普通高等教育"十二五"计算机类规划教材

# 基于Linux
# 系统的汇编语言程序设计

主　编●苟　娟　关国利
副主编●张　青　张战军

JIYU LINUX XITONG DE HUIBIAN YUYAN CHENGXU SHEJI

郑州大学出版社

# 基于 Linux 系统的汇编语言程序设计

**主　编**　程　楠　关国利
**副主编**　张　青　张战军

郑州大学出版社

**图书在版编目(CIP)数据**

基于 Linux 系统的汇编语言程序设计/程楠,关国利主编. —郑州:郑州大学出版社,2011.8
(普通高等教育"十二五"计算机类规划教材)
ISBN 978-7-5645-0456-4

Ⅰ.①基… Ⅱ.①程…②关… Ⅲ.①Linux 操作系统-程序设计-高等学校-教材 Ⅳ.①TP316.89

中国版本图书馆 CIP 数据核字(2011)第 090442 号

郑州大学出版社出版发行
郑州市大学路 40 号　　邮政编码:450052
出版人:王　锋　　发行部电话:0371-66966070
全国新华书店经销
郑州中兴印务有限公司印制
开本:787 mm×1 092 mm　1/16
印张:21.5
字数:512 千字
版次:2011 年 8 月第 1 版　　印次:2011 年 8 月第 1 次印刷

书号:ISBN 978-7-5645-0456-4　　定价:38.00 元

# 目 录 CONTENTS

## 第一部分 汇编语言程序设计基础



## 第二部分 IA－32 处理器的基本指令系统

## 第三部分 高级汇编语言技术

# 第一部分
# 汇编语言程序设计基础

本部分将为读者揭开汇编语言的神秘面纱，介绍汇编语言程序设计环境基础。汇编语言是面向机器的语言，它在各种处理器和汇编器之间是不同的，所以学习汇编语言程序设计时，首先需要清楚所学习的汇编语言是基于什么软硬件环境。本书使用程序实例运行在 Intel 32 位微处理器系列上的 Linux 操作系统。

本部分内容由四章组成。

第一章“认识汇编语言”：通过介绍汇编语言的特点及应用，使读者清楚为什么要学习汇编语言，掌握汇编语言与高级语言的区别。以一个完整的汇编语言程序为范例，使读者看到汇编语言程序的组成结构，有助于读者在汇编语言的学习中尽早地上机实践。

第二章“IA－32 硬件平台”：对本书汇编语言所基于的硬件环境进行介绍。学习汇编语言时，了解底层的处理器和它如何处理程序是非常重要的。本章没有对 IA－32 平台的操作进行深入的分析，而是针对汇编语言程序设计的需要，介绍 IA－32 微处理器的功能结构、存储器的组织等方面的内容，为后面的程序设计打下坚实的硬件基础。

第三章“数据表示和寻址方式”：对汇编语言编程时涉及的数据的表示和寻址方式的内容进行详细的分析和讲解，目的是为后面进入汇编指令的学习做好扎实的铺垫。

第四章“Linux 系统汇编程序开发工具”：对本书汇编语言开发的相关工具进行介绍，使读者学会在 Linux 系统下搭建汇编程序开发所需的软件平台，学会对汇编程序进行编译、汇编、连接和调试的方法，为顺利地进行汇编语言程序的上机实践做好必要的准备。

# 第一章

# 认识汇编语言

什么是汇编语言？它与其他程序设计语言有什么不同？为什么要学习汇编语言？这些问题都是我们在学习汇编语言时首先应该明白的问题。本章从汇编语言的产生入手，介绍汇编语言的特点、组成以及学习意义，并给出一个完整的汇编语言程序范例。希望读者通过本章的学习能够了解汇编语言的特殊作用，掌握汇编语言与高级语言的区别，从而对汇编语言有一个基础性的认识。

## 1.1　什么是汇编语言

### 1.1.1　机器语言

从电子计算机诞生至今，已经有各种各样的计算机语言面世。然而只有一种计算机语言程序能够在计算机上直接执行，它就是机器语言。除机器语言以外的任何一种计算机语言程序，只有经过翻译转变成机器语言后才能够在计算机上执行。机器语言具有其他计算机语言无法替代的特点。如果想了解计算机底层的工作原理，想真正驾驭计算机，想对计算机设计进行改进或有所创新，都必须了解和掌握机器语言。

