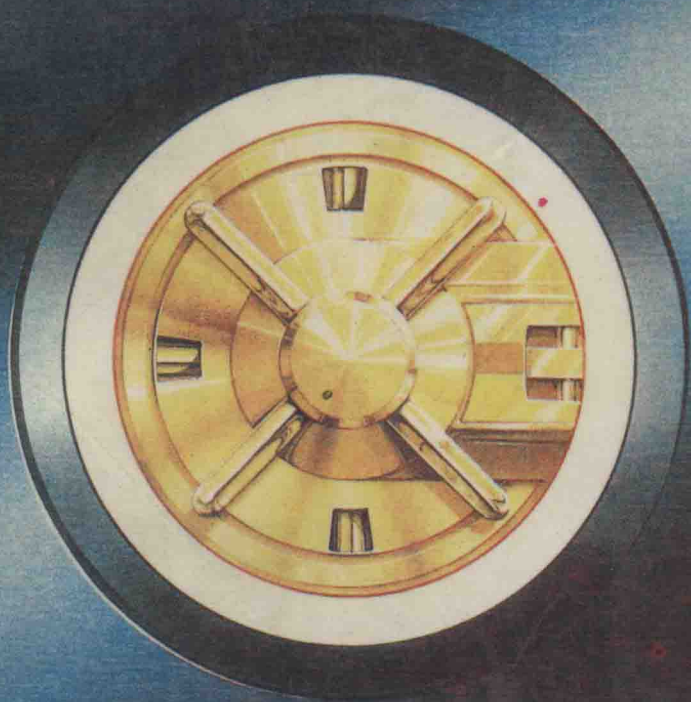


MS—DOS高级开发指南

—编程人员必备的高级技术参考书



中国科学院希望高级电脑技术公司

MS-DOS 高级开发指南

虞育新 李竹华

徐拥军 覃社庭

中国科学院希望高级电脑技术公司

一九九一年三月

前 言

《MS-DOS 高级开发指南》提供了在 MS-DOS 操作系统下进行程序设计的大量的强有力技术，并对 MS-DOS 操作系统本身进行了相当深入的剖析。本书的内容与 MS-DOS 4.0、MASM 5.1 和 Microsoft C 5.1 兼容，书中给出了为 MS-DOS 操作系统最新版本编写应用程序所需的所有细节。

本书包含以下各方面的内容：

- MS-DOS 4.0 的兼容性。包括所有的磁盘和文件格式、文件分配表 (FAT)，以及管理超过 32 兆硬盘分区的方法。
- 进行硬件控制的新内容，包括对串行端口的编程等。
- 关于设计内存驻留程序 (TSR) 详细、完整的说明。
- 使用未公布的功能来处理中断、实现功能和处理错误代码。
- 通过 MASM 5.0 程序设计实例以及各种表格、程序、附录和快速参考来说明内存管理、可安装的设备驱动程序、磁盘布局及文件恢复、实时程序设计和结构化程序设计等内容。

本书是 MS-DOS 编程人员人手必备的高级技术参考书，它将会彻底更新你的编程观念，使你的程序设计技术进入一个崭新的阶段。

感谢中科院希望公司资料部秦人华、杨淑欣在本书出版过程中所给与的帮助。

目 录

第一章 结构化程序设计 1: 结构化编程工具	(1)
简化语句的必要性	(1)
关于宏	(2)
LOCAL 标号	(4)
宏列表命令	(8)
Macro 库	(8)
宏重复语句—REPT	(9)
另外的宏重复语句—IRP 和 IRPC	(11)
关于宏的小结	(12)
条件汇编	(12)
关系操作符	(17)
条件汇编小结	(18)
条件汇编和宏	(18)
判断操作数类型	(19)
阶段错误和其它的 MASM 特点	(20)
串匹配——一个实例	(21)
分析宏参数	(23)
关于 MASM 中条件汇编和宏的警告	(27)
汇编语言中的结构控制语句	(28)
结构化控制宏如何工作	(36)
策略和警告	(37)
伪 case 宏	(40)
数据宏	(41)
代码宏	(46)
宏与子程序的比较	(47)
条件宏	(47)
宏嵌套	(48)
关于宏的更多特征	(49)
调用子程序的宏	(50)
使用 STRUC 语句	(52)
多个结构定位数据	(53)
结构作为子程序参数	(55)
小结	(56)
第二章 结构化程序设计 2: 模块化程序的设计与实现	(57)
模块化程序设计原理	(57)
设计选择项	(57)
设计功能上独立的单元	(58)

