附录 A　汇编编译器的安装和使用

本章要点：

- 安装随书附带的 CD-ROM
- 编译和链接 32 位保护模式程序
- 编译和链接 16 位实地址模式程序

A.1　安装随书附带的 CD-ROM

读者可以运行光盘根目录下的 setup 文件来安装随书附带的 CD-ROM。随书附带的 CD-ROM 上包括下面的软件：

- Microsoft 的宏汇编编译器，版本是 6.15。默认安装在 C:\Masm615 目录中。
- Microsoft 的 16 位和 32 位的链接器（link.exe 和 link32.exe）。默认安装在 C:\Masm615 目录中。
- extPad 编辑器的评估版本，由 Helios 软件公司出品。要安装 TextPad 的话，可以运行\TextPad 目录中的 TextPad4.exe 程序。
- 本书的所有例子程序。默认情况下安装在 Examples 子目录中。
- 第 17 章。本书的补充章节，以 Adobe Acrobat 文件格式存储（原英文版）。
- 用于编译、链接和调试的批处理文件。默认情况下和 MASM 安装在同一目录下。下面是文件列表：

make32.bat	编译和链接 32 位保护模式程序
make16.bat	编译和链接 16 实地址模式程序
runCV.bat	运行 Microsoft 的 16 位调试器 CodeView
runQH.bat	运行 Microsoft 的 QuikHelp 实用工具，可显示编译器，链接器，CodeView 和其他使用工具的补充信息

- 例子程序中使用的链接库。默认安装在 LIB 子目录中，包括 kernel32.lib，user32.lib，irvine32.lib 和 irvine16.lib 等。
- 例子程序使用的包含文件。默认安装在 INCLUDE 子目录中，包括 irvine32.inc，irvine16.inc，SmallWin.inc，GraphWin.inc，macro.inc 和 win.inc 等。
- Microsoft 汇编编译器附带的各种实用程序，例如 cref.exe，cvpack.exe 和 nmake.exe，它们与 MASM 在同一目录中。

A.2　编译和链接 32 位保护模式程序

可以使用 make32.bat 批处理文件编译和链接保护模式汇编语言程序。格式如下（大小写不敏感）：

```
make32 progname
```

progname 是汇编语言源代码文件的名字，不包括扩展名。例如，下面的命令将编译和链接 AddSub.asm 源代码文件：

```
make32 AddSub
```

假设没有发生错误，当前目录下将创建如下文件：

AddSub.obj　目标文件
AddSub.lst　列表文件
AddSub.exe　可执行文件

在运行 make32 命令之前，在源代码文件的同一目录中必须有 make32.bat 文件的一份拷贝。读者可从汇编编译器的安装目录中将 make32.bat 文件拷贝过来。

有两种方法可以执行 make32 命令：

1. 进入源代码文件（如 AddSub.asm）所在的目录，在命令提示符下键入如下的命令：

   ```
   make32 AddSub
   ```

2. 或者可以使用 Helios TextPad 之类的编辑器编辑和编译源代码文件。TextPad 的评估版本在随书附带的 CD-ROM 中提供，关于设置 TextPad 的详细信息请参阅我们的 Web 站点。

A.2.1 调试保护模式程序

Microsoft MASM 软件包并没有包含一个 32 位的调试器，但是读者可以使用 Mcirosoft Visual C++ 6.0 附带的 Microsoft Developer Studio Debugger 调试器进行调试，或者可以从 Microsoft 的 Web 站点上下载 Windows Debugger。另外，在我们的站点上查找：Debugging 32-bit Programs 也可以获得更多的信息。

A.2.2 mak32.bat 文件

批处理文件是一个文本文件，其中包含的命令执行起来就像在 MS-DOS 命令提示符下直接键入一样。批处理文件必须以 BAT 作为扩展名，它可以在命令提示符下或其他程序中执行。

表 A.1 对 make32.bat 中包含的命令进行了解释。左边一列中的两条命令（REM 和 LINK32）折行显示是由于表格的长度限制。在批处理文件中，它们只占用一行。

表 A.1　make32.bat 批处理文件的说明

批处理命令	说明
REM Make32.bat, for assembling and linking Protected mode programs	任何以REM开头的行都是注释行，用于对批处理文件加以说明。REM 行不执行
PATH C:\Masm615	将路径设置为 C:\Masm615，这使操作系统能够定位 ML 和 LINK32 程序。
SET INCLUDE=C:\Masm615\INCLUDE	设置 INCLUDE 环境变量。这使源代码文件中的 INCLUDE 伪指令可以定位指定的文件，例如我们使用的 Irvine32.inc 文件
SET LIB=C:\Masm615\LIB	设置 LIB 环境变量，以允许链接程序定位库文件，例如 Irvine32.lib

（续表）

批处理命令	说明
ML-Zi-c-Fl-coff %1.asm	调用 Microsoft 的汇编编译器（ML.EXE）
IF errorlevel 1 goto terminate	如果上面的命令产生了错误，则跳转到名为 terminate 的标号处（文件的最后一行）
LINK32 %1.obj Irvine32.lib kernel32.lib /SUBSYSTEM:CONSOLE /DEBUG	调用 Microsoft 的 32 位链接器，这里使用了两个库文件：Irvine32.lib 和 Kernel32.lib
IF errorLevel 1 goto terminate	如果上面的命令产生了错误，那么跳转到 terminate 标号
DIR %1.*	当源文件被编译链接后，列出编译器和链接器创建的文件
:terminate	该标号由前面的 GOTO 语句使用

你拷贝的 make32.bat 文件中的编译器和链接器的目录名可能不同，这是由安装目录不同造成的。例如如果编译器在 D:\Apps\Masm615 目录中，下面三条批处理命令就要相应改变：

```
SET PATH=D:\Apps\Masm615
SET INCLUDE=D:\Apps\Masm615\include
SET LIB=D:\Apps\Masm615\lib
```

