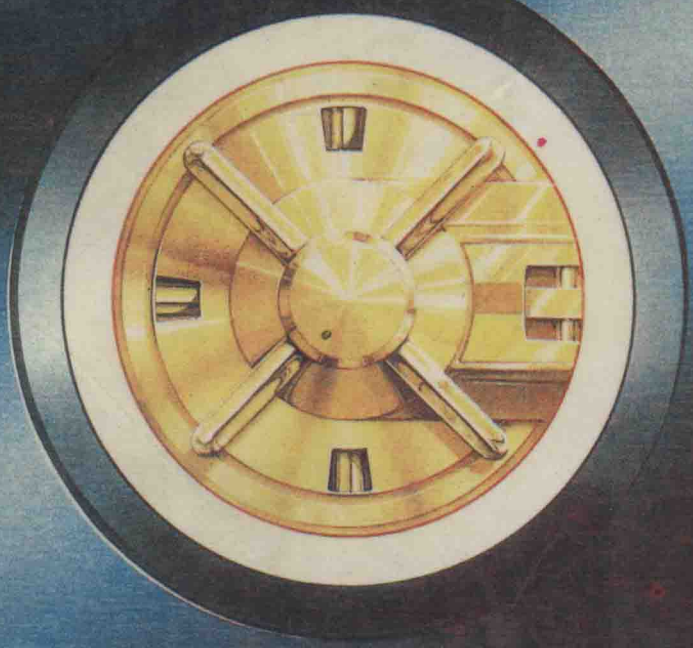
MS—DOS高级开发指南

—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

每当人们讨论结构化程序设计的奥妙时，常注重于 IF-THEN-ELSE 语句这样的一个语言结构子集。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呢？简洁！能够用编程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:

```
            @DosCall     ←宏调用
1           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 的符号表存贮区域溢出．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          The Macro Expansion
@Example MACRO arg            @Example Y
   mov  dl,arg                    1   mov  dl,Y       ←correct
   mov  dl,&arg                   1   mov  dl,Y       ←correct
   mov  dl,argZ                   1   mov  dl,argZ
   mov  dl,&argZ                  1   mov  dl,argZ
   mov  dl,arg&Z                  1   mov  dl,YZ      ←correct
   mov  dl,Xarg                   1   mov  dl,Xarg
```

```
mov  dl,X&a g                1   mov  dl,XY      ←correct
mov  dl,XargZ                1   mov  dl,XargZ
mov  dl,X&a gZ               1   mov  dl,XargZ
mov  dl,Xa g&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，就好象是 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 Hellow 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

```
; ************************************************************
;  M A C R O   D E F I N I T I O N   S E C T I O N
; ************************************************************
;
@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:                          ; main entry point
        mov     ax,segment
        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
```