计算机最早使用的是机器语言。机器语言程序由机器指令构成，机器指令都由二进制 0 或 1 编码组成。机器指令程序运行前需要存储到内存，正确的机器指令一旦送入 CPU 被译码后就变成动作予以执行。所以机器语言是计算机唯一可以直接识别并执行的语言，因此，它的执行速度是最快的。

一台计算机的所有机器指令的集合称为这台计算机的指令系统。机器指令面向机器，因机器而异。不同的机器具有不同的机器指令，因而具有不同的机器语言。所谓不同的机器，是指具有不同 CPU 的计算机。由于机器指令与 CPU 密切相关，所以不同种类的 CPU 所对应的机器指令也就不同，而且，它们的指令系统往往相差很大。但对于同一系列的 CPU 来说，为了满足各型号之间良好的兼容性，新一代 CPU 的指令系统必须兼容先前开发的同系列 CPU 的指令系统，这样才能够保证先前开发出来的各类程序在新一代 CPU 上能正常运行。

一条机器指令通常由操作码和地址码两部分组成。操作码指出该指令所要完成的操作,即指令功能;地址码指出参与运算的操作数或操作数的地址。每一条机器指令都必须要有唯一的操作码,而地址码则根据指令的具体功能不同,可有可无,可多可少。

例如:89 E5 就是一条 IA - 32 处理器的机器指令(用十六进制表示),这条指令的功能是将寄存器 esp 的内容送到寄存器 ebp 中。该机器指令的二进制代码是 1000100111100101。这只是一条简单的机器指令,但也暴露了机器码的晦涩难懂和不易查错。实际上一个有用的程序至少要有几十行机器码,那么,情况将会怎么样呢? 我们是完全可以想象的。

早期的程序设计均使用机器语言。程序员们将用数字 0 和 1 编成的程序代码打在纸带或卡片上,再用纸带机或卡片机输入计算机,进行计算。程序员们很快发现了使用机器语言带来的麻烦,因为它是如此难于辨别和记忆,必将阻碍整个产业的发展。于是,唯一能够替代机器语言的汇编语言产生了。

### 1.1.2 汇编语言

虽然机器语言编写程序有许多不便,但程序执行效率高,所以,在保证程序执行效率的前提下,人们选用能反映机器指令功能的单词或词组来代替机器指令的操作码,选用相应的符号表示 CPU 内部资源和内存的操作数,这就是汇编指令。

例如:汇编指令　　movl　%esp,%ebp
二进制机器指令为　　1000100111100101
用十六进制表示为　　89 E5
完成的功能为　　将寄存器 esp 的内容传送到寄存器 ebp

汇编指令中 mov 源于单词 move,表示传送指令;ebp、esp 分别表示 IA - 32 微处理器内的 ebp 和 esp 两个寄存器(寄存器是 CPU 内部可以存储数据的器件,在第二章有详细讲解)。寄存器名前面的% 是语法的要求。这样,令人难懂的二进制或十六进制表示的机器代码指令就可以用通俗易懂的、具有一定含义的符号指令来表示了。这些具有一定含义的符号,例如上述例子中的 mov,被称为助记符。使用助记符、符号地址组成的符号指令称为汇编指令。

汇编语言是汇编指令集、伪指令集及其使用规则的统称。如前所述的例子中,能够一对一地翻译成机器指令的这种助记符表示的机器指令称作汇编指令。伪指令则是出现在汇编语言程序中的一些辅助性的说明,它不对应任何具体的机器指令。

用汇编语言编写的程序称作汇编语言程序,或汇编语言源程序,在本书中简称为源程序。显而易见,汇编语言程序较之机器语言程序容易理解和维护。

汇编语言源程序必须经过翻译才能变成可执行的机器语言程序,这个翻译过程称作汇编。汇编的核心工作是将汇编指令逐条翻译成机器指令,这正是汇编语言中“汇编”一词的含义。图 1 - 1 给出了一段简单汇编语言程序与机器语言程序之间的对应关系。

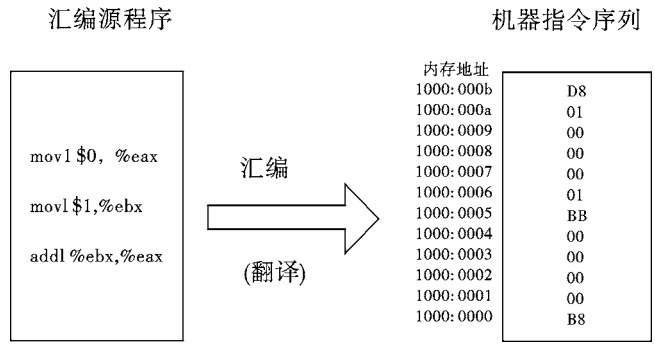


**图 1－1　汇编程序翻译过程示意图**

汇编语言是机器指令的助记符,它同机器语言一样,不同的 CPU 有不同的汇编语言,但不同种类的汇编语言都有其共同规律。因此,学会一种汇编语言,对于其他汇编语言就会无师自通。