A.3　编译和链接 16 位实地址模式程序

使用make16命令可以编译和链接16位实地址模式程序。例如，如果源代码的文件名为AddSub.asm，在 MS-DOS 提示符下可以运行下面的命令：

```
make16 AddSub
```

在运行 make16 命令之前，在源代码文件的同一目录中必须有 make16.bat 文件的一份拷贝。读者可从编译器的安装目录中拷贝 make16.bat 文件。

在 MS-DOS 提示符下键入如下命令可以运行 CodeView 调试器来对 AddSub 程序进行调试：

```
C:\Masm615\runCV AddSub
```

CodeView 将 AddSub.exe 装入内存，显示其源代码（来自 AddSub.asm）并允许你跟踪和调试程序的执行过程。使用 CodeView 之前请阅读本书 Web 站点上的 CodeView 使用教程。

16 位链接器只能识别少于 8 个字符（不包括扩展名）的文件名，而且程序路径中的目录名也不能超过 8 个字符长。

表 A.2 对 make16.bat 中包含的命令进行了解释。同样，左边一列中的两条命令（REM 和 LINK）折行显示是由于表格的长度限制，在批处理文件中它们只占一行。关于传递给汇编编译器和链接器的命令行参数的信息请参考附录 D。

表 A.2　make16.bat 批处理文件的说明

批处理命令	说明
REM Make16.bat, for assembling and linking Real-address mode programs	任何以 REM 开头的行都是注释行，用于对批处理文件加以说明。REM 行不执行
PATH C:\Masm615	将路径设置为 C:\Masm615，这使操作系统能够定位 ML 和 LINK 程序

（续表）

批处理命令	说　明
SET INCLUDE=C:\Masm615\INCLUDE	设置 INCLUDE 环境变量，这允许源代码文件中的 INCLUDE 伪指令可以定位指定的文件，例如我们使用的 Irvine16.inc 文件
SET LIB=C:\Masm615\LIB	设置 LIB 环境变量，以允许链接程序定位库文件，例如 Irvine16.lib
ML /nologo –c –Fl –Zi %1.asm	调用 Microsoft 的汇编编译器（ML.EXE）
IF errorlevel 1 goto terminate	如果上面的命令产生了错误，则跳转到名为 terminate 的标号处（文件的最后一行）
LINK /nologo /CODEVIEW %1, , NUL, Irvine16;	调用 Microsoft 的 16 位链接器，这里使用了库文件 Irvine16.lib
IF errorLevel 1 goto terminate	如果上面的命令产生了错误，那么跳转到 terminate 标号
DIR %1.*	当源文件被编译链接后，列出编译器和链接器创建的文件
:terminate	该标号由前面的 GOTO 语句使用

附录 B　Intel 指令集

本章要点：

- B.1 简介
- B.1.1 标志
- B.1.2 指令描述及格式
- B.2 指令集

B.1　简介

本附录是 Intel IA-32 系列处理器所有实地址模式指令的速查手册。

B.1.1　标志

每条指令的说明中都包含一系列用于描述指令如何影响 CPU 标志的方格，其中的每个标志用一个字母表示：

O	溢出标志	S	符号标志	P	奇偶标志
D	方向标志	Z	零标志	C	进位标志
I	中断标志	A	辅助进位标志		

在方格中，使用下面的符号来表示每条指令是以何种方式影响标志的：

1	设置标志
0	清除标志
?	标志值的改变无法预测
*	根据与该标志相联系的特定规则改变标志值

例如，下面的 CPU 标志图摘自某条指令的描述：

O	D	I	S	Z	A	P	C
?			?	?	*	?	*

从图中可见：溢出标志、符号标志、零标志和奇偶标志将改变为未知值，辅助进位标志和进位标志将根据与这些标志值相联系的规则进行修改，方向标志和中断标志不会改变。

B.1.2　指令描述及格式

当引用源操作数和目的操作数时，我们使用 Intel 80x86 指令的自然顺序，即其中第一个操作数是目的操作数，第二个操作数是源操作数。例如在 MOV 指令中，目的操作数将被赋以源操作数的一份拷贝：

```
MOV  目的，源
```

一条指令可能有多种格式。表 B.1 是在指令格式中使用的符号的列表。在单条指令描述中，使用符号（IA-32）表示一条指令或其变例只能用于 IA-32 系列处理器（Intel 386 以上），与之类似，符号（80286）表示至少要使用 80286 处理器。

寄存器符号如(E)CX,(E)SI,(E)DI,(E)SP,(E)BP 和(E)IP 等分别用于使用 32 位寄存器的 IA-32 处理器和使用 16 位寄存器的早期处理器。

表 B.1 指令格式中使用的符号

符号	描述
reg	下列 8 位、16 或 32 位通用寄存器中的一个：AH，AL，BH，BL，CH，CL，AX，BX，CX，DX，SI，DI，BP，SP，EAX，EBX，ECX，EDX，ESI，EDI，EBP 和 ESP
reg8,reg16,reg32	通用寄存器，以其包含的数据位的位数来标识
segreg	16 位段寄存器（CS，DS，ES，SS，FS，GS）
accum	AL，AX 或 EAX
mem	使用任何标准内存寻址方式的内存操作数
mem8, mem16, mem32	内存操作数，数字指明了操作数的位数
shortlabel	代码段内距当前位置-128 到+127 字节范围之内的地址
nearlabel	当前代码段内的位置，以标号标识
farlabel	外部代码段内的位置，以标号标识
imm	立即操作数
imm8, imm 16,imm32	立即操作数，数字指明了操作数的位数
instruction	一条 Intel 汇编语言指令

B.2 指令集

AAA — 加法后进行 ASCII 调整（ASCII Adjust After Addition）

O	D	I	S	Z	A	P	C
?			?	?	*	?	*

调整两个 ASCII 数字相加之后在 AL 中的结果。如果 AL 的低 4 位大于 9 或辅助进位标志等于 1，则 AL 加 6 并清除 AL 的高 4 位，同时 AH 加 1，设置进位标志和辅助进位标志；否则，直接清除 AL 的高 4 位，清除进位标志和辅助进位标志

指令格式：

```
AAA
```

AAD — 在除法前进行 ASCII 调整（ASCII Adjust Before Divesion）

O	D	I	S	Z	A	P	C
?			*	*	?	*	?

