面向“工程教育认证”
计算机系列课程规划教材

普通高等教育“十一五”
国家级规划教材

新概念汇编语言

◎ 杨季文 编著

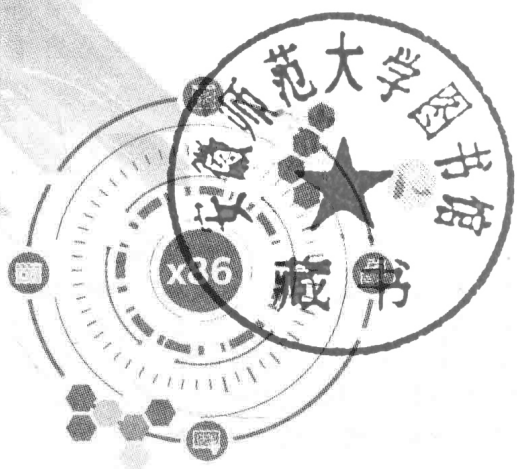
清華大學出版社

面向“工程教育认证”
计算机系列课程规划教材

普通高等教育“十一五”
国家级规划教材

新概念汇编语言

◎ 杨季文 编著



清华大学出版社
北京

内容简介

本书设定新目标，采用新方法，基于新平台，讲解 IA-32 结构系列(80x86 系列)CPU 的 32 位编程。本书分为 4 个部分：第一部分利用 VC 2010 环境的嵌入汇编和目标代码，介绍 IA-32 系列(80x86 系列)CPU 的基本功能和 32 位编程技术；第二部分利用开源汇编器 NASM、开源虚拟机 VirtualBox 和模拟器 Bochs，介绍汇编语言和计算机系统底层输入输出的实现方式；第三部分详细讲解保护方式编程技术，生动展示保护方式编程细节；第四部分简要说明相关软件工具的使用。

本书依托高级语言，讲解低级语言；利用虚拟平台，演示系统原理。第一部分和第二部分可作为高校计算机及电子信息类专业学生学习汇编语言的教材，第三部分可作为编程爱好者学习保护方式编程技术的教材或参考书。

图书在版编目(CIP)数据

新概念汇编语言/杨季文编著. —北京：清华大学出版社，2017
(面向"工程教育认证"计算机系列课程规划教材)
ISBN 978-7-302-47634-4

Ⅰ. ①新…　Ⅱ. ①杨…　Ⅲ. ①汇编语言—程序设计　Ⅳ. ①TP313

中国版本图书馆 CIP 数据核字(2017)第 155190 号

责任编辑：魏江江　王冰飞
封面设计：刘　键
责任校对：焦丽丽
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954
印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：31.75　　**字　　数**：812 千字
版　　次：2017 年 10 月第 1 版　　**印　　次**：2017 年 10 月第1 次印刷
印　　数：1～2000
定　　价：69.50 元

产品编号：053648-01

前言

随着计算资源的日益丰富和开发环境的日趋完善，直接运用汇编语言编写程序的场合越来越少，因此汇编语言课程需要新定位，汇编语言课程需要新概念。

在这样的背景下，本书设定新的目标，采用新的方法，基于新的平台，讲解 IA-32 结构系列(80x86 系列)CPU 的 32 位编程。学习汇编语言的新目标是深入理解计算机系统的工作原理，全面提升高级语言程序设计能力，而不再是熟练运用汇编语言编写程序。汇编语言课程将起到“上承高级语言，下启机器系统”的桥梁作用。学习汇编语言的新方法是依托高级语言。在学习汇编语言之前，通常已经具备高级语言(C 或者 C++语言等)程序设计的基础。通过采用嵌入汇编和分析目标代码等方法，不仅可以降低学习和掌握汇编格式指令的难度，而且有助于“知其然，知其所以然”，有助于更好地掌握高级语言。实践汇编语言的新平台是虚拟机。目前虚拟机已经十分流行，它是很理想的“裸机”。基于虚拟机不仅可以突破操作系统的约束，为所欲为地操纵“机器”，从而轻松调试设备驱动程序或者系统程序，而且有助于熟悉计算机系统的启动过程，有助于明了计算机系统硬件和软件的相互关系。