汇编语言与机器语言相比,编写容易、修改方便、阅读简单、程序清楚。它为机器语言向算法语言(即高级语言)的靠拢迈出了一大步。但汇编语言仍被列入“低级语言”的范畴,因为它属于面向机器的语言。

### 1.1.3　高级语言与汇编语言

汇编语言虽然较机器语言直观一些,但仍然繁琐难记,于是在 20 世纪 50 年代,人们研制出了高级程序设计语言(high－level language,HLL)。高级语言比较接近于人类自然语言的语法习惯及数学表达形式,应用起来更灵活。利用高级语言编程,不需要知道 CPU 能支持哪些指令,不需要懂得计算机的结构和工作原理,不需要考虑如何执行。当然,用高级语言编写的源程序不会被机器直接执行,而需要编译或解释程序的翻译才可变为机器语言程序。

高级语言和汇编语言相比,程序的可读性、可移植性好,编写和调试程序相对容易,编程效率高,但其产生的目标程序的效率却不高,也就是说,占用空间大,运行时间长。

**例题** 1.1　C 语言与汇编语言对比

下面是用 C 语言编写的一个输出"hello world!"的小程序(程序名:hello.c):

```
#include <stdio.h>
int main()
{
printf("hello world! \n");
return 0;
}
```

C 语言在编译过程中可生成汇编语言的中间程序,执行以下命令编译可生成汇编语言的中间程序(hello.s):

gcc -S hello.c(该命令详见第四章)

编译后生成的汇编语言程序清单如下:

```
    .file   "hello.c"
    .section    .rodata
    .LC0:
    .string   "hello world!"
    .text
    .globl   main
    .type    main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    subl    $8,%ebp
    andl    $-16,%esp
    movl    $0,%eax
    subl    %eax,%esp
    subl    $12,%esp
    pushl   $.LC0
    call    printf
    addl    $16,%esp
    movl    $0,%eax
    leave
    ret
.Lfe1:
    .size main,.Lfe1-main
    .ident "GCC:(GNU) 3.2.2 20030222 (Red Hat Linux 3.2.2-5)
```

上述清单中没有包括 printf()函数所需的代码。

如果用汇编语言直接写同样功能的程序,其汇编程序如下:

```
    .section    .data
    msg:.ascii   "Hello, world!\n"
    len = .-msg
    .section.text
    .globl   _start
    _start:
        movl    $4,%eax
        movl    $1,%ebx
        movl    $msg,%ecx
        movl    $len,%edx
```

```
int    $0x80
movl   $1,%eax
movl   $0,%ebx
int    $0x80
```

以上所有的代码都会在后面的章节中详细讲解,在这里不必深究。通过对比可看出:如此短的 C 程序所产生的汇编程序却很长。

不难看出,与汇编语言相比,高级语言解决同样的问题,最终的机器代码长,占用存储空间大,执行效率低。但用高级语言开发程序相对容易,特别是开发大型程序。另外,高级语言很难直接对硬件加以控制,因此,在对程序的空间和时间要求很高的场合和在要求直接对硬件控制的场合,一般需要使用汇编语言编程。因此,高级语言和汇编语言各有优缺点。

### 1.1.4　汇编语言的主要特性

汇编语言的特性归纳起来有以下几点。

(1)与机器的相关性　汇编语言与 CPU 密切相关。每种 CPU 都有自己的指令系统,相应的汇编语言各不相同。除了同系列、不同型号 CPU 之间的汇编语言程序有一定程度的兼容性之外,其他不同类型 CPU 之间的汇编语言程序是无法兼容的,即汇编语言程序的通用性和可移植性较差。

(2)执行的高效性　汇编语言本质上是机器语言,程序员用汇编语言编写程序时,可以充分发挥自己的聪明才智,对机器内部的各种资源进行合理地安排,让它们始终处于最佳的使用状态,因而容易产生运行速度快、指令序列短小的高效率目标程序。

(3)编写源程序的繁琐性　汇编指令同机器指令一样具有功能单一、具体的特点。在编写汇编语言程序时,既要考虑机器资源的限制,又要考虑汇编指令语法上的细节及限制。这就使得编写汇编语言程序比较繁琐、复杂。一个简单的计算公式或计算方法,却要用一系列汇编指令一步一步来实现。

