

GNU gcc 概述

1.1 自由软件与 GNU、GPL

计算机软件作为人类的知识财富，为人类社会的发展起到了巨大的作用，但长期以来软件源码作为个人或公司的私有财产受到严格的保密，很难做到像文学艺术作品一样地进行公开的交流，很大程度上造成软件的低水平，重复劳动严重，在一定意义上制约了软件的发展。直到 1985 年由 MIT 教授理查德·斯托曼(Richard Stallman)提出应将软件源码看成人类共同拥有的知识财富，应该公开地自由交换、修改，提出了 GNU 计划(因英文名相同，GNU 的 logo 就是一只牛羚)，并建立了自由软件基金会；同时，发布了一份举足轻重的法律文件，GNU 通用公共授权书(GNU GPL，GNU General Public License)。

该授权书主要有以下几点：

① 自由软件(free software)指的是源码自由，不是价格；

② 自由软件必须附带程序源代码，但可收取费用；

③ 任何人都可以自由分发自由软件并收取费用，但必须列明原创者姓名；

④ 任何人都可以修改源代码，但必须列明修改人名字，以保护原创者名誉；

⑤ 任何人都可以采用源代码中的某一段，但其开发之软件必须也为自由软件(例如，如果 Netscap 是自由软件，而 IE 采用了其中的部份源代码，则 IE 也必须成为自由软件)；

⑥ 任何自由软件的衍生品也必须是自由软件；

⑦ 自由软件没有担保，以保护分发者。

1991 年，Richard Stallman 对授权做了微小的修改，即所谓的通用公共授权第 2 版。同时，他也推出了更宽松的通用公共授权，用于自由程序库。这一系列的授权有效地保护了自由软件不受商业软件的非法侵犯，例如，1998 年 Netscap 决定采用与 GPL 差不多的 NPL (Netscap Public Liscense)，这样一来，Microsoft 就无法将 Netscap 中的源代码运用在 IE 上，除非它们也要成为自由软件。

至此，在 GPL 下人们就可以自由交流、修改软件源码了，这一协议极大地推动了整个计算机软件行业的发展，并带来了以下明显的益处：

① 对于广大计算机软件的学习者来说，可以直接从源码中吸取营养，缩短学习的时间，提高学习的效率，少走弯路，再也不必花大量时间去看那些不知正确与否的“未解之谜”了，学习在某种程度上变成了一件轻松愉快的事情了。

② 可以集中大家的智慧发展软件，避免重复劳动。一个软件只有公开源码，通过很多人的研究才有可能发现其中深藏的错误，大家才能公开探讨相关的问题，并进行改进，在大家的共同“挑剔与监督”下才有可能编写出尽善尽美的软件来。

GPL 协议的核心就是要对源码进行公开，并且允许任何人修改源码，但是只要使用了 GPL 协议的软件源码，其衍生软件也必须公开源码，准许其他人阅读和修改源码，即 GPL 协议具有继承性。另一个问题就是 GPL 软件并非就是免费软件，这里所说的自由软件是指对软件源码的自由获得与自由使用、修改，软件开发者不但可以通过服务来收费，而且还可以通过出售 GPL 软件来获利。

适应 GPL 协议的软件一般都是自由软件，自由软件是指一件可以让用户自由复制、使用、研究、修改、分发等，而不附带任何条件的软件。

Stallman 为了停止中间人对自由软件权利的侵害，提出了 copyleft 授权，因为自由软件在发布过程中可能会有一些不合作的人通过对程序的修改而将软件变成私有软件，将程序变成 copyleft 授权。我们首先声明它是有版权的，而后加入了分发条款，这些条款是法律指导，使得任何人都拥有对这一程序代码或者任何这一程序的衍生品的使用、修改和重新发布的权力，但前提是这些发布条款不能被改变。这样在法律上，代码和自由就不可分割了。