将 AH 和 AL 中未压缩的 BCD 数字转换成二进制数值存放在 AL 中，AH 清零，为 DIV 指令做好准备

指令格式：

```
AAD
```

AAM

乘法后进行 ASCII 调整(ASCII Adjust After Multiply)

O	D	I	S	Z	A	P	C
?			*	*	?	*	?

调整两个未压缩的 BCD 数字相乘之后 AX 中的结果，即将二进制值调整为非压缩的 ASCII 格式。调整方法是 AL 除以 0Ah，得到的商存放在 AH 中，余数存放在 AL 中。

指令格式：

```
AAM
```

AAS

减法后进行 ASCII 调整(ASCII Adjust After Subtraction)

O	D	I	S	Z	A	P	C
?			?	?	*	?	*

调整 ASCII 减法之后 AL 中得到的结果。如果 AL 的低 4 位大于 9 或辅助进位标志等于 1，AL 减 6，清除 AL 的高 4 位，AH 减 1，设置进位标志和辅助进位标志。否则，直接清除 AL 的高 4 位，清除进位标志和辅助进位标志

指令格式：

```
AAS
```

ADC

带进位加(Add Carry)

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

将源操作数、目的操作数和进位标志相加。操作数尺寸必须相同

指令格式：

```
ADC   reg,reg          ADC   reg,imm
ADC   mem,reg          ADC   mem,imm
ADC   reg,mem          ADC   accum,imm
```

ADD

加(Add)

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

源操作数与目的操作数相加，结果存储在目的操作数中。操作数尺寸必须相同

指令格式：

```
ADD   reg,reg          ADD   reg,imm
ADD   mem,reg          ADD   mem,imm
ADD   reg,mem          ADD   accum,imm
```

AND 逻辑与（Logical AND）

O	D	I	S	Z	A	P	C
*			*	*	?	*	0

目的操作数中的每个数据位与源操作数中的对应位进行与操作

指令格式：

```
AND   reg,reg          ADD   reg,imm
AND   mem,reg          ADD   mem,imm
AND   reg,mem          ADD   accum,imm
```

BOUND 检查数组边界（80286）（Check Array Bounds）

O	D	I	S	Z	A	P	C

检查一个有符号的指针值是否在数组边界之内。在 80286 处理器上，目的操作数可以是任何包含要检查的指针的 16 位寄存器，源操作数必须是 32 位内存操作数，其高字和低字分别包含数组的上边界指针值和下边界指针值。在 IA-32 上，目的操作数也可以是 32 位寄存器，源操作数可以是 64 位内存操作数。如果待检查的指针值在数组边界之外，则产生界常

指令格式：

```
BOUND   reg16,mem32          BOUND   reg32,mem64
```

BSF
BSR 位扫描（Bit Scan）（IA-32）

O	D	I	S	Z	A	P	C
				*			

扫描操作数并寻找第一个被设置的数据位。如果找到则清除零标志，目的操作数存放第一个被设置位的位号（索引）。如果没有找到被设置的数据位则 ZF = 1。BSF 指令按照从位 0 到最高位的顺序扫描。BSR 指令按从最高位到位 0 的顺序扫描

指令格式（以 BSF 为例，适用于 BSF 和 BSR，后面有些指令的描述方法类似，不再重复说明）：

```
BSF reg16,r/m16          BSF reg32,r/m32
```

BSWAP 字节交换（Byte Swap）（IA-32）

O	D	I	S	Z	A	P	C

反转 32 位目的寄存器中的字节顺序

指令格式：

```
BSWAP reg32
```

BT
BTC
BTR
BTS

位测试(Bit Tests)(IA-32)

O	D	I	S	Z	A	P	C
							*

将指定位(*n*)拷贝入进位标志中，目的操作数包含要操作的位号。BT 将源操作数的位 *n* 拷贝到进位标志中；BTC 将源操作数的位 *n* 拷贝到进位标志中并将位 *n* 变反；BTR 将源操作数的位 *n* 拷贝到进位标志中并将位 *n* 清除；BTS 将源操作数的位 *n* 拷贝到进位标志中并将位 *n* 置 1

指令格式：

```
BT r/m16,imm8          BT r/m16,r16
BT r/m32,imm8          BT r/m32,r32
```

CALL

调用过程(Call a Proedure)

O	D	I	S	Z	A	P	C

将下一条指令的地址压入堆栈并将控制转移到目的地址。如果过程是近过程(在同一个段内)，指令只压入下一条指令的偏移；否则，下一条指令的段和偏移都被压入堆栈

指令格式：

```
CALL  nearlabel          CALL  mem16
CALL  farlabel           CALL  mem32
CALL  reg
```

CBW

字节扩展到字(Convert Byte to Word)

O	D	I	S	Z	A	P	C

将 AL 中的符号位扩展到 AH 中

指令格式：

```
CBW
```

CDQ

双字扩展到 8 字节(Convert Doubleword to Quadword)(IA-32)