设计成具有最少的传递参数	(58)
设计成具有最少的调用数	(58)
模块化的规则	(59)
在汇编语言中实现模块化程序	(59)
参数、自变量、变量和常数定义	(60)
参数和模块	(60)
参数传递选项	(61)
使用值或地址传送参数	(67)
函数与子程序比较	(68)
例外报告	(69)
编码类型	(69)
程序代码定位	(70)
程序代码类型	(71)
代码定位小结	(78)
与高级语言的接口	(79)
Microsoft C 的调用约定	(79)
Microsoft Pascal 调用约定	(80)
Microsoft BASIC 和 FORTRAN 的调用约定	(81)
Microsoft 的段模式	(82)
在内存中分配和使用局部存贮	(82)
MS-DOS 内存管理介绍	(84)
保护数据和控制数据的范围	(86)
保护栈的完整性	(88)
小结	(88)
 第三章 程序和内存管理	(89)
MS-DOS 内存	(89)
MS-DOS 物理内存映象	(89)
扩展的 (Expanded) 和扩充的 (Extended) 内存	(89)
利用 MS-DOS 内存	(90)
MS-DOS 内存链	(92)
程序环境块	(99)
MS-DOS 进程	(100)
MS-DOS 进程上下文	(101)
程序段前缀 (PSP)	(101)
MS-DOS 进程文件 .EXE 与 .COM 比较	(109)
覆盖	(114)
内存驻留程序	(114)
定义一个运行时库	(114)
从命令行装入内存驻留例程	(115)
通过 int 存取内存驻留例程	(117)
判断一个内存驻留程序是否安装	(123)

移去内存驻留例程	(124)
功能 4Bh—装入并执行程序	(124)
通过 MS-DOS (代码 4Bh, AL = 0) 装入并执行程序	(129)
子程序的继承性和控制	(130)
使用功能 4Bh 执行 MS-DOS 命令	(130)
一个重要警告	(131)
通过 MS-DOS (代码 4Bh, AL = 3) 装入程序覆盖	(131)
从父程序存取程序覆盖	(132)
装入内存驻留程序	(134)
一种特殊情况: 部分时间运行时库	(134)
上下文切换和栈切换	(135)
对栈切换的附加考虑	(137)
内存驻留附注	(138)
ROM BIOS 与可安装 BIOS	(138)
中断与轮询系统比较	(138)
插入中断向量	(139)
REMOVE——一个完整的程序例子	(142)
小结	(149)
 第四章 终止并驻留程序	(150)
概述	(150)
PC 机硬件	(151)
硬件中断	(151)
软件中断	(152)
定时器中断	(152)
键盘	(152)
显示器 (硬件部分)	(153)
捕获中断	(156)
设置热键	(156)
捕获 Int 1ch 的选择	(161)
显示器控制	(162)
与 DOS 打交道	(164)
DOS I/O 数据结构	(164)
BIOS 调度器, Int 21h	(170)
字符 I/O 程序	(171)
DOS 全局变量	(171)
间断处理	(172)
致命错误处理	(172)
加载程序	(173)
程序终止	(174)
TSR 的加载初始化	(174)
查验 DOS 版本	(175)