自由软件的支持者相信，总有一天，随着自由软件的日渐成熟，自由软件终将主宰整个软件行业，人们不再受少数商业软件公司的控制，真正实现“市集式开发模式”。

1.2 gcc 的发展历史及特点

GNU 项目计划的主要目的是创建一个名叫 GNU's Not Unix(GNU)的完全免费的操作系统。该操作系统将包括绝大多数自由软件基金会所开发的其他软件，以对抗所有商业软件，而这个操作系统的核心(kernel)就叫 HURD。但是 GNU 在开发完全免费的操作系统上并未取得成功，直到 20 世纪 90 年代由林纳斯·本纳第克特·托瓦兹(Linus Benedict Torvalds)开发了 Linux 操作系统，GNU 才算在免费操作系统上完成了任务。

虽然 GNU 计划在开发免费操作系统上不成功，但是却成功开发几个广为流传的 GNU 软件，其中最著名的是 GNU C Complier(gcc)。这个软件成为历史上最优秀的 C 语言编译器，其执行效率与一般的编译器相比平均效率要高 20%～30%，使得那些靠贩卖编译器的公司大吃苦头，因为它们无法研制出与 gcc 同样优秀，却又完全免费、并开放源代码的编译器来。而由于它又是 copylefted，所以一旦有用户发现错误，就会通知 Richard Stallman，所以几乎每个月都可以推出新版本。然而，它还有一个十分特殊而且不同寻常的意义：几乎所有的自由软件

都是通过它编译的。可以说，它是自由软件发展的基石与标杆。现在，gcc 已经可以支持 7 种编程语言和 30 种编程结构，是学术界最受欢迎的编译工具。其他 GNU 软件还包括 GNU emacs、GNU Debugger(GDB)、GNU Bash 以及大部分 Linux 系统的程序库和工具等。

目前，gcc 已发展到了 4.0 的版本，几乎所有开源软件和自由软件中都会用到，因此它的编译性能会直接影响到 Linux、Firefox、OpenOffice.org、Apache 以及一些数不清的小项目的开发。gcc 无疑处在开源软件的核心地位。

作为自由软件的旗舰项目，Richard Stallman 在十多年前刚开始写作 gcc 的时候，还只是把它当作一个 C 程序语言的编译器；gcc 的意思也只是 GNU C Compiler 而已。经过这么多年的发展，gcc 已经不仅仅能支持 C 语言，它现在还支持 Ada、C++、Java、Objective C、Pascal、COBOL 以及函数式编程和逻辑编程的 Mercury 语言等。因此，现在的 gcc 已经变成了 GNU Compiler Collection，也即是 GNU 编译器家族的意思了。这个名称同时也说明了 gcc 对于各种硬件平台无所不在的支持，甚至包括一些生僻的硬件平台。

gcc 不仅功能非常强大，结构也异常灵活。最值得称道的一点就是，它可以通过不同的前端模块来支持各种语言，如 Java、Fortran、Pascal、Modula-3 和 Ada 语言等。

在使用 gcc 编译程序时，编译过程可以被细分为 4 个阶段：

- 预处理(Pre-Processing)；
- 编译(Compiling)；
- 汇编(Assembling)；
- 链接(Linking)。

程序员可以根据自己的需要让 gcc 在编译的任何阶段结束，以检查或使用编译器在该阶段的输出信息，或者对最后生成的二进制文件进行控制，以便通过加入不同数量和种类的调试代码来为今后的调试做好准备。和其他常用的编译器一样，gcc 也提供了灵活而强大的代码优化功能，利用它可以生成执行效率更高的代码。

gcc 提供了 30 多条警告信息和 3 个警告级别，有助于增强程序的稳定性和可移植性。此外，gcc 还对标准的 C 和 C++语言进行了大量扩展，提高了程序的执行效率，有助于编译器进行代码优化，能够减轻编程的工作量。

如果没有给出可执行文件的名字，gcc 将生成一个名为 a.out 的文件。在 Linux 系统中，可执行文件没有统一的后缀，系统从文件的属性来区分可执行文件和不可执行文件，而 gcc 则通过后缀来区别输入文件的类别。下面我们来介绍 gcc 所遵循的部分约定规则。

.c 为后缀的文件，是 C 语言源代码文件；

.a 为后缀的文件，是由目标文件构成的档案库文件；

.C、.cc、.cpp、.C++、.cp 或.cxx 为后缀的文件，是 C++源代码文件；gcc 支持 C++语言，可以根据这些 C++文件的后缀自动使用 C++语法进行编译；