O	D	I	S	Z	A	P	C

将 EAX 中的符号位扩展到 EDX 中

指令格式：

```
CDQ
```

CLC

清除进位标志（Clear Carry Flag）

O	D	I	S	Z	A	P	C
							0

清除进位标志

指令格式：

```
CLC
```

CLD

清除方向标志（Clear Direction Flag）

O	D	I	S	Z	A	P	C
	0						

清除方向标志。此时，字符串指令自动增加(E)SI 和(E)DI 的值

指令格式：

```
CLD
```

CLI

清除中断标志（Clear Interrupt Flag）

O	D	I	S	Z	A	P	C
		0					

清除中断标志。禁止可屏蔽硬件中断，直到执行一条 STI 指令为止

指令格式：

```
CLI
```

CMC

进位标志取反（Complement Carry Flag）

O	D	I	S	Z	A	P	C
							*

当前进位标志值变反

指令格式：

```
CMC
```

CMP

比较（Compare）

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

比较目的操作数和源操作数，隐含执行（相应设置标志位，但不改变操作数）从源操作数中减掉目的操作数的减法操作

指令格式：

```
CMP   reg,reg          CMP   reg,imm
CMP   mem,reg          CMP   mem,imm
CMP   reg,mem          CMP   accum,imm
```

CMPS
CMPSB
CMPSW
CMPSD

比较字符串(Compare Strings)

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

比较内存中由 DS:(E)SI 和 ES:(E)DI 寻址的字符串。隐含执行源减去目的的减法操作。CMPSB 比较字节，CMPSW 比较字，CMPSD 比较双字(IA-32 处理器)。(E)SI 和(E)DI 的值依据操作数的大小以及方向标志的状态增减。如果设置了方向标志，减少(E)SI 和(E)DI；否则，增加(E)SI 和(E)DI

指令格式：

```
CMPSB                    CMPSW
CMPSD
```

CMPXCHG

比较并交换(Compare and Exchange)

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

目的操作数和累加器(AL，AX 或 EAX)进行比较，如果相等，源操作数拷贝到目的操作数中。否则，目的操作数拷贝到累加器中

指令格式：

```
CMPXCHG reg,reg          CMPXCHG mem,reg
```

CWDE

字扩展到双字(Convert Word to Extended Double)(IA-32)

O	D	I	S	Z	A	P	C

AX 符号扩展到 EAX 的高字

指令格式：

```
CWDE
```

DAA

加法后进行十进制调整(Decimal Adjust After Addition)

O	D	I	S	Z	A	P	C
?			*	*	*	*	*

调整两个压缩 BCD 值相加之后 AL 中的结果。将和转换成两个 BCD 数存放在 AL 中

指令格式：

```
DAA
```

DAS

减法后进行十进制调整(Decimal Adjust After Subtraction)

O	D	I	S	Z	A	P	C
?			*	*	*	*	*

将两个压缩 BCD 值减法操作后在 AL 中得到的结果转换为两个压缩的 BCD 数并存储在 AL 中

指令格式：

```
DAS
```

DEC

减 1（Decrement）

O	D	I	S	Z	A	P	C
*			*	*	*	*	

从操作数中减去 1。注意：本指令不影响进位标志

指令格式：

```
DEC  reg                    DEC  mem
```

DIV

无符号整数除法（Unsigned Integer Divide）

O	D	I	S	Z	A	P	C
?			?	?	?	?	?

执行 8 位、16 位或 32 位的无符号整数除法操作，如果除数是 8 位的，被除数是 AX，除法操作后商在 AL 中，余数在 AH 中；如果除数是 16 位的，被除数是 DX:AX，除法操作后商在 AX 中，余数在 DX 中；如果除数是 32 位的，被除数是 EDX:EAX，除法操作后商在 EAX 中，余数在 EDX 中

指令格式：

```
DIV  reg                    DIV  mem
```

ENTER

生成堆栈框架（Make Stack Frame）（80286）

O	D	I	S	Z	A	P	C
?			*	*	*	*	*

为接收堆栈参数和使用局部堆栈变量的过程创建堆栈框架。第一个操作数表示要为局部变量保留的字节数。第二个操作数表示过程嵌套层次（C，BASIC，FORTRAN 必须设为 0）

指令格式：

```
ENTER imm16,imm8
```

HLT

停机（Halt）

O	D	I	S	Z	A	P	C

中止 CPU，直到硬件中断发生（注意：只有使用 STI 指令设置了中断标志才可能发生硬件中断）

指令格式：

```
HLT
```

IDIV

有符号整数除法(Signed Integer Divide)

O	D	I	S	Z	A	P	C
?			?	?	?	?	?

对EDX:EAX,DX:AX或AX执行有符号整数除法操作。如果除数是8位的，被除数是AX，商在AL中，余数在AH中；如果除数是16位的，被除数是DX:AX，商在AX中，余数在DX中；如果除数是32位的，被除数是EDX:EAX，商在EAX中，余数在EDX中。通常在执行IDIV指令之前要使用CBW或CBD对被除数进行符号扩展

指令格式：

```
IDIV  reg                    IDIV  mem
```

IMUL

有符号整数乘法(Signed Integer Multiply)

O	D	I	S	Z	A	P	C
*			?	?	?	?	*

对AL，AX或EAX执行有符号整数乘法操作。如果乘数是8位的，被乘数在AL中，积在AX中；如果乘数是16位的，被乘数在AX中，积在DX:AX中；如果乘数是32位的，被乘数在EAX中，积在EDX:EAX中

指令格式：

单操作数：

```
IMUL r/m8                    IMUL r/m16
IMUL r/m32
```

双操作数：

```
IMUL r16,r/m16               IMUL r16,imm8
IMUL r32,r/m32               IMUL r32,imm8
IMUL r16,imm16               IMUL r32,imm32
```

三操作数：

```
IMUL r16,r/m16,imm8          IMUL r16,r/m16,imm16
IMUL r32,r/m32,imm8          IMUL r32,r/m32,imm32
```

