北京希望电脑公司汇编语言技术丛书

面向目标的汇编语言程序设计

严 吾 编译

海洋出版社

北京希望电脑公司汇编语言技术丛书

面向目标的汇编语言程序设计

严 吾 编 译

海 洋 出 版 社

内 容 提 要

本书通过介绍汇编语言的宏定义技巧，为我们展示了面向目标的程序设计风格和程序设计方法。通过书中大量的宏定义，读者可以掌握建立类，方法和目标的手段，还可以学会编写可重复使用代码的技巧。作为副产品，读者得到了大量的汇编语言源程序。

本书供已掌握了汇编语言，又想了解面向目标的程序设计原理的读者使用。

欲购本书的用户，请直接与北京8721信箱联系，邮政编码100080，电话 2562329。

面向目标的汇编语言程序设计

严 吾 编译

希 望 校

责任编辑：阎世尊

* * *

海洋出版社（北京市复兴门外大街1号）

海洋出版社发行 双青印刷厂印刷

* * *

开本：787×1092毫米1/16 印张：19.18 字数：388千字

1991年11月第一版 1991年11月第一次印刷

印数：1—3000册 定价：13.00元

ISBN7-5027-2206-8/TP·34

译 者 的 话

面向目标的设计方法已经开始在程序设计领域传播。目前对这种方法有各种各样的评论。有人认为它创立了一种新的软件工程方法，也有人持相反的看法。面对这种情况，作为一个程序员，应该冷静地研究一下各种事实，做出自己的判断。

面向目标的设计不是指几种语言，而是指一种分解问题的方法。这种方法的基本原理是信息分类和信息隐藏。目前Smalltalk和C++都是带有面向目标特征的程序设计语言，但这种特征和语言之间并没有不可分离的关系。本书利用我们所熟悉的宏汇编语言作为基础，为它增加了面向目标的设计方法的一些特性。从另一个角度证明了面向目标的方法并不是神秘的魔术。

作者 Len Dorfman 利用汇编语言中的宏定义能力以及面向目标的程序设计思想构造了一组有力的程序设计工具。它可以缩短程序开发时间，提高代码可读性，简化程序维护的工作量，并且可以重新利用原有的代码。

把面向目标的原理与实际相结合是本书的一大特点。通过阅读本书，读者可以掌握用汇编语言

- 建立类，方法和目标
- 编写可重复使用的代码
- 简化字符串操作
- 订做键盘控制方法
- 在屏幕上显示正文
- 控制打印机的方式
- 编写弹出式窗口
- 编写EGA图形程序
- 建立图符（ICON）

实际上，提供以上技术只是介绍面向目标方法的某些副产品，读者应该通过学习掌握一种新的程序设计或软件工程方法。

目　　录

第一章　什么是面向目标的设计 …… (1)
1.1　类 …… (1)
1.2　方法 …… (2)
1.3　继承性 …… (4)
1.4　实例 …… (5)
1.5　多态 …… (5)
1.6　汇编语言中能使用面向目标的方法吗 …… (5)
1.7　总结 …… (6)
第二章　结构化汇编语言的回顾 …… (6)
2.1　宏定义 …… (7)
2.2　方法：内存模式和调用约定 …… (8)
2.3　文件 DEFINES.MAC …… (8)
2.4　文件 REGSEXT.MAC …… (15)
2.5　文件O_CHUCK.MAC …… (19)
2.6　控制流宏定义的命名约定 …… (38)
2.7　总结 …… (40)
第三章　汇编语言：一个简单的面向目标设计实例 …… (40)
3.1　类 EXE …… (41)
3.2　方法：初始化显示器 …… (49)
3.3　方法：设置屏幕属性 …… (51)
3.4　方法：清除屏幕 …… (52)
3.5　方法：在屏幕上显示正文 …… (54)
3.6　方法：等待按键 …… (56)
3.7　总结 …… (64)
第四章　CURSOR 类和光标处理方法 …… (65)
4.1　方法：在屏幕上写字符 …… (65)
4.2　方法：在屏幕上显示字符串 …… (66)
4.3　类 CURSOR …… (69)
4.4　方法：得到和设置光标位置 …… (70)
4.5　方法：得到和设置光标形状 …… (73)
4.6　方法：保存和恢复光标位置和形状 …… (75)
4.7　方法：取得和设置光标信息 …… (80)
4.8　方法：保存和恢复光标信息 …… (83)
4.9　方法：显示和隐藏光标 …… (89)