本书分为 4 个部分，共 10 章。第一部分由前五章组成，利用 VC 2010 环境的嵌入汇编和目标代码，讲解 IA-32 系列(80x86 系列)CPU 的基本功能和 32 位编程技术。第 1 章介绍基础知识；第 2 章说明 IA-32 系列 CPU 的基本功能；第 3 章和第 4 章讲解利用 IA-32 系列 CPU 的指令设计程序；第 5 章分析 VC 源程序的目标代码。第二部分由第 6 章、第 7 章和第 8 章组成，利用汇编器 NASM 和虚拟机，讲解汇编语言和系统输入输出。第 6 章基于汇编器 NASM 介绍汇编语言；第 7 章在介绍 BIOS 和主引导记录之后，说明虚拟机的原理及其使用方法；第 8 章基于虚拟机讲解计算机系统底层输入输出的实现方式。第三部分是第 9 章，详细讲解基于 IA-32 系列 CPU 的保护方式程序设计，该章内容十分丰富。第四部分是第 10 章，简要说明相关工具的使用，包括开源汇编器 NASM、开源虚拟机 VirtualBox 和开源模拟器 Bochs 等。

本书依托高级语言，讲解低级语言；利用虚拟平台，演示系统原理。第一部分和第二部分可作为学习汇编语言的教材，第三部分可作为学习保护方式编程技术的教材或参考书。本书还提供教学用 PPT。

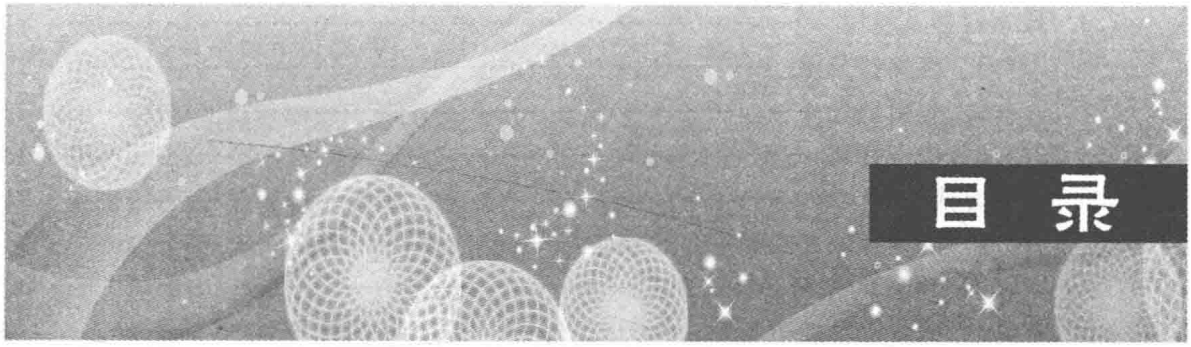
杨季文撰写第 1～4 章和第 6～9 章，朱晓旭撰写第 5 章，胡沁涵撰写第 10 章，赵雷参与部分工作。杨季文负责全书统稿、定稿。

本书得到了指导老师钱培德教授的大力支持，在此表示衷心感谢。本书还得到了同事朱巧明、吕强、李云飞和李培峰等教授的大力帮助，在此表示感谢。还要感谢同事卢维亮、查伟忠、陈宇和王莉等老师的帮助。

由于编者时间仓促和水平所限，书中难免有不妥之处，恳请读者批评指正。

作　者

2017 年 6 月



目 录

第1章 基础知识

利用计算机系统进行科学计算或者信息处理，本质上是运行各种程序。在普通计算机系统中，由作为核心的CPU执行程序。本章首先介绍CPU的基本功能，然后介绍汇编语言的相关基本概念，最后说明数据的表示和存储方法。

1.1 CPU简介

计算机系统的核心是中央处理器，也就是CPU。本节结合可由IA-32系列CPU执行的目标代码，介绍CPU的基本功能。

1.1.1 目标代码

计算机系统中的CPU只能执行机器指令，也就是说只能运行由机器指令组成的程序。利用相应的编译器，可以把由高级语言编写的源程序转换成由机器指令组成的程序，也就是目标程序，又被称为目标代码。无论源程序是用哪种语言编写的，无论程序是作为简单的小软件还是构成复杂的大系统，计算机系统最终运行的是对应的目标程序。