IN

从端口输入(Input from Port)

O	D	I	S	Z	A	P	C

从端口输入一个字节或字到AL或AX中。源操作数是端口地址，可以是8位的常量或DX中的一个16位地址。在IA-32处理器上，还可以从端口输入一个双字至EAX中

指令格式：

```
IN  accum,imm                IN  accum,DX
```

INC

加 1（Increment）

O	D	I	S	Z	A	P	C
*			*	*	*	*	

寄存器或内存操作数加 1

指令格式：

```
INC  reg                    INC  mem
```

INS
INSB
INSW
INSD

从端口输入一个字符串（Input from Port to String）（80286）

O	D	I	S	Z	A	P	C

从端口输入一个字符串至 ES：(E)DI 指向的缓冲区中。端口号在 DX 中指定，对于接收到的每个值，(E)DI 的调整方式与 LODSB 及其他类似的字符串操作指令类似。REP 前缀可以和该指令联用

指令格式：

```
INS  dest,DX                REP  INSB  dest,DX
REP  INSW  dest,DX          REP  INSD  dest,DX
```

INT

中断（Interrupt）

O	D	I	S	Z	A	P	C
		0					

产生软件中断，导致调用操作系统的中断服务程序。在转移到中断服务程序之前，该指令清除中断标志，并将标志、CS 和 IP 压入堆栈

指令格式：

```
INT  imm                    INT  3
```

INTO

溢出中断（Interrupt on Overflow）

O	D	I	S	Z	A	P	C
		*	*				

如果设置了溢出标志，该指令产生 4 号 CPU 中断。在 DOS 下调用 INT 4 时，默认的服务程序不采取任何动作，但是用户可以用自己编写的服务程序替换它

指令格式：

```
INTO
```

IRET	中断返回(Interrupt Return) O D I S Z A P C * * * * * * * * 从中断处理例程返回。从堆栈上弹出(E)IP，CS 以及标志 指令格式： `IRET`

Jcondition	条件跳转(Conditional Jump) O D I S Z A P C 如果特定的标志为真则跳转到指定的标号处。在 IA-32 处理器之前，标号距当前位置必须在–128 到+127 字节范围之内；在 IA-32 处理器上，标号距当前位置必须在–32 768 到+32 767 字节范围之内。指令助记符列表参见表 B.2。 指令格式： *`Jcondition label`*

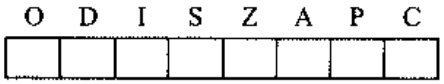
表 B.2 条件跳转助记符[①]

指令助记符	含义	指令助记符	含义
JA	大于则跳转	JE	相等则跳转
JNA	不大于则跳转	JNE	不等则跳转
JAE	大于等于则跳转	JZ	为 0 则跳转
JNAE	不大于等于则跳转	JNZ	不为 0 则跳转
JB	小于则跳转	JS	为负则跳转
JNB	不小于则跳转	JNS	不为负则跳转
JBE	小于或等于则跳转	JC	进位则跳转
JNBE	不小于等于则跳转	JNC	无进位则跳转
JG	大于则跳转	JO	溢出则跳转
JNG	不大于则跳转	JNO	未溢出则跳转
JGE	大于等于则跳转	JP	奇偶位置位则跳转
JNGE	不大于等于则跳转	JPE	奇偶位相等则跳转
JL	小于则跳转	JNP	奇偶位清除则跳转
JNL	不小于则跳转	JPO	奇偶位清除则跳转
JLE	小于或等于则跳转	JNLE	不小于等于则跳转

① ja，jna，jae，jnae，jb，jnb 等带 a 或者 b 的指令用于无符号数比较；jg，jng，jl，jnl 等使用 g 或者 l 的指令用于有符号数的比较——译者注。

JCXZ
JECXZ

CX 为 0 则跳转（Jump if CX Is Zero）



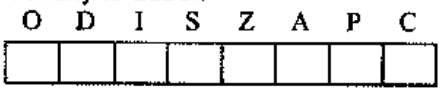
如果 CX 寄存器等于 0 则跳转。标号距其后指令必须在–128 到+127 字节范围之内。在 IA-32 处理器上，JECXZ 表示如果 ECX 等于 0 则跳转

指令格式：

```
JCXZ  label                    JECXZ label
```

JMP

无条件跳转（Jump Unconditionally to Label）



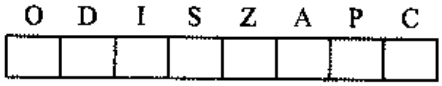
跳转到标号处。短跳转的目的地址距当前位置应在–128 到+127 字节范围之内。近跳转的目的地址应在同一代码段之内。远跳转的目的地址可以在当前段之外

指令格式：

```
JMP  shortlabel            JMP  reg16
JMP  nearlabel             JMP  mem16
JMP  farlabel              JMP  mem32
```

LAHF

将标志送 AH（Load AH from Flags）



传送标志的低 8 位，但不会传送陷阱、中断、溢出、方向和符号标志

指令格式：

```
LAHF
```

LDS
LES
LFS
LGS
LSS

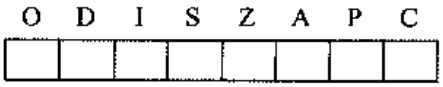
装入远指针（Load Far Pointer）

将双字内存操作数装入段寄存器和特定的目的寄存器中。在 IA-32 处理器之前，LDS 装入 DS，LES 装入 ES；在 IA-32 处理器上，LFS 装入 FS，LGS 装入 GS，LSS 装入 SS

指令格式（LDS，LES，LFS，LGS，LSS 相同）：

```
LDS  reg,mem
```

LEA — 装入有效地址(Load Effective Address)