4.10 方法：按相对位置移动光标……（91）
4.11 总结……（97）
第五章 STRING类和字符串处理方法……（99）
5.1 方法：整数转换成ASCII……（99）
5.2 方法：整数转换成十六进制数……（105）
5.3 类STRING……（111）
5.4 方法：复制字符串……（112）
5.5 方法：从一个字符串中复制几个字节……（114）
5.6 方法：取字符串长度……（117）
5.7 方法：把字符串转换成大写或小写……（120）
5.8 方法：对字符串进行初始化……（124）
5.9 方法：把字符串中某些字节替换成新值……（129）
5.10 方法：字符串连接……（132）
5.11 方法：比较两个字符串……（135）
5.12 方法：字符串交换……（138）
5.13 类MEMORY……（141）
5.14 方法：内存区置初值……（141）
5.15 方法：复制内存区……（143）
5.16 方法：复制指定数目的字节……（145）
5.17 总结……（148）
第六章 KEY类和键盘处理方法……（151）
6.1 过程：ISPRINT……（151）
6.2 类KEY……（153）
6.3 方法：得到按键状态……（153）
6.4 方法：取键盘标志……（157）
6.5 方法：读字符串……（162）
6.6 总结……（169）
第七章 VIDEO和SCREEN类及屏幕处理程序……（169）
7.1 类VIDEO……（169）
7.2 方法：设置显示页……（170）
7.3 方法：设置显示方式……（171）
7.4 类SCREEN……（174）
7.5 方法：从屏幕上读一个字符……（174）
7.6 方法：从屏幕上读一串正文……（177）
7.7 方法：保存和恢复屏幕……（178）
7.8 方法：改变屏幕上字符串的属性……（182）
7.9 总结……（189）
第八章 PRINT类和打印方法……（189）
8.1 类PRINT……（190）

8.2 文件O_PRINT.MAC …… (190)
8.3 方法：print_character …… (196)
8.4 方法：get_print_status …… (197)
8.5 方法：print_screen …… (202)
8.6 方法：print_string和print_form_feed …… (203)
8.7 方法：set_print_column_location …… (204)
8.8 方法：print_byte_sequence …… (205)
8.9 总结 …… (206)
第九章 PARAMETER类和参数处理方法 …… (212)
9.1 类PARAMETER …… (213)
9.2 方法：取参数数目 …… (216)
9.3 方法：取命令行参数 …… (218)
9.4 方法：把参数同字符串比较 …… (220)
9.5 总结 …… (223)
第十章 在文本方式下建立EGA图符 …… (228)
10.1 文本方式下显示图符的程序 …… (230)
10.2 类EGA …… (236)
10.3 类EGA中的方法 …… (236)
10.4 总结 …… (243)
第十一章 EGN图符编辑器 …… (244)
11.1 类FILE和文件处理方法 …… (244)
11.2 类WINDOW和窗口处理方法 …… (249)
11.3 图符编辑器的源程序 …… (254)
11.4 总结 …… (288)

第一章　什么是面向目标的设计

面向目标的设计是一种成熟的信息分类手段。它可以被应用在计算机科学领域。带有面向目标的设计特征的程序设计语言主要强调建立可重用的代码模块，提高程序员生产率，增加源程序的可读性，以及减少维护程序所花费的时间。

很多推崇面向目标技术的人都相信面向目标的程序设计语言会成为改变程序员思考方式和工作方式的新潮流。他们还认为使用面向目标的设计模型对问题求解的过程会有巨大的影响。

尽管面向目标的设计方法中很多内容都不是最近才出现的，但我们仍然要介绍一下这个领域中的一些特殊词汇。直接介绍这些内容可能会使读者感到过于抽象，但很快就会通过接触带有面向目标性质的汇编语言源代码消除这种感觉。

面向目标的设计方法后面实际上是一种分类原则。在过去的生物学中有一套很好的分类体系。但在程序设计语言领域中，这种面向目标的程序设计主要包含对数据的分类以及对操纵这些数据的过程和函数分类。当然，对于汇编语言程序员来说，这种分类方法的优点是不容易一下子就看出来的。