如下用C语言编写的函数cf11的功能是计算1至10的平方和。为了较好地体现目标代码和反映CPU的功能，这个函数采用循环累加1至10之间每个整数的平方。为此安排了两个变量，一个存放累加和，另一个用于计数。

```
int  cf11(void)
{
     int  sum, i;
     sum=0;
     for(i=1; i<=10; i+=1)
         sum+=i*i;
     return  sum;
}
```

利用微软Visual Studio 2010的VC集成开发环境，在采用编译优化选项“使大小最小化”的情况下，编译上述程序，可得到如下所示的IA-32系列CPU的目标代码。为了便于理解，这里的目标代码采用了汇编格式指令的表示形式，而真正的目标代码是用二进制编码的。

```
    xor   ecx, ecx
    xor   eax, eax
    inc   ecx
$LL3@cf11:
    mov   edx, ecx
```

```
imul  edx, ecx
add   eax, edx
inc   ecx
cmp   ecx, 10
jle   $LL3@cf11
ret
```

可以这样认为，以 IA-32 系列处理器为 CPU 的计算机执行上述 C 函数 cf11，实际上就是执行这段目标代码。也就是说，这段目标代码实现了 C 函数 cf11 的功能。现在结合上述 C 函数 cf11，解释这段目标代码。

上述目标代码由 10 条指令组成。其中，EAX、ECX、EDX 分别是 IA-32 系列 CPU 内部的寄存器，这些寄存器可用于存放运算数据和运算结果。这些寄存器类似于 C 语言源程序中的变量。

开始两条指令中的 XOR 表示将两个数据进行“异或”运算，两个相同的数据进行“异或”运算的结果是 0。所以，执行指令“xor ecx, ecx”就使得寄存器 ECX 的值为 0。第三条指令中的 INC 表示将寄存器或者存储单元的值加 1，执行指令“inc ecx”就使得寄存器 ECX 的值加 1。至此可以看到，寄存器 EAX 和 ECX 分别代表了函数 cf11 中的变量 sum 和 i，第二条指令“xor eax, eax”对应语句“sum=0”，第一条指令和第三条指令一起对应语句“i=1”。

第四条指令中的 MOV 表示将数据传送到寄存器或存储单元。执行指令“mov edx, ecx”会把寄存器 ECX 的值传送到寄存器 EDX。第五条指令中的 IMUL 表示将两个数据进行相乘运算，执行指令“imul edx, ecx”就使得寄存器 EDX 的值和寄存器 ECX 的值相乘，乘积存放在寄存器 EDX 中，这里实际上就是计算平方。第六条指令 ADD 表示将两个数据进行相加运算，执行指令“add eax, edx”就使得寄存器 EAX 的值和寄存器 EDX 的值相加，其和存放在寄存器 EAX 中。所以，下面 3 条指令对应上述函数 cf11 中的语句“sum+=i*i”：

```
mov   edx, ecx
imul  edx, ecx
add   eax, edx
```

第七条指令仍然是“inc ecx”，执行它使得寄存器 ECX 的值加 1。在这里该指令相当于函数 cf11 中的语句“i+=1”。

接下来的指令“cmp ecx, 10”和指令“jle $LL3@cf11”的具体操作是把寄存器 ECX 的值与 10 进行比较，当寄存器 ECX 的值小于等于 10 时，跳转到标号为 $LL3@cf11 的指令处执行，否则顺序执行下一条指令。在这里就是判断是否要继续循环累加，如果变量 i 的值没有超过 10，则继续计算平方和。

最后一条指令“ret”表示返回到调用者。在这里就代表函数 cf11 结束返回。

虽然上述目标代码比用 C 语言编写的函数 cf11 烦琐和难以理解，但在计算机系统中就是这样处理的。通过学习汇编语言程序设计，可以深入地理解计算机系统的处理方式。

1.1.2 基本功能

CPU 是计算机系统的核心，无论计算机系统处理什么，最终都归结为 CPU 执行机器指令进行相应的运算或者处理。

CPU 的基本功能是执行机器指令、暂存少量数据和访问存储器。

1. 执行机器指令

CPU 能够一条接一条地依次执行存放在存储器中的机器指令。也就是说，CPU 能够自动执行存放在存储器中的由若干条机器指令组成的目标程序或目标代码。例如，对于在 1.1.1 节中以汇编格式指令的形式列出的目标代码，CPU 从执行指令“xor　ecx，ecx”开始，依次顺序执行随后的各条指令，在执行指令“jle　$LL3@cf11”时，如果满足小于等于的条件，就接着从指令“mov　edx，ecx”开始依次顺序执行，否则执行下一条指令“ret”。