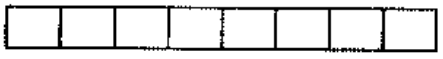


计算并装入 16 位或 32 位的内存操作数的有效地址。类似于 MOV..OFFSET,不同点在于 LEA 指令获取的地址是在运行时进行计算的

指令格式:

```
LEA   reg,mem
```

LEAVE — 高级过程退出(High-Level Procedure Exit)

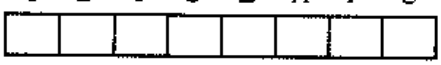


释放过程的堆栈框架。通过将 ESP 和 EBP 恢复为原始值对过程开始的 ENTER 指令进行逆操作

指令格式:

```
LEAVE
```

LOCK — 锁定系统总线(Load the System Bus)

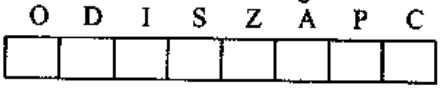


在多处理器的计算机中,其他处理器在下一条指令执行期间暂停运行。在其他处理器可能修改当前 CPU 正在访问的数据时使用该指令

指令格式:

```
LOCK instruction
```

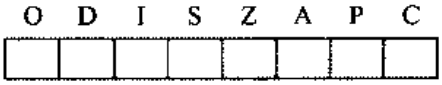
LODS
LOOSB
LODSW
LODSO — 将字符串装入累加器(Load Accumulator from String)

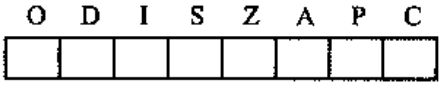


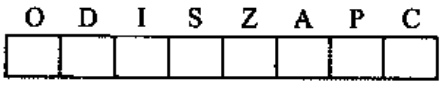
将由 DS:(E)SI 寻址的一个内存字节或字装入累加器(AL, AX 或 EAX)中。如果使用 LODS,必须指定内存操作数。LODSB 将一个字节装入 AL,LODSW 将一个字装入 AX。IA-32 处理器的 LODSD 将一个双字装入 EAX。(E)SI 根据操作数大小和方向标志值自动增减。如果方向标志(DF)=1,ESI 增加;如果 DF=0,ESI 减少

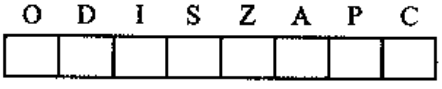
指令格式:

```
LODS   mem                LODSB
LODS   segreg:mem         LODSW
LODSD
```

LOOP LOOPW	循环(Loop)  (E)CX 减 1，如果(E)CX 大于 0 则跳转到一个短标号处。目的标号距当前位置必须在-128到+127 字节范围之内。在 IA-32 上，ECX 用做默认的循环计数器 指令格式： `LOOP shortlabel`　　`LOOPW shortlabel`

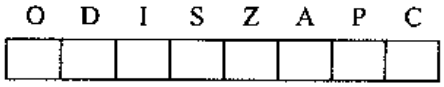
LOOPD	循环(IA-32)(Loop)  (E)CX 减 1，如果(E)CX 大于 0，跳转到一个短标号处。目的标号距当前位置必须在-128到+127 字节范围之内 指令格式： `LOOPD shortlabel`

LOOPE LOOPZ	如果相等[(为 0)则循环(Loop If Equal (Zero)]  (E)CX 减 1，如果(E)CX>0 并且零标志设置则跳转到短标号处 指令格式： `LOOPE shortlabel`　　`LOOPZ shortlabel`

LOOPNE LOOPNZ	如果不等[(不为 0)则循环(Loop If Not Equal (Zero)]  (E)CX 减 1，如果(E)CX>0 并且零标志清除则跳转到短标号处 指令格式： `LOOPNE shortlabel`　　`LOOPNZ shortlabel`

MOV

数据传送(Move)



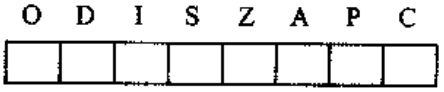
拷贝字节、字源操作数到目的操作数中

指令格式：

```
MOV reg,reg                MOV reg,imm
MOV mem,reg                MOV mem,imm
MOV reg,mem                MOV mem16,segreg
MOV reg16,segreg           MOV segreg,mem16
MOV segreg,reg16
```

MOVS MOVSB MOVSW MOVSD

字符串传送(Move String)



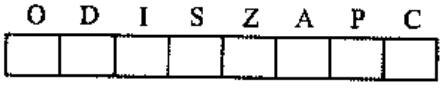
拷贝 DS:(E)SI 寻址的内存操作数至 ES:(E)DI。MOVS 指令要求指定源和目的操作数。MOVSB 拷贝字节，MOVSW 拷贝字。在 IA-32 处理器上，MOVSD 拷贝双字。(E)SI 和(E)DI 的值根据操作数的大小和方向标志的状态增减。如果方向标志(DF) =1，(E)SI 和(E)DI 减少，如果 DF = 0，(E)SI 和(E)DI 增加

指令格式：

```
MOVSB
MOVSW
MOVSD
MOVS dest,source
MOVS ES:dest,segreg:source
```

MOVSX

符号扩展传送(Move With Sign-Extend)



将一个字节或字从源操作数中拷贝到目的寄存器中，并将符号扩展到目的操作数的高半部分。该指令用于将一个 8 位或 16 位的操作数拷贝到更大的目的操作数中

指令格式：

```
MOVSX reg32,reg16          MOVSX reg32,mem16
MOVSX reg16,reg8           MOVSX reg16,m8
```

MOVZX 零扩展传送（Move With Zero-Extend）

O	D	I	S	Z	A	P	C

将一个字节或字从源操作数中拷贝到目的寄存器中并零扩展到目的操作数的高半部分。该指令用于将一个 8 位或 16 位的操作数拷贝到更大的目的操作数中

指令格式：

```
MOVZX reg32,reg16          MOVZX reg32,mem16
MOVZX reg16,reg8           MOVZX reg16,m8
```