从实用的观点来看，面向目标的程序设计语言或者程序设计语言中面向目标的扩展功能实际上都提供了一种对计算机语言能力的扩充方法。本书通过汇编语言的宏定义模拟一些面向目标的功能，得到了标准汇编语言的一个超集。汇编语言的高效率特性仍然被保留下来。分类方法不是唯一的，而是整个汇编语言环境中很多种实用方法中的一种。

1.1　类

在我们的讨论中，类由一组目标（数据）和一组描述类中实例的行为的方法(过程)组成。如果一个类包含目标，一旦目标被说明，就会建立该类中的一个实例。类中的方法只作用于该类中被说明的目标上。

本书中所做的面向目标的模拟并不是说所有的类都必须带有目标。那些不包含目标的类是不需要说明其方法的。

面向目标设计方法的术语	普通词汇
目标	数据元素
方法	子程序，过程，函数

下面先研究一个具体的例子，从中你可以看到关于类的一些描述。PC机的程序员都很熟悉屏幕上闪动的光标。由于这个光标可以通过位置和形状（目标）以及改变它的手段（方法）来描述，因此有理由建立一个名字为CURSOR的类。

一旦你决定建立一个类，就应该先列出该类中的目标。下面是CURSOR类中的所有目标。

```
;.....
;
;objects used in CURSOR CLASS
;
;
;LOCATION structure
;

LOCATION STRUC
    lrow    DW    ?      ;row
    lcol    DW    ?      ;column
LOCATION ENDS

;
;CURSOR_SHAPE structure
;

CURSOR_SHAPE STRUC
    cs_top    DB    ?      ;0-4 bits top scan line
    cs_bot    DB    ?      ;0-4 bits bottom scan line
CURSOR_SHAPE ENDS

;
;CURSOR structure inherits LOCATION and CURSOR_SHAPE
;

CURSOR STRUC
    cur_loc    LOCATION       <>    ; row & column
    cur_shape CURSOR_SHAPE    <>    ;start & end cursor shape
CURSOR ENDS

;
;.....
```

1.2 方法

所谓方法是改变某个类中目标行为的过程或函数。

首先观察一下CURSOR中的目标，然后再看一下如何改变光标的行为。你可以移动光标，改变光标尺寸，使光标出现或消隐，以及得到当前光标的状态。在本书中，为了描述类中的方法，我们在宏定义的名字上下了一番功夫。既要在名字中反映出面向目标的设计原理，也要容易记忆和容易理解。

因此，对方法的命名我们采取如下的约定：

- 方法中必须包含类的名字
- 方法中必须包含动作和行为修饰内容
- 动作必须出现在类名以前，并用下划线连接
- 行为修饰符必须出现在类名之后，并用下划线连接。在某些情况下可能会有多个行为修饰符

方法名的语法形式为：

〔动作〕_类名_〔行为修饰符〕

在我们的例子中，可以用这个原则构造移动光标的方法名。

1.2.1 移动光标的方法名

成份	描述
set	动作
cursor	类名
location	行为修饰符

实际构成的方法名为：

set_cursor_location

尽管set_cursor_location不是标准的英文，但它能清晰地表达出必要的信息。下面我们再看一个改变光标形状的方法名。

1.2.2 改变光标形状的方法名

成份	描述
set	动作
cursor	类名
shape	行为修饰符

实际构成的方法名为：

set_cursor_shape

在我们进一步工作以前，先列出名字为CURSOR的类的正规定义。

```
Class CURSOR

objects

cur_location
cur_shape

Mefhods

get_cursor_info
set_cursor_info
get_cursor_location
set_cursor_location
set_cursor_shape
set_cursor_shape
set_cursor_relative
set_cursor_up_one
set_cursor_dn_one
set_cursor_rt_one
set_cursor_lt_one
set_cursor_newline
save_cursor_info
restore_cursor_info
save_cursor_location
restore_cnrsor_location
save_cursor_shape
restore_cnrsor_shape
display_cursor
remove_cursor
```