.h 为后缀的文件，是程序所包含的头文件；

.i 为后缀的文件，是已经预处理过的 C 源代码文件；

.ii 为后缀的文件，是已经预处理过的 C++源代码文件；

.m 为后缀的文件，是 Objective-C 源代码文件；

.o 为后缀的文件，是编译后的目标文件；

.s 为后缀的文件，是汇编语言源代码文件；

.S 为后缀的文件，是经过预编译的汇编语言源代码文件。

gcc 首先调用 cpp 进行预处理，在预处理过程中，对源代码文件中的文件包含(include)、预编译语句(如宏定义 define 等)进行分析。接着调用 ccl 进行编译，这个阶段根据输入文件生成以.o 为后缀的目标文件。汇编过程就是针对汇编语言的步骤，调用 as 进行工作，一般来讲，.S 为后缀的汇编语言源代码文件和.s 为后缀的汇编语言文件经过预编译和汇编之后都生成以.o 为后缀的目标文件。当所有的目标文件都生成之后，gcc 就调用 ld 来完成最后的关键性工作，这个阶段就是链接。在链接阶段，所有的目标文件被安排在可执行程序中的恰当位置，同时，该程序所调用到的库函数也从各自所在的库中连到合适的地方。

gcc 是整个协作软件开发理念的基础，时至今日，gcc 的使用范围已不仅仅限于 Linux 平台，而是扩展到了包括 Windows 在内的很多平台。这样，gcc 性能的高低，还关系到许多专有软件的核心竞争力。

1.3 gcc 的使用简介与命令行参数说明

1.3.1 gcc 的基本用法

使用 gcc 编译器时，必须给出一系列必要的调用参数和文件名称。不同参数的先后顺序对执行结果没有影响，只有在使用同类参数时的先后顺序才需要考虑。如果使用了多个-L 的参数来定义库目录，gcc 会根据多个-L 参数的先后顺序来执行相应的库目录。因为很多 gcc 参数都由多个字母组成，所以 gcc 参数不支持单字母的组合，Linux 中常被叫短参数(short options)，如-dr 与-d -r 的含义是不一样。gcc 编译器的调用参数大约有 100 多个，其中多数参数我们可能根本就用不到，这里只介绍其中最基本、最常用的参数。

gcc 最基本的用法是：

gcc [options] [filenames]

其中，options 就是编译器所需要的参数，filenames 给出相关的文件名称，最常用的有以下参数。

-c，只编译，不链接成为可执行文件。编译器只是由输入的.c 等源代码文件生成.o 为后缀的目标文件，通常用于编译不包含主程序的子程序文件。

-o output_filename，确定输出文件的名称为 output_filename。同时这个名称不能和源文件同名。如果不给出这个选项，gcc 就给出默认的可执行文件 a. out。

-g，产生符号调试工具(GNU 的 gdb)所必要的符号信息。要想对源代码进行调试，就必须加入这个选项。

-O，对程序进行优化编译、链接。采用这个选项，整个源代码会在编译、链接过程中进行优化处理，这样产生的可执行文件的执行效率可以提高，但是编译、链接的速度就相应地要慢一些，而且对执行文件的调试会产生一定的影响，造成一些执行效果与对应源文件代码不一致等一些令人“困惑”的情况。因此，一般在编译输出软件发行版时使用此选项。

-O2，比-O 更好的优化编译、链接。当然整个编译、链接过程会更慢。

-Idirname，将 dirname 所指出的目录加入到程序头文件目录列表中，是在预编译过程中使用的参数。C 程序中的头文件包含两种情况：

#include <stdio. h>

#include“stdio. h”

其中，前者使用尖括号(< >)，后者使用双引号(“”)。对于前者，预处理程序 cpp 在系统默认包含文件目录(如/usr/include)中搜寻相应的文件；而对于后者，cpp 在当前目录中搜寻头文件，这个选项的作用是告诉 cpp，如果在当前目录中没有找到需要的文件，就到指定的 dirname 目录中去寻找。在程序设计中，如果需要的这种包含文件分别分布在不同的目录中，就需要逐个使用-I 选项给出搜索路径。