(4)调试的复杂性　调试汇编语言程序比调试高级语言程序困难,其主要原因如下:

1) 汇编指令涉及机器资源的细节,在调试过程中,要清楚每个资源的变化情况。

2) 程序员在编写汇编语言程序时,为了提高资源的利用率,可以使用各种实现技巧,而这些技巧完全有可能破坏程序的可读性。因此在调试过程中,除了要知道每条指令的执行功能,还要清楚它在整个解题过程中的作用。

3) 汇编语言中要用到大量的各类转移指令,这些转移指令增加了调试程序的难度。

4) 高级语言可以在源程序级进行符号跟踪,而汇编语言程序只能跟踪机器指令(不过,现在这方面有所改善,有的软件也可在源程序级进行符号跟踪)。

(5)硬件控制的直接性　汇编语言可以直接地、有效地控制计算机硬件部件,可以编写在“时间”和“空间”两方面最有效的程序,这使得汇编语言在程序设计中占有重要的位置,是不可被取代的。

## 1.2 学习汇编语言的意义

### 1.2.1 汇编语言的主要应用

根据汇编语言的特点,汇编语言的主要应用场合有以下几个方面:

(1) 程序要具有较快的执行时间,或者只能占用较小的存储容量。例如,操作系统的核心程序段,实时控制系统的软件,智能仪器仪表的控制程序等。

(2) 程序与计算机硬件密切相关,程序要直接有效地控制硬件。例如,I/O 接口电路的初始化程序段,外部设备的底层驱动程序等。

(3) 大型软件需要提高性能、优化处理的部分。例如,计算机系统频繁调用的子程序、动态连接库等。

(4) 没有合适的高级语言或只能采用汇编语言的时候。例如,开发最新的处理器程序时,暂时没有支持新指令的编译程序。

(5) 汇编语言还有许多实际应用,如分析具体系统尤其是该系统的底层软件、加密解密软件,分析和防治计算机病毒等。

汇编语言可以独立用于编写程序,也可以与高级语言配合使用,应用广泛。

### 1.2.2 为什么要学习汇编语言

如果说学习生命科学不了解 DNA 不行,学习物理科学不了解基本粒子不行,那么,研究计算机不研究指令系统及其应用也不行。通过以上对汇编语言应用场合的分析,可以看出,汇编语言作为最基本的编程语言之一,其重要性毋庸置疑。

汇编语言是面向机器的语言,通过对汇编语言的学习可以更深刻地理解计算机软件系统底层对硬件的控制、操作的方法和手段,有利于今后理解和学习操作系统原理;可以帮助我们深入了解微处理器的内部结构,熟悉端口和外部设备的关系,为掌握设备驱动程序的原理打下基础,也为掌握嵌入式操作系统打下基础;可以深入了解微处理器的指令体系,牢牢掌握面向机器语言程序设计的基本步骤和方法,为开发高质量、高效率的程序打下坚实基础。掌握了汇编语言程序设计方法的同学能够更容易地学习和掌握其他计算机语言。

汇编语言能够帮助我们深刻地了解 bit 和 byte 在数字化领域中的应用,有助于理解 bit 流在网络、通信、图像、视频、音频中的组织形式,有助于计算机工作者从信息编码的底层工作做起。

所以,汇编语言是理解整个计算机系统的最佳起点和最有效途径,通过学习和使用汇编语言,能够感知、体会、理解机器的逻辑功能,向上为理解各种软件系统的原理打下技术理论基础,向下为掌握硬件系统的原理打下实践应用基础。

### 1.2.3　学习 Linux 环境的汇编语言

因为汇编语言在各种处理器和汇编器之间是不同的，所以学习汇编语言程序设计的第一步是决定在什么样的环境下使用什么类型的汇编语言。本书使用运行在 IA－32 处理器系列机上的 Linux 操作系统作为平台，利用 GNU 开发环境进行汇编语言的程序设计。

Linux 是一款免费的操作系统，用户可以通过网络或其他途径免费获得，并可以任意修改其源代码，这是其他的操作系统所做不到的。正是由于这一点，来自全世界的无数程序员参与了 Linux 的修改、编写工作，程序员可以根据自己的兴趣和灵感对其进行改变。这让 Linux 吸收了无数程序员的精华，不断壮大。