CURSOR类中的所有方法都包含类名cursor。程序员可以立刻知道该方法是对哪个类进行操作的。在方法名display_cursor中，display为动作，但这里没有行为修饰符。这个方法把光标显示在屏幕上。

想让方法名成为完全合乎语法的句子是不现实的。程序员应该在适当的时候尽量少用行为修饰符。

在某些情况下，类名本身就带有强烈的动作意味。例如，在类PRINT中，可以用print_byte作为向打印机输送字节的方法名。

有些时候方法名的动作信息不仅没有使该方法的意义更加清晰，反而破坏了方法名的语义完整性。这时采用较短的方法名反而更加有利。

下面开始定义CURSOR类。在汇编语言中要为实现这个类做两件事情。首先，要把CURSOR类中的目标说明成数据段中的CURSOR结构，然后要为一些操作给出方法名的宏定义。这些方法都定义在一个名为O_CURSOR.MAC的文件中。

这里要尤其注意类、动作和行为修饰符的命名方法，它们大大地增强了源代码的可读性。实际上，当你要增加一些类时，也必须为它们设计出适当的名字。面向目标的设计就意味着在工作以前先把问题想清楚。

建立宏定义方法名的过程将在后面讨论，这里我们先看一下如何在数据段中说明目标，进而介绍面向目标的设计方法中的另一个概念：继承性。

1.3 继承性

目标（数据）是可以被继承的。当一个被定义的目标作为另一个目标的成份时，就出现了继承。

例如，我们可以先说明一个称为LOCATION的目标。它具有下面的结构：

```
LOCARION  STRUC
        row   DW   ? ; 存放行值
        col   DW   ? ; 存放列值
LOCATION  ENDS
```

尽管目标LOCATION在类CURSOR中非常重要，但它只表示出CURSOR类中目标的部分信息。当然，目标LOCATION也可以用在其它类中。

为了完成对类CURSOR中目标的描述，我们可以在LOCATION的基础上再增加一些数据。最终得到的CURSOR类应该有如下的形式：

```
CURSOR  STRUC
              cur_loc  LOCATION < >
              cur_top  DB ?
              cur_bot  DB ?
CURSOR  ENDS
```

一旦定义了目标LOCATION，其它包括LOCATION的类中就不必再定义该目标了。它们可以直接继承LOCATION的特性。在汇编语言中，它是用结构的嵌套实现的。

1.4 实例

一个类的实例是由源程序中对该类的目标结构的说明建立的。在我们的例子中，类的实例的名字与类的目标结构的名字是相同的。

如果你要定义CURSOR的一个实例，可以采用下面的语句：

```
entry  CURSOR  〈 〉
```

这表明CURSOR类中有一个名为entry的实例。在汇编语言中对实例的说明与对结构的说明具有同一种形式。

1.5 多态

在本书中，多态是指允许类名和行为修饰符互相转化。例如，在CURSOR类中有一个方法名：

```
display_cursor
```

它当然是表示显示光标。同样，在类WINDOW中，方法名：

```
display_window
```

表示显示某个窗口。

仔细研究这种结构我们会发现，类名前的动作在方法名中是包含着常识性知识的。在前面的例子中，动作名是不太会引起混淆的，但是当允许存在继承性时，动作名的选择确实是有一定难度的。很可能类名会在继承性的作用下转化为方法名中的行为修饰符。

例如，假定有一个名为LOCATION的类，有两个与类LOCATION相关的方法如下：

```
set_location
get_location
```

这里，set和get是动作，location是类名。如果在CURSOR类中继承LOCATION类中的目标，可能会得到另外两个方法名：

```
set_cursor_location
get_cursor_location
```

在这里，LOCATION类中的方法名set_location和get_location已经变成了CURSOR类中的方法名的行为修饰符。

这种写法在逻辑上和语义上都是有道理的。当你使用set_location时，表示要为目标location设置一个值。而当你使用set_cursor_location时，则表示把光标移到指定位置。

1.6 汇编语言中能使用面向目标的方法吗？

答案是可以。作者曾经在《结构化汇编语言》一书中使用过移动光标的宏定义mvcur。另外在《建立C语言库：窗口，菜单和用户接口》一书中也使用过C语言函数mvCur（　）。但它们都不如面向目标的方法中所使用的形式便于理解。在C语言中：

```
mvCur(10, 20);
```