TSR 程序驻留副本的定位	(175)
记录 PSP 地址	(179)
记录关键部分 (INDOS) 和致命错误地址	(179)
捕获中断矢量	(180)
检测显示器类型	(181)
释放环境	(182)
程序终止	(183)
重新激活及 DOS 的结构和服务	(184)
确定能否安全地重新激活	(184)
堆栈切换存贮寄存器内容	(185)
中断和致命错误陷阱	(186)
处理 DOS 全局变量	(187)
使用 Int 28h 的后台处理	(188)
从存贮器中删除 TSR 程序	(192)
小结	(193)
 第五章 实时编程	(194)
实时系统概述	(194)
什么是实时系统	(194)
实时系统的特性	(195)
实时系统的基本类型	(195)
典型定时要求和实时解决方法	(197)
MS-DOS 的实时应用	(199)
MS-DOS 有多快?	(200)
几种数据传输方式的比较	(207)
编写快速程序的技术	(207)
使用 MS-DOS 的实时系统设计	(209)
实例——一个简单家庭控制系统	(211)
查询系统	(213)
带中断的主循环	(214)
循环制表器	(215)
决定设计方法	(217)
MS-DOS 中的多任务	(218)
IBM-PCAT 中的多任务设置	(218)
小结	(219)
 第六章 可安装的设备驱动程序(一)	(220)
为什么使用设备驱动程序?	(220)
何时使用设备驱动程序	(221)
不可再入的 MS-DOS 带来的限制	(221)
安装设备驱动程序	(222)
CONFIG.SYS 文件	(223)

利用 ASSIGN 替代磁盘设备驱动程序	(226)
设备驱动程序的类型	(227)
在 MS-DOS 中存取设备驱动程序	(227)
CP/M 式字符设备 I/O	(228)
利用 FCB (文件控制块) 存取设备	(228)
利用文件把柄存取设备	(228)
功能 44h—针对设备的 I/O 控制	(228)
经由 IOCTL 命令进行配置	(231)
利用中断 25h 和 26h 进行的直接磁盘存取	(231)
Verify 开关	(232)
I/O 摘要	(232)
 第七章 可安装的设备驱动程序(二)	(233)
编写设备驱动程序	(233)
设备头	(234)
策略例程	(238)
中断例程	(239)
驱动程序命令	(243)
创建可装入的设备驱动程序文件	(254)
调试设备驱动程序	(255)
在系统中显示设备驱动程序	(256)
无所不在的虚拟盘	(262)
摘要	(279)
 第八章 串行端口程序设计(一)	(280)
异步串行通信的基本知识	(280)
校验和错误捕获	(282)
同 RS-232C 标准通信	(282)
用 XON/XOFF 进行流控制	(283)
从程序设计的角度看待串行端口	(284)
中断驱动的串行 I/O	(286)
来自串行适配器的中断	(286)
8259A 可编程中断控制器	(288)
8259A 程序设计	(289)
 第九章 串行端口程序设计(二)	(291)
利用 MS-DOS 工具对串行端口编程	(291)
驱动程序、内存驻留程序(TSR)或独立的程序	(291)
利用 BIOS 进行串行通信	(291)
为中断驱动串行 I/O 操作进行设置工作	(294)
处理串行端口产生的中断	(295)
中断处理程序中的队列	(297)

在关张前先清除	(297)
示例程序	(298)
摘要	(308)
第十章 Intel NPX 编程	(309)
NPX 的编程人员观点	(309)
NPX 中的数据寄存器	(309)
NPX 中的浮点实数表达	(310)
NPX 中使用的其它数据格式	(312)
数据类型总结	(314)
NPX 指令集	(315)
FWAIT 前缀	(315)
NPX 的寻址方式	(318)
FINIT 和 FFREE 指令	(319)
控制 NPX	(319)
使用 MS-DOS 工具和 NPX	(323)
使用 MASM 和 NPX	(323)
MASM 的 NPX 开关——/r 和 /e	(324)
MASM 中的 NPX 数据类型	(324)
用 MASM 的 NPX 编程例子	(326)
FWAIT 和 FINIT 指令	(326)
DUMP87 程序	(327)
NPX 用于二进制和十进制转换	(338)
总结	(347)
第十一章 磁盘布局 and 文件恢复	(348)
文件恢复原理	(349)
5 ¹ / ₄ 英寸 40 道单面软盘的布局	(349)
5 ¹ / ₄ 英寸 40 道双面软盘布局	(350)
5 ¹ / ₄ 英寸 80 道双面软盘布局	(351)
启动扇区	(352)
目录扇区	(373)
文件分配表 (FAT) 扇区	(375)
译码 FAT 项	(378)
簇转换成逻辑扇区	(381)
文件恢复过程综述	(381)
用 CHKDSK 和 RECOVER 恢复被毁文件	(381)
恢复被删文件	(382)
基础知识	(382)
恢复被删文件硬方法	(384)
使用 RESCUE 程序	(385)
使用 Norton 实用程序	(398)