就最基本的汇编语言程序设计而言，Windows 环境的汇编语言和 Linux 环境的汇编语言只是语法格式上的区别。而 Linux 提供丰富的程序开发工具，比如优化编译器、汇编器、连接器和调试器，它们的费用很低，或者是免费的。Linux 环境中这些丰富的开发工具使它非常适合把 C 程序剖析为汇编代码。这样的学习，除了可掌握基础的汇编语言程序设计外，还能帮助我们更好地理解编译器是如何工作的，高级语言程序是如何被转换为汇编语言的，以及如何掌握生成的汇编语言代码，更有利于我们在实际中用高级语言和汇编语言的配合编程。

## 1.3　汇编语言程序的组成

一般程序设计语言的源程序除了程序主体之外，还有相应的变量、类型、子程序等说明部分。从例题 1.1 中我们已经看到，汇编语言源程序也不只是由指令系统中的指令组成的，程序中一般还有指示源程序如何汇编、变量、子程序的定义等不产生 CPU 动作的说明性工作，这些工作在程序执行前由汇编程序完成处理，相应的语句被称为伪指令。与之相对应，常称使 CPU 产生动作并在程序执行时才处理的语句为硬指令或真指令。

### 1.3.1　语句格式

汇编语言编写的源程序由语句序列构成，每个语句由 1～4 个部分组成。

(1)执行性语句　执行性语句用于表达处理器指令(也称为硬指令)，汇编后对应一条机器指令代码。由处理器指令组成的代码序列是程序设计的主体。执行性语句的格式如下：

标号:硬指令助记符　操作数,操作数　#注释

(2)说明性语句　说明性语句用于表达伪指令，指示源程序如何汇编、变量怎样定义、过程怎么设置等。说明性语句的格式如下：

名字:伪指令助记符　参数,参数,……　#注释

#### 1.3.1.1 标号与名字

执行性语句中,冒号前的标号表示处理器指令在主存中的逻辑地址,主要用于指示分支、循环等程序的目的地址,可有可无。说明性语句中的名字可以是变量名、逻辑段名、子程序名等,反映变量、逻辑段和子程序等的逻辑地址。标号和名字都具有自身的各种属性。标号和名字是符合汇编语言语法的用户自定义的标识符。

标识符(identifier)一般最多由 31 个字母、数字及规定的特殊符号(如 _、$、?、@)组成,不能以数字开头(与其他高级程序语言一样)。在一个源程序中,用户定义的每个标识符必须是唯一的,不能和汇编程序采用的保留字一致。保留字(reserved Word)是编程语言本身需要使用的各种具有特定含义的标识符,也称为关键字。汇编程序中保留字主要有硬指令助记符、伪指令助记符、操作符、寄存器名以及预定义符号等。

#### 1.3.1.2 助记符

助记符是帮助记忆指令的符号,反映指令的功能。硬指令助记符表示一种处理器操作。不同的汇编器使用不同的助记符表示指令码。

例如,程序中使用最多的数据传送指令,其助记符是“mov”(取自 move),功能是将源操作数传送至目的操作数。如将数字 1 传送到寄存器 eax 中,指令表达为:

```
movl  $1,%eax
```

汇编语言源程序中使用最多的字符串变量定义伪指令,其助记符是“ascii”,功能是在主存中占用若干的存储空间,用于保存一串字符串的 ASCII 码。在 GNU 汇编器中数据类型是使用汇编命令声明的,命令前面有一个点号。例如,可以用 ascii 定义一个字符串,并使用变量名 msg 表达其在主存的逻辑地址:

```
msg:  .ascii "Hello, world! \n"
```

#### 1.3.1.3 操作数和参数

处理器指令的操作数表示参与操作的对象,可以是一个具体的常量,也可以是保存在寄存器的数据,还可以是一个保存在存储器中的数据。一条处理器指令中操作数的个数为 0 ~ 3 个,其中最常见的是含有 2 个操作数的双操作数指令。双操作数指令中,源操作数写在逗号前,目的操作数写在逗号后,用于存放指令操作的结果。

例如,指令“movl $1,%eax”中数字“1”是常量形式的源操作数,“eax”是寄存器形式的目的操作数。

伪指令里的参数可以是常量、变量名、表达式等,可以有多个,参数之间用逗号分隔。

#### 1.3.1.4 注释

语句中#号后的内容是注释,它通常是对该指令或该段程序功能的说明,是为了程序便于阅读而加上的,不是必须有的。必要时,一个语句行也可以由#号开始作为阶段性注释。汇编程序在翻译源程序时将跳过该部分,不对它们做任何处理。建议大家一定要养成书写注释的良好习惯。