MUL 无符号整数乘法（Unsigned Integer Multiply）

O	D	I	S	Z	A	P	C
*			?	?	?	?	*

将 AL，AX 或 EAX 与源操作数相乘。如果源操作数是 8 位的，则与 AL 相乘，积存储在 AX 中。如果源操作数是 16 位的，与 AX 相乘，积存储在 DX:AX 中。如果源是 32 位，与 EAX 相乘，积存储在 EDX:EAX 中

指令相式：

```
MUL reg                    MUL mem
```

NEG 求补（Negate）

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

计算目的操作数的补码并将结果存储在目的中

指令格式：

```
NEG reg                    NEG mem
```

NOP 空操作（No Operation）

O	D	I	S	Z	A	P	C

这条指令什么也不做，可用于计时循环中或用于后续指令的按字边界对齐

指令格式：

```
NOP
```

NOT — 求反(Not)

O	D	I	S	Z	A	P	C

通过将操作数的各位变反执行逻辑非操作

指令格式：

```
NOT  reg                    NOT  mem
```

OR — 或(Inclusive OR)

O	D	I	S	Z	A	P	C
0			*	*	?	*	0

对目的操作数和源操作数的对应数据位执行布尔(位)或操作

指令格式：

```
OR   reg,reg                OR   reg,imm
OR   mem,reg                OR   mem,imm
OR   reg,mem                OR   accum,imm
```

OUT — 输出到端口(Output to Port)

O	D	I	S	Z	A	P	C

在IA-32之前，这条指令将累加器中的一个字节或字输出到端口。端口地址如果在范围0~FFh之间可以是一个常量，也可以在DX中存放0到FFFFh之间的端口地址。在IA-32处理器上，可向端口输出一个双字

指令格式：

```
OUT  imm8,accum             OUT  DX,accum
```

OUTS / OUTSB / OUTSW / OUTSD — 向端口输出字符串(80286)(Output String to Port)

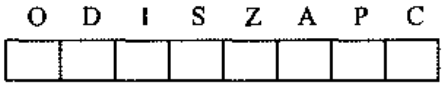
O	D	I	S	Z	A	P	C

将ES:(E)DI指向的字符串输出到端口。端口号在DX中指定。每输出一个值，(E)DI按照与LODSB或其他字符串操作指令类似的方式进行调整。REP前缀可以和该指令联合使用

指令格式：

```
OUTS dest,DX                REP OUTSB dest,DX
REP OUTSW dest,DX           REP OUTSD dest,DX
```

POP 出栈(Pop from Stack)

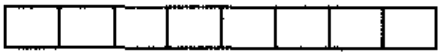


将当前堆栈指针位置处的一个字或双字拷贝到目的操作数中并，并把(E)SP 加 2（或 4）

指令格式：

```
POP   reg16/reg32          POP   segreg
POP   mem16/mem32
```

POPA
POPAD 全部出栈(Pop All)

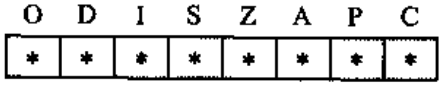


将堆栈顶部的 16 个字节弹出到 8 个通用寄存器中，顺序如下：DI，SI，BP，SP，BX，DX，CX，AX。SP 的值被丢弃，因此 SP 并不重新赋值。POPA 出栈送 16 位的寄存器中，IA-32 处理器上的 POPAD 指令出栈送 32 位的寄存器中

指令格式：

```
POPA                       POPAD
```

POPF
POPFD 标志出栈(Pop Flags from Stack)

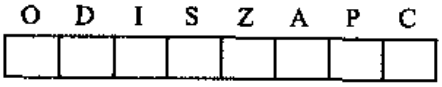


POPF 将堆栈顶部的一个字出栈送至 FLAGS 寄存器。IA-32 上的 POPFD 将堆栈顶部的一个双字出栈送至 32 位的 EFLAGS 寄存器

指令格式：

```
POPF                       POPFD
```

PUSH 压栈(Push on Stack)



(E)SP 减去 2（或 4）并将源操作数拷贝到(E)SP 堆栈指针所指的堆栈位置。在 80186 以上的处理器上，也可以将立即数压入堆栈

指令格式：

```
PUSH   reg16/reg32         PUSH   segreg
PUSH   mem16/mem32         PUSH   imm16/imm32
```

PUSHA
PUSHAD

全部压栈(80286)(Push All)

O	D	I	S	Z	A	P	C

在堆栈中按顺序压入下列寄存器：AX，CX，DX，BX，SP，BP，SI 和 DI。IA-32 的 PUSHAD 指令压入 EAX，ECX，EDX，EBX，ESP，EBP，ESI 和 EDI

指令格式：

```
PUSHA                    PUSHAD
```

PUSHF
PUSHFD

标志压栈(Push Flags)

O	D	I	S	Z	A	P	C

PUSHF 在堆栈上压入 16 位的 FLAGS 寄存器。PUSHFD（IA-32）在堆栈上压入 32 位的 EFLAGS 寄存器

指令格式：

```
PUSHF                    PUSHFD
```

PUSHW
PUSHD

压栈(Push on Stack)

O	D	I	S	Z	A	P	C

PUSHW 在堆栈上压入一个 16 位的字。在 IA-32 上，PUSHD 在堆栈上压入一个 32 位双字

指令格式：

```
PUSH  reg16/reg32        PUSH  segreg
PUSH  mem16/mem32        PUSH  imm16/imm32
```

RCL

带进位循环左移(Rotate Carry Left)