使用 Ultra 实用程序	(399)
总结	(399)
 第十二章 内存丢失数据的恢复	(400)
从字处理 / 正文编辑失败中恢复	(400)
总结	(403)
 第十三章 MS-DOS 版本间的差异	(404)
概要	(404)
一般兼容建议	(404)
高级语言考虑	(407)
MS-DOS 中断	(408)
功能调用	(409)
执行功能调用标准方法	(409)
以兼容方式执行功能调用	(409)
再一个方法(只适用 MS-DOS 2.00 和更高版本)	(410)
不同版本支持的功能	(410)
程序终止组	(416)
标准字符设备输出 / 输出组(01h-0Ch)	(417)
标准文件管理组(0Dh-24h, 27h-29h)	(417)
标准非设备功能(25h, 26h, 2Ah-2Eh)	(417)
扩充(一般)功能组(2Fh-38h, 4Ch-4Fh, 54h-5Th, 59h-5Fh, 62h)	(417)
目录组(39h-3Bh, 47h)	(418)
内存 / 进程管理组(48h-4Bh)	(418)
错误代码	(418)
关键或硬错误代码(通过 Int 24h)	(418)
功能调用错误返回代码(只在 MS-DOS 2.0 和更高版本)	(419)
功能调用扩充错误信息(只在 MS-DOS 3.0 和更高版本)	(422)
磁盘格式	(424)
文件操作	(426)
使用文件控制块(FCB)	(426)
MS-DOS 文件处理	(426)
MS-DOS 和 IBM PC 及 IBM PS / 2	(427)
相同点	(427)
不同点	(428)
与其它操作系统兼容	(429)
CP / M-80	(429)
CP / M-86 和并行 CP / M-86	(430)
并行 PC-DOS 和并行 DOS-286	(430)
XENIX 和 UNIX	(431)
概要	(431)

附录 A 开发工具	(432)
使用批文件自动化汇编过程	(432)
为 MASM 5 以前版本使用批文件	(432)
为 MASM 5 和更高版本使用批文件	(434)
使用 Microsoft 的 MAKE 工具	(437)
使用模板建立.COM 和.EXE 程序	(438)
使用库程序	(457)
附录 B 未公布的 MS-DOS 中断和功能	(474)
未公布的 MS-DOS 中断	(474)
中断 28h(40):DOS 安全中断	(474)
中断 29h(41):控制台设备输出	(474)
中断 2Ah(42)至 2Dh(45):MS-DOS 内部程序	(475)
中断 2Eh(46):COMMAND 处理器后门	(475)
中断 30h(48)至 FFh(255)	(475)
未公布的中断 21h(33)功能调用	(475)
功能 1Fh(31):为缺省设备定位磁盘块信息	(475)
功能 32h(50):为指定设备定位磁盘块信息	(476)
功能 34h(52):获得 MS-DOS 忙标记	(476)
功能 37h(55):获得 / 设置转换字符	(477)
功能 50h(80):设置 PSP 段	(477)
功能 51h(81):获得 PSP 段	(477)
功能 52h(82):获得表的 MS-DOS 表地址	(477)
功能 53h(83):转换 BIOS 参数块(BPB)为磁盘块	(478)
功能 55h(85):建立 PSP 块	(478)
功能 58h(88):获得 / 设置内存分配策略	(478)
功能 60h(96):分解路径串或完全合格的路径串	(478)
功能 63h(99):获得引导字节表	(479)
附录 C ASCII 交叉引用和数值转化	(480)
不可打印的 ASCII 字符定义	(482)
十六进制到十进制转化	(484)
十进制到十六进制转化	(484)

第一章 结构化程序设计 1:

结构化编程工具

每当人们讨论结构化程序设计的奥妙时，常注重于 IF-THEN-ELSE 语句这样一个语言结构子集。Pascal 或 C 语言的专家可能会通过与汇编语言的比较来说明高级程序设计语言结构化程序设计的优点，而 GOTO 语句的使用又可能成为阐述的焦点。但所有的这些讨论都未能阐明所有的细节，因为这些讨论本质上都只集中于结构化编程。稍后读者将明白，在任何语言中结构化编程都是可能的，甚至一些汇编语言也支持所有的高级控制结构，其中之一是 Microsoft 公司的 MS-DOS 宏汇编，即 MASM。

简化语句的必要性

在说明汇编语言中的高级控制结构之前，先看一下高级语言的一些优点。最基本的，在高级语言中能完成的任何事情也可在汇编中实现。那么，使用高级语言的优点究竟是什么？简洁！能够用程序员或读者易于理解的方式表达一个程序设计意图。考虑到每种汇编语言的语句或多或少地依赖于机器指令，另一方面，一个高级语句可能扩展至数十甚至数百条机器代码指令（如果有人怀疑数百条，只须查看一个用嵌套参数计算的 FORTRAN 子例程）。

图 1-1 示出了在 FORTRAN 和 8086 汇编语言中的相同意义代码片段。该片段对一个给定的 NUM 计算 1...NUM 的总和。无疑汇编语言例程无论在生成的目标代码数量还是在执行时间方面都更为优化，但不管如何观察，用 FORTRAN 书写要比用汇编书写容易得多。书写汇编语言例程时，要作出更多的决定，由于汇编中涉及的这些额外工作，更易产生编程错误。可以承认 FORTRAN 例程能完好地执行，但仍不免怀疑汇编例程的运行。为什么会怀疑呢？因为 FORTRAN 例程的每一行构成一个完整的思想，而汇编语言例程则要有许多行才能代表同一思想。

FORTRAN	Assembly Language
SUM = 0	mov sum,0
DO 100 I = 1, NUM	mov ax,1
100 SUM = SUM + I	loop1: cmp ax,num
	jg loop1_end
	add sum,ax
	inc ax
	jmp loop1
	loop1_end:

图 1-1 FORTRAN 与汇编语言比较

简单说来,使用高级结构导致编程简单和代码更为可靠。这些结构使得编程难度下降,允许程序员注重于程序的逻辑,假定实际实现是正确的。这样程序员就会信任自己的工作,支持这种信心的工具将造就更好的程序员。

关于宏

若能为一些常用语句建立一个简写则将极大地加速汇编语言的编程。MASM 为此提供了 macro 机制,macro 是“超级指令”,为汇编语言的编程节省了大量枯燥无味的重复工作。使用 macro 时,程序员定义汇编语句块,然后用单个的引用使 MASM 在汇编程序中包含该重复块。本章将介绍一些宏,并逐步地引导读者书写自己的工具。这将使你能将汇编语言的执行速度与高级语言的能力结合起来。

用于建立并使用一个宏的两个步骤如下:

步骤 1: 定义宏

```
;; Define      "Function Request" as @DosCall
@DosCall      MACRO
                int 21h      ; Call MS-DOS to perform function
                ENDM
```

步骤 2: 使用宏

@DosCall ←宏调用

程序中显示的内容:

```
1      @DosCall      ←宏调用
      int 21h      ; Call MS-DOS to perform function
```

该程序被汇编后,语句 DosCall 被语句 int 21h 替换,包括注释。列表文件包含 DosCall 行为一个引用,但目标文件只包含 int 21h 的代码。该操作称为宏替换或宏扩展。

注意上例中汇编器在列表文件中插入一个符号指出该扩展的宏代码。在 MASM 的第四版及更高版中,在第一个宏扩展前放一个 1,而在第二级宏扩展处放一个 2,如此等等。在 MASM 第三版及以前版中,所有的宏扩展行,不管其级,一律标以加号 (+) 字符。