CPU 能够直接识别并遵照执行的指令被称为机器指令。一款 CPU 能够执行的全部机器指令的集合，被称为该 CPU 的指令集。

每一条机器指令的功能通常是很有限的。一条高级语言的语句所完成的功能，往往需要几条机器指令，或者几十条机器指令，甚至几百条机器指令才能够实现。例如，在 1.1.1 节 C 函数 cf11 中，虽然语句“sum＋＝i＊i”很简单，并且变量 sum 和 i 分别由寄存器担当，该语句的功能仍需要三条机器指令来实现。

CPU 决定机器指令。不同种类的 CPU，其指令集往往不相同。有的 CPU 指令集比较小，有的 CPU 指令集比较大。但是，同一个系列 CPU 的指令集常常具有良好的向上兼容性，即下一代 CPU 的指令集通常是上一代 CPU 指令集的超集。例如，Intel 80386 处理器的指令集包含了早先的 8086 处理器的指令集，Pentium 处理器的指令集包含了 Intel 80386 处理器的指令集。

按指令的功能来划分，通常机器指令可分为以下几大类：数据传送指令、算术逻辑运算指令、转移指令、处理器控制指令和其他指令等。例如，在 1.1.1 节列出的目标代码片段中，MOV 指令属于数据传送指令，ADD 指令、IMUL 指令、INC 指令属于算术运算指令，XOR 指令属于逻辑运算指令，JLE 指令和 RET 指令属于转移指令。

2. 暂存少量数据

一个目标程序中的绝大部分指令是对数据进行各种运算或者处理。例如，在 1.1.1 节列出的目标代码片段中，共计 10 条指令，有 8 条是对数据进行运算或者处理的。参与运算的数据存放在哪里？运算的结果存放在哪里？一般来说，运算数据和运算结果可以存放在寄存器中，也可以存放在存储器中。

CPU 有若干个寄存器，可以用于存放运算数据和运算结果。例如，在 1.1.1 节列出的目标代码片段中，就充分运用了寄存器 EAX、ECX、EDX 存放运算数据和运算结果，寄存器 EAX 和 ECX 分别作为变量 sum 和 i。

利用寄存器存放运算数据和运算结果，效率是最高的。寄存器在 CPU 内部，处理寄存器中的数据要比处理存储器中的数据快得多，因此一般总是尽量利用寄存器。这也就是为什么在 C 函数 cf11 的目标代码中用两个寄存器分别作为局部变量的原因。在采用编译优化选项“使大小最小化”的情况下，VC 2010 编译器生成这样的目标代码；否则，在默认情况下，采用存储单元存放普通局部变量。

指令集中大部分指令的操作数据至少有一个在寄存器中。在目标程序中，绝大部分的指令都使用到寄存器。

但是，CPU 内可用于存放运算数据和运算结果的寄存器数量是很有限的。例如，IA-32 系列 CPU 只有 8 个这样的通用寄存器。通常，编译器会充分利用 CPU 内的寄存器。在用汇编语言编写程序时，也必须注意灵活运用寄存器。

3. 访问存储器

由机器指令组成的目标程序在存储器中，待处理的数据也在存储器中，CPU 要执行目标程序，就要访问存储器。这里存储器指 CPU 能够直接访问的计算机系统的物理内存。

存储器(内存)由一系列存储单元线性地组成，最基本的存储单元为一个字节。为了标识和存取每一个存储单元，给每一个存储单元规定一个编号，也就是存储单元地址。

通常，CPU 支持以多种形式表示存储单元的地址。一些功能较强的 CPU 还支持以多种方式组织管理存储器。

设有以下 C 程序片段，其中 x 和 y 是两个全局变量，函数 cf12 进行简单的数据处理。

```
int x=1;
int y=2;
void  cf12(void)
{
    y=x*x+3;
    return;
}
```

利用微软 Visual Studio 2010 的 VC 集成开发环境，在采用编译优化选项“使大小最小化”的情况下，编译上述程序，对应 C 函数 cf12 的目标代码如下所示。为便于理解，目标代码采用汇编格式指令的表示形式。

```
mov   eax, varx3HA
imul  eax, eax
add   eax, 3
mov   vary3HA, eax
ret
```