在汇编语言中：

```
mvcur  10, 20;
```

而在面向目标的汇编语言中：

```
set_cursor_location  10, 20;
```

很明显，面向目标的设计方法中隐含着一组对方法的命名规则。这种规则可以帮助提高程序的可读性。

下面让做一个小小的实验，假定有下面几个在屏幕上显示字符串的过程名，要求我们选择出一个最容易读懂和最直观的：

```
puts
write
writeln
scwrite
print
write—screen—string
```

我们当然都会选择write—screen—string。

1.7 总结

面向目标的设计方法与程序语言无关。在本书中，我们通过汇编语言中的宏定义来模拟面向目标的设计特性。面向目标的方法要求程序员用分类的方法思考问题。在面向目标的设计方法中使用了一些特殊的词汇。下面我们逐一列出其中的主要部分。

• 类

它是目标和方法的组合体，该方法描述该类实例的行为。当在源程序中说明目标结构时，也就说明了类。应用于该类的方法被包含在另一个宏定义文件中。

• 方法

过程或函数，它与指定类相关，可以改变类中的目标。

• 继承性

一个被定义的目标被用作另一个目标的成份时，就出现了继承。

• 实例

当与某个类相关的目标结构在源程序中被说明时，就得到了一个实例。

• 多态

指方法中的通用动作。这种通用动作可以超越类的限制。

第二章　结构化汇编语言的回顾

尽管结构化程序设计是在高级语言中提出的，但它在汇编语言中也是一个重要技术。在《结构化汇编语言》一书中，作者利用宏定义实现了条件转移，for／next循环，switch和case语句等等。

本书在控制流宏定义的基础上增加了面向目标的内容，并把它们存放在O_CHUC-K．MAC文件中。本书中的程序流控制宏定义已经在《结构化汇编语言》一书的基础

上进行了一些改进。条件转移和循环不再限制在128字节以内。虽然这会在实际使用时增加几个字节，但却给程序员带来了很大的方便性。

为了便于理解和使用本书所提供的样本程序，我们先介绍一下宏定义的基本用法。虽然这种介绍不会面面俱到，但却能给不熟悉宏定义的程序员一个基本思路。

在讨论过宏定义以后，我们给出三个源文件。它们的具体内容如下：

• DEFINES.MAC

它包含大量等值定义，以便用名字替代常用的数值，如键盘扫描码，屏幕颜色等等。

• REGSEXT.MAC

它使用简单的宏定义对标准的8086指令集进行有效的扩充。

• O_CHUCK.MAC

它包含控制流宏定义。其中吸收了面向目标的方法，并以此来实现高级语言的结构。

2.1 宏定义

汇编语言的宏定义功能允许你把一组正文赋给一个宏定义名字。在定义了这个宏之后，可以在源程序中通过名字引用这个定义。当汇编程序遇到一个宏时，先检查宏定义的内容，然后用这个内容替换宏定义的名字。这个过程被称为宏定义的扩展。假定我们定义一个宏把寄存器AX，BX，CX和DX压入栈中：

```
PUSHX   MACRO
        push  AX
        push  BX
        push  CX
        push  DX
ENDM
```

另外我们还有一个从栈中弹出寄存器的宏定义：

```
POPX    MACRO
        pop   DX
        pop   CX
        pop   BX
        pop   AX
ENDM
```

通过宏定义PUSHX可以把8086四个通用寄存器的值压入栈中。在执行完相应任务以后，可以用popx在栈中恢复通用寄存器的值。在源程序中的使用形式如下：

```
PUSHX
call  any-routine
POPX
```

当汇编程序遇到PUSHX时，就按着宏定义的内容把它展开。然后会调用过程any-routine。该过程返回后，再用POPX恢复寄存器的值。

宏定义可以像高级语言的过程或函数那样接收参数。随着本书的展开，你会遇到越来越复杂的宏定义。如果你需要了解有关宏定义的更详细的信息，可以参考宏汇编程序的手册。

2.2 方法：内存模式和调用约定

在本书中，所有的方法都被设计成在小模式下工作。小模式要求程序的代码段和数据段都不超过64K。当然，把这里介绍的内容改成中模式或大模式也是很容易的。

只要汇编子程序中包含了OOPSTAB.LIB中的用宏定义编写的方法，就必须遵守一系列调用约定。如果为了节省代码而通过寄存器传递参数，也可以破坏这些约定。

本书中的子程序要求所有参数都存放在16位的寄存器或内存单元中。在适当的条件下，参数也可以是立即数。另外一个限制是，传送给子程序的参数个数不能大于4。寄存器的使用顺序如下：