经过汇编器处理后,宏调用被该宏代表的代码替换。宏不产生调用宏代码的 CALL 指令,尽管有时宏引用采用这种方法。

类似于任何的编程,宏必须遵守严格的格式。定义宏的格式为:

```
mname MACRO argument_list
```

 ←宏代码体

```
ENDM
```

其中, mname 是宏的名, argument_list 是一个由逗号分隔的参数表,当不包含参

数时该列表可以为空（例如在例中的@DosCall）。

这只是简单的说明，如果仅此而已，则真是太令人失望了，所幸宏可以使用参数。下一个例子带有参数：

```
;; Define      "Print Character" as PRINT__CHR
@ PrintChr    MACRO Char
               mov ah, 05
               mov dl, &char
               @ DosCall
               ENDM
```

现在使用该宏：

```
@ PrintChr 'A'    ←宏调用
```

则列表文件中出现：

```
@ PrintChr 'A'    ←宏调用
```

```
1 mov ah, 05
```

```
1 mov dl, 'A'
```

```
2 int 21h ; Call MS-DOS to perform function
```

宏调用中&char 被 'A' 替换。（我们称使用宏为调用，只要读者记住其实并不执行 CALL 指令）。在行首出现的数字是 MASM 用以告诉程序员这些代码是宏扩展结果的方法。还应注意宏 @PrintChr 包含一个对先前定义的宏 @DosCall 的一个调用，它被扩展为 @DosCall 包含的 int 21h 语句。MASM 继续计数宏调用嵌套的级数，直到 MASM 的符号表存储区域溢出。嵌套指一个宏再调用宏，如此继续。

宏 @PrintChr 中的 Char 称为一个哑参数，每当哑参数 Char 出现于该宏中，它就被用于调用该宏的值替代。在 @PrintChr 例子中，Char 替换意为宏中任何出现 Char 的地方都用字符 A 替代。

注意在宏中作为哑参数的名不能再作为变量使用。因此若将 AX 作为哑参数，就不能用 AX 表示 AX 寄存器了。

关于命名哑参数的限制同样也适用于命名宏本身。假如用名 add 定义了一个宏，将会发现程序中所有引用 ADD 操作的地方全部被生成宏扩展 add。如果需要的话，你甚至可以重定义 MASM 的语句。因此避免造成冲突是很重要的。

在宏 @PrintChr 中 Char 前的 & 用于将 Char 的值加入到串 mov dl。对于传送哑参数来说 & 不是必须的，仅告诉 MASM Char 是一个哑参数，而并不是串 mov dl, char 的一部分。当哑参数包含在大的串中时 & 操作符尤其重要，比如下一个例子。

The Macro Definition

@Example MACRO arg

```
mov dl,arg
```

```
mov dl,&arg
```

```
mov dl,argZ
```

```
mov dl,&argZ
```

```
mov dl,arg&Z
```

```
mov dl,Xarg
```

The Macro Expansion

@Example Y

```
1 mov dl,Y    ←correct
```

```
1 mov dl,Y    ←correct
```

```
1 mov dl,argZ
```

```
1 mov dl,argZ
```

```
1 mov dl,YZ    ←correct
```

```
1 mov dl,Xarg
```

mov dl,X&a g	1	mov dl,XY	←correct
mov dl,XargZ	1	mov dl,XargZ	
mov dl,X&argZ	1	mov dl,XargZ	
mov dl,Xarg&Z	1	mov dl,XargZ	
mov dl,X&arg&Z	1	mov dl,XYZ	←correct
ENDM			

严格地说，在宏@PrintChr中不需要&。MASM能够检查出Char是一个哑参数，因为Char在逗号后面。但即使在并不需要时也用&是一个好习惯，因为当你读宏时能提醒这是一个哑参数，且能清楚地告诉MASM。

LOCAL 标号

迄今为止，我们使用的宏局限于生成简单的汇编指令。假定要设计这样一个宏，从两个数中选择较小的一个，并将其值放入到另一地址。该宏可以是这样的：

```
min    MACRO result, first, second
        mov &result, &first
        cmp &first, &second
        jl order_ok
        mov &result, &second
```

```
order_ok:
```

```
ENDM
```

执行min时，产生正确的代码，但有一个问题：即使该宏很好地执行了，也只能使用一次。因为标号order_ok在程序中只能定义一次，若该宏在两个地方使用时MASM就警告Symbol is multidefined。