在上述以汇编格式指令的形式表示的目标代码中，符号 varx3HA 和 vary3HA 分别代表存放全局变量 x 和 y 的存储单元。指令“mov　eax,varx3HA”的功能是把全局变量 x 所在的存储单元的内容取到寄存器 EAX 中，换句话说就是把内存中的全局变量 x 送到寄存器 EAX。类似地，指令“MOV　vary3HA,eax”的功能是把寄存器 EAX 中的内容送到全局变量 y 中。

运行函数 cf12，就是执行上面的这些指令。在执行指令“mov　eax,varx3HA”后，寄存器 EAX 就含有变量 x 的值；执行指令“imul　eax,eax”，实现把寄存器 EAX 与 EAX 相乘，这样寄存器 EAX 就有 x * x 的积；执行指令“add　eax,3”，使寄存器 EAX 所含值再加 3；执行指令“mov　vary3HA,eax”，把最后的结果送到变量 y；执行指令“ret”，从函数返回。

1.2　汇编语言概念

由 CPU 执行的机器指令采用二进制编码，人们很难识别和理解，为此采用符号表示，成为汇编格式指令。本节在介绍机器指令和汇编格式指令的基础上，介绍汇编语言的概念。

1.2.1　机器指令

把 CPU 能够直接识别并遵照执行的指令称为机器指令。

机器指令一般由操作码和操作数两部分构成。操作码指出要进行的操作或运算，如加、减、传送等。操作数指出参与操作或运算的对象，也指出操作或运算结果存放的位置，如寄存

器、存储单元和数据等。

机器指令采用二进制编码表示。换句话说，就是用二进制代码表示机器指令。例如，把通用寄存器 EAX 与 EDX 相加，结果存放到 EAX 中的机器指令如下所示：

```
0000 0011 11 000 010
```

对机器指令的编码是按一定规则进行的。编码中有一部分代表操作码，还有一部分代表操作数。例如，对于上述的编码，前面的 8 位(0000 0011)代表加法操作，也就是操作码部分；后面的 8 位(11 000 010)代表两个通用寄存器 EAX 和 EDX，也就是操作数部分。

为了阅读和书写方便，常用十六进制形式或八进制形式表示二进制编码的机器指令。例如，1.1.1 节所列的 C 函数 cf11，在经过 VC 2010 编译后得到如下所示的真正的目标代码，其中，每一行分别是采用十六进制形式表示的 IA-32 系列 CPU 的机器指令。

```
33 C9
33 C0
41
8B D1
0F AF D1
03 C2              ;0000 0011 11 000 010
41
83 F9 0A
7E F3
C3
```

上述目标代码片段犹如天书，几乎没有人能直接看出它的功能。因此，程序员难以用机器指令编写程序，更难写出健壮的程序；用机器指令编写出的程序也不易被人们理解、记忆和交流。所以，只是在计算机出现早期时才用机器指令编写程序，现在几乎没有人用机器指令编写程序了。

1.2.2　汇编格式指令

为了克服机器指令的上述缺点，人们采用便于记忆、并能描述指令功能的符号来表示指令的操作码。这些符号被称为指令助记符。助记符一般是说明指令功能的英语词汇或者词汇的缩写。例如，前面所见的符号 MOV 和 ADD 等。同时，也用符号表示操作数，如寄存器、存储单元地址等。这样就有了汇编格式指令。

用指令助记符、地址符号等表示的指令称为汇编格式指令。汇编格式指令的一般格式如下：

[标号:] 指令助记符 [操作数表]

其中，指令助记符是必需的。操作数随指令而定，有的指令有一个操作数，有的指令有两个操作数，有些指令有 3 个操作数，还有些指令没有操作数。如果有多个操作数，操作数之间用逗号分隔。标号可有可无，标号后带一个冒号。指令助记符与操作数表之间用空格或制表符分隔。

汇编格式指令与机器指令一一对应。例如，1.1.1 节所列的 C 函数 cf11 的机器指令和汇编格式指令的对应情况如下所示(为了对齐效果把标号 $LL3@cf11 换成了标号 lab1)：

```
33 C9                    xor   ecx, ecx
```