-Ldirname，将 dirname 所指出的目录加入到程序函数库文件的目录列表中，是在链接过程中使用的参数。在默认状态下，链接程序 ld 在系统的默认路径中(如/usr/lib)寻找所需要的库文件。这个选项告诉链接程序，首先到-L 指定的目录中去寻找，然后到系统默认路径中寻找；如果函数库存放在多个目录下，就需要依次使用这个选项，给出相应的存放目录。

-lname，链接时装载名为 libname. a 的函数库。该函数库位于系统默认的目录或者由-L 选项确定的目录下。例如，-lm 表示链接名为 libm. a 的数学函数库。

假定有一个程序名为 test. c 的 C 语言源代码文件，要生成一个可执行文件。

清单 1：test. c

```
#include < stdio.h>
int main(void)
{
printf ("Hello world, Linux programming! \n");
return 0;
}
```

最简单的办法就是：

```
gcc test. c -o test
```

首先，gcc 需要调用预处理程序 cpp，由它负责展开在源文件中定义的宏，并向其中插入

"#include"语句所包含的内容；接着，gcc 调用 ccl 和 as，将处理后的源代码编译成目标代码；最后，gcc 调用链接程序 ld，把生成的目标代码链接成一个可执行程序。因此，默认情况下，预编译、编译链接一次完成。

编译过程的分步执行：

为了更好地理解 gcc 的工作过程，我们可以让在 gcc 工作的 4 个阶段中的任何一个阶段中停止下来。相关的参数有：

-E 预编译后停下来，生成后缀为.i 的预编译文件；

-S 汇编后停下来，生成后缀为.s 的汇编源文件；

-c 编译后停下来，生成后缀为.o 的目标文件。

可以把上述编译过程分成几个步骤单独进行，并观察每步的运行结果。第 1 步是进行预编译，使用-E 参数可以让 gcc 在预处理结束后停止编译过程：

```
# gcc -E test.c -o test.i
```


gcc 执行到预编译后停下来，生成 test.i 的预编译文件。此时若查看 test.i 文件中的内容，会发现 stdio.h 的内容确实都插到文件里去了，而其他应当被预处理的宏定义也都做了相应的处理。下一步是将 test.i 编译为目标代码，这可以通过使用-c 参数来完成：

```
# gcc -c test.c -o test.o
```

gcc 执行到编译后停下来，生成 test.o 的目标文件：

```
# gcc -S test.c -o test.s
```

gcc 执行到汇编后停止，生成 test.s 的汇编源文件，这是一个非常重要的中间文件。因为在嵌入式系统中，受系统硬件资源的限制往往对目标文件的执行效率、文件大小都有要求，通过人工查看汇编源文件有时往往可以找到问题所在进行人工优化。

最后一步是将生成的目标文件链接成可执行文件：

```
# gcc test.o -o test
```

对于稍为复杂的情况，比如有多个源代码文件、需要链接库或者有其他比较特别的要求，就要给定适当的调用选项参数。再看一个简单的例子。

整个源代码程序由两个文件 testmain.c 和 testsub.c 组成，程序中使用了系统提供的数学库(所有与浮点相关的数学运算都必须使用数学库，这一点初学者往往忽略，以为与 Windows 下 C 语言一致，结果往往在作浮点运算时，如三角函数运算，编译执行没有任何错误，但却得不到正确的结果)，同时希望给出的可执行文件为 test，这时的编译命令可以是：

```
gcc testmain.c testsub.c -lm -o test
```

其中，-lm 表示连接系统的数学库 libm.a。

在编译一个包含许多源文件的工程时，若只用一条 gcc 命令来完成编译是非常浪费时间的。假设项目中有 100 个源文件需要编译，并且每个源文件中都包含 10 000 行代码，如果像上面那样仅用一条 gcc 命令来完成编译工作，那么 gcc 需要将每个源文件都重新编译一遍，然

后再全部链接起来。很显然，这样浪费的时间相当多，尤其是当用户只是修改了其中某个文件的时候，完全没有必要将每个文件都重新编译一遍，因为很多已经生成的目标文件是不会改变的。要解决这个问题，需要借助像 make 这样的工具(将在第 3 章详细讲述)。