可以在该宏中作小小的修改使得可以指定一个标号参数：

```
min    MACRO result, first, second, order_ok
        mov &result, &first
        cmp &first, &second
        jl &order_ok
        mov &result, &second
```

```
order_ok&:
```

```
ENDM
```

执行该新min时，如下面例中所示，我们可以指定用作跳转标号的名。现在min可以在需要时重新使用，但仍需每次都指定一个新的跳转标号名，然而由于标号局部于min，所以实际的标号名并不重要。

```
min ax, bx, cx, jmp1
1 min ax, bx
1 cmp bx, cx
1 jl jmp1
1 mov ax, cx
```

1 jmp1:

有一种更好的方法能用于在每次调用 min 时建立一个新的名, MASM 用 LOCAL 语句来做到这一点。每当 MASM 遇到 LOCAL, 自动产生一个唯一的标号。换句话说, 就好像是 LOCAL 参数包含于 MACRO 的参数表中, 但 MASM 自己填以实际参数。注意, LOCAL 语句必须紧跟在 MACRO 定义行的后面。包含了 LOCAL 语句的 min 如下:

```
min    MACRO result, first, second
        LOCAL order__ok
        mov &result, &first
        cmp &first, &second
        jl order__ok
        mov &result, &second
order__ok:
        ENDM
```

此时执行 min 后, 扩展后的列表如下面例中所示。order__ok 的值被替换为??0000。每次调用时, order__ok 被 MASM 产生的一个新值替代。

```
        min ax, bx, cx    ; first call
1  mov ax, bx
1  cmp bx, cx
1  jl ?? 0000
1  mov ax, cx
1 ?? 0000:
        min ax, bx, cx    ; second call
1  mov ax, bx
1  cmp bx, cx
1  jl ?? 0001
1  mov ax, cx
1 ?? 0001:
```

当然, 如果使用以??开头的标号时仍有可能会发生标号冲突, 但若程序员避免使用以??开头的标号, 则可以任意多次地调用宏 min。

LOCAL 标号并不仅用于跳转地址, 也可用于数据, 如后面的宏中所述。在这种情况下, 这些宏用于将正文串插入到数据段, 同时在代码段中建立对该串的引用。通过比较程序 1-1 中的源代码和宏扩展, 可以清楚地看出宏的使用。

程序 1-1 中还包含了一些用于书写 .EXE 程序的有用宏。一旦你定义了这些宏, 就不用再担心正确地获得 .EXE 程序的格式。

读者可以直接将该程序键入, 然后汇编并执行, 这时显示 Hello World!。该程序本身结果并不多, 但若将它保存在一个 include 文件中时, 将使得书写 .EXE 程序更为容易。让我们看一下程序 1-2 中的已扩展程序列表。

程序 1-1 Hello World 程序

```

; *****
;  MACRO  DEFINITION  SECTION
;  *****
;
@DosCall MACRO
    int      21h          ; call MS-DOS function
    ENDM

;
@InitStk MACRO            ; define stack size
stk_seg SEGMENT stack
    DB      32 dup ('stack ')
stk_seg ENDS
    ENDM

;
@InitPrg MACRO  segment   ; initialize data segment
    ASSUME ds:segment
start:
    mov     ax,segment    ; main entry point
    mov     ds,ax         ; set up data segment
    mov     es,ax         ; set up extra segment
    ENDM

;
@Finis  MACRO
    mov     ax,4C00h      ; terminate process
    @DosCall
    ENDM

;
@DisStr MACRO  string     ; display string from memory
    mov     dx,offset string
    mov     ah,09h
    @DosCall
    ENDM

;
@TypeStr MACRO  string    ; define and display a string
    LOCAL  saddr          ; set up a local label
    cod_seg ENDS          ; stop code segment
    dat_seg SEGMENT       ; change to data segment
    saddr DB      string,'$' ; define string in data segment
    dat_seg ENDS          ; stop data segment
    cod_seg SEGMENT       ; return to code segment
    @DisStr saddr         ; display string
    ENDM

```