AX	第一个参数
DX	第二个参数
BX	第三个参数
CX	第四个参数

返回值用下列寄存器存放：

AL	8位值
AX	16位值
DX：AX	32位值

2.3 文件DEFINES.MAC

在这个文件中，我们给出很多定义，它们可以加快编写程序的速度。

```
;.....................................................
;
; File Name ; DEFINES.MAC
;
;.....................................................
;
; MOVE_FILE_POINTER equates
;

SEEK_SET equ    0
SEEK_CUR equ    1
SEEK_END equ    2


;.....................................................
; Equates ; These simplified equates
          max be used along with the
;         assembler assigned register
;         names

SRCE_PTR equ    SI
DEST_PTR equ    DI
```

```
DBASE_PTR equ    BX
RET_VAL   equ    AX

;.....
; Disk IO equates
;

O_RDONLY equ    0
O_WRONLY equ    1
O_RDWR   equ    2
F_NORMAL equ    0
F_HIDDEN equ    2
F_SYS    equ    4
F_HIDSYS equ    6

;.....;
; DRIVE_STAT Drive status Structure ;
;.....;

DFREE   STRUC
    clust_avail   DW    ?       ; available clusters
    total_clust   DW    ?       ; total clusters
    byte_per_sec  DW    ?       ; bytes per sector
    sec_per_clust DW    ?       ; sectors per cluster
    free_bytes    DW    ? ,?    ; Free bytes on disk
DFREE   ENDS

;.....;
; Defines for MAKEATTR          ;
;.....;

ON_INTEN        equ    8
OFF_INTEN       equ    0
ON_INTENSITY    equ    8
OFF_INTENSITY   equ    0
ON_BLINK        equ    128
OFF_BLINK       equ    0

BLACK           equ    0
BLUE            equ    1
GREEN           equ    2
CYAN            equ    3
RED             equ    4
MAGENTA         equ    5
BROWN           equ    6
WHITE           equ    7
NORMAL          equ    7
REVERSE         equ    112

;.....
; Equates for window borders
;
S_S_S_S     equ    0
S_S_D_D     equ    1
D_D_S_S     equ    2
D_D_D_D     equ    3
```

```
;.....
; KeYs
;

SPACE_BAR       equ 3920h
E_CR            equ 1C0Dh
ESCAPE          equ 011Bh
PGDN            equ 5100h
PGUP            equ 4900h
PERIOD          equ 342Eh
TAB             equ 0F09h
RT_SQUARE       equ 1B5Dh
LT_SQUARE       equ 1A5Bh

RT_BRACKET      equ 1B7Dh
LT_BRACKET      equ 1A7Bh
CNTL_HOME       equ 7700h
CNTL_END        equ 7500h
HOME            equ 4700h
END_KEY         equ 4F00h
s_BS            equ 0008h
BS              equ 0E08h
BACKSPACE       equ 0E08h
s_CR            equ 00Dh
CR              equ 1C0Dh
ENTER_KEY       equ 1C0Dh
UP_ARROW        equ 4800h
RIGHT_ARROW     equ 4D00h
LEFT_ARROW      equ 4B00h
DOWN_ARROW      equ 5000h

;
; function keys
;

F1      equ 3B00h
F2      equ 3C00h
F3      equ 3D00h
F4      equ 3E00h
F5      equ 3F00h
F6      equ 4000h
F7      equ 4100h
F8      equ 4200h
F9      equ 4300h
F10     equ 4400h

;
; shift+key
;

SHIFT_TAB       equ 0F00h
SHIFT_HOME      equ 4747h
SHIFT_END       equ 4F31h
SHIFT_INSERT    equ 5230h
SHIFT_DELETE    equ 532Eh
SHFT_INSFRT     equ 5230h
SHFT_F1         equ 5400h
```