1.3.2　警告提示功能选项

gcc 包含完整的出错检查和警告提示功能，它们可以帮助 Linux 程序员写出更加专业的代码。

1) -pedantic 选项

当 gcc 在编译不符合 ANSI/ISO C 语言标准的源代码时，产生相应的警告信息。

例如：

illcode.c 清单：

```
#include < stdio.h>
void main(void)
{
long long int var = 1;
printf("It is not standard C code!n");
}
```

它有以下问题：

- main 函数的返回值被声明为 void，但实际上应该是 int；
- 使用了 GNU 语法扩展，即使用 long long 来声明 64 位整数，不符合 ANSI/ISO C 语言标准；
- main 函数在终止前没有调用 return 语句。

使用命令行：gcc -pedantic illcode.c -o illcode，则编译将产生以下错误信息：

```
illcode.c:In function'main':
illcode.c:4:warning:ISO C89 does not support'long long'
illcode.c:3:warning:return type of'main' is not'int'
```

2) -Wall 选项

除了-pedantic 之外，gcc 还有一些其他编译选项，也能够产生有用的警告信息。这些选项大多以-W 开头。其中最有价值的当数-Wall 了，使用它能够使 gcc 产生尽可能多的警告信息：

```
#  gcc -Wall illcode.c -o illcode
illcode.c:3:warning:return type of'main' is not'int'
illcode.c:In function'main':
illcode.c:4:warning:unused variable'var'
```

gcc 给出的警告信息虽然从严格意义上说不能算作错误，但却很可能成为错误来源。一个优秀的程序员应该尽量避免产生警告信息，使自己的代码始终保持简洁、优美和健壮的特性。

gcc 给出的警告信息是很有价值的，它们不仅可以帮助程序员写出更加健壮的程序，而且

还是跟踪和调试程序的有力工具。建议在用 gcc 编译源代码时始终带上-Wall 选项，并把它逐渐培养成为一种习惯，这对找出常见的隐式编程错误很有帮助。

3) -Werror 选项

在处理警告方面，另一个常用的编译选项是-Werror。它要求 gcc 将所有的警告当成错误进行处理，这在使用自动编译工具(如 Make 等)时非常有用。如果编译时带上-Werror 选项，那么 gcc 会在所有产生警告的地方停止编译，迫使程序员对自己的代码进行修改。只有当相应的警告信息消除时，才可能将编译过程继续朝前推进。

4) -Wcast-align 选项

当源程序中地址不需要对齐的指针指向一个地址需要对齐的变量地址时，则产生一个警告，例如，"char *"指向一个"int *"地址，而通常在机器中 int 变量类型是需要地址能被 2 或 4 整除的对齐地址。

此外，常用的选项还有：

-v	输出 gcc 工作的详细过程；
--target-help	显示目前所用的 gcc 支持 CPU 类型；
-Q	显示编译过程的统计数据和每一个函数名。

1.3.3 库操作选项

在 Linux 下开发软件时，完全不使用第三方函数库的情况是比较少见的，通常来讲都需要借助一个或多个函数库的支持才能够完成相应的功能。从程序员的角度看，函数库实际上就是一些头文件(.h)和库文件(.so 或者.a)的集合。虽然 Linux 下的大多数函数都默认将头文件放到/usr/include/目录下，而库文件则放到/usr/lib/目录下，但并不是所有的情况都是这样。正因如此，gcc 在编译时必须有自己的办法来查找所需要的头文件和库文件。常用的有：

(1) -I 选项

可以向 gcc 的头文件搜索路径中添加新的目录。例如，如果在/home/dong/include/目录下有编译时所需要的头文件，为了让 gcc 能够顺利地找到它们，就可以使用-I 选项：

```
# gcc foo.c -I /home/xiaowp/include -o foo
```

(2) -L 选项

如果使用了不在标准位置的库文件，那么可以通过-L 选项向 gcc 的库文件搜索路径中添加新的目录。例如，如果在/home/dong/lib/目录下有链接时所需要的库文件 libfoo.so，为了让 gcc 能够顺利地找到它，可以使用下面的命令：

```
# gcc foo.c -L /home/xiaowp/lib -lfoo -o foo
```

(3) -l 选项

Linux 下的库文件在命名时有一个约定，那就是应该以 lib 这 3 个字母开头，由于所有的