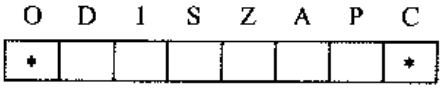
O	D	I	S	Z	A	P	C
*							*

目的操作数循环左移，源操作数决定移位的数量。进位的值拷贝到最低位，最高位送进位标志中。使用 8086/8088 处理器时 Imm8 操作数必须是 1

指令格式：

```
RCL   reg,imm8           RCL   mem,imm8
RCL   reg,CL             RCL   mem,CL
```

RCR 带进位循环右移（Rotate Carry Right）

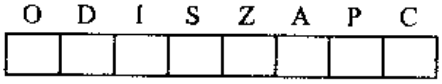


目的操作数循环右移，源操作数决定移位的数量。进位的值拷贝到最高位，最低位送进位标志中。使用 8086/8088 处理器时 Imm8 操作数必须是 1

指令格式：

```
RCR   reg,imm8                RCR   mem,imm8
RCR   reg,CL                  RCR   mem,CL
```

REP 重复字符串操作（Repeat String）

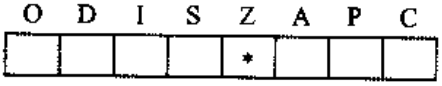


使用(E)CX 作为计数器重复字符串操作指令。每次指令重复的时候(E)CX 减 1，直到(E)CX = 0

指令格式（以 MOVS 为例）：

```
REP MOVS dest,source
```

REP*condition* 有条件重复字符串操作（Repeat String Conditionally）

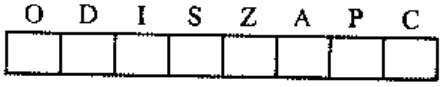


在标志条件为真时重复字符串操作直到(E)CX = 0。REPZ（REPE）当零标志设置时重复，REPZ 和 REPNE 当零标志清除时重复。只有 SCAS 和 CMPS 才能和 REPcondition 联合使用，因为它们是惟一的修改零标志的字符串操作指令

指令格式，以 SCAS 为例：

```
REPZ  SCAS  dest      REPNE  SCAS  dest
REPZ  SCASB           REPNE  SCASB
REPE  SCASW           REPNZ  SCASW
```

RET
RETN
RETF 过程返回（Return from Procedure）

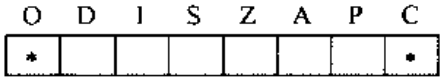


从堆栈上弹出返回地址。RETN（近返回）只从堆栈顶部弹出(E)IP。在实地址模式下，RETF（远返回）首先弹出(E)IP，然后弹出 CS。根据 PROC 伪指令指定的或暗含的属性的不同，RET 可以是近的或远的。可选的 8 位操作数通知 CPU 在弹出返回地址后给(E)SP 加上一个值

指令格式：

```
RET                           RET   imm8
RETN                          RETN  imm8
RETF                          RETF  imm8
```

ROL 循环左移(Rotate Left)

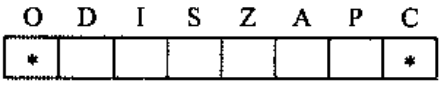


目的操作数循环左移，源操作数决定移位的数目。最高位拷贝到进位标志中并同时送至最低位中。使用8086/8088处理器时imm8操作数必须是1

指令格式：

```
ROL   reg,imm8          ROL   mem,imm8
ROL   reg,CL            ROL   mem,CL
```

ROR 循环右移(Rotate Right)

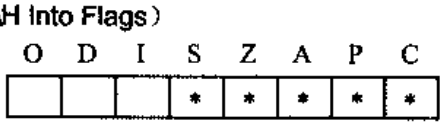


目的操作数循环右移，源操作数决定移位的数目。最低位拷贝到进位标志中并同时送至最高位。使用8086/8088处理器时imm8操作数必须是1

指令格式：

```
ROR   reg,imm8          ROR   mem,imm8
ROR   reg,CL            ROR   mem,CL
```

SAHF AH送标志寄存器(Store AH into Flags)

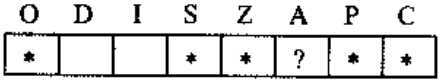


拷贝AH到标志寄存器的位0到位7中

指令格式：

```
SAHF
```

SAL 算术左移(Shift Arithmetic Left)



目的操作数中的每一位左移，源操作数决定移位数目。最高位拷贝到进位标志中，最低位以0填充。使用8086/8088处理器时imm8操作数必须是1

指令格式：

```
SAL   reg,imm8          SAL   mem,imm8
SAL   reg,CL            SAL   mem,CL
```

SAR 算术右移（Shift Arithmetic Right）

O	D	I	S	Z	A	P	C
*			*	*	?	*	*

目的操作数中的每一位右移，源操作数决定移位数目。最低位拷贝到进位标志中，最高位保持原值。SAR 指令通常用于有符号数操作，因为它保留了符号位的值。使用 8086/8088 处理器时 imm8 操作数必须是 1

指令格式：

```
SAR   reg,imm8          SAR   mem,imm8
SAR   reg,CL            SAR   mem,CL
```

SBB 带进位减（Subtract With Borrow）

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

从源操作数中减去目的操作数，然后再减去进位标志值

指令格式：

```
SBB   reg,reg           SBB   reg,imm
SBB   mem,reg           SBB   mem,imm
SBB   reg,mem
```

SCAS
SCASB
SCASW
SCASD 扫描字符串（Scan String）

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

扫描 ES:(E)DI 指向的内存字符串查找与累加器匹配的值。SCAS 要求指定操作数。SCASB 扫描查找与 AL 匹配的 8 位值。SCASW 扫描与 AX 匹配的 16 位值。SCASD 扫描与 EAX 匹配的 32 位值。(E)DI 依据操作数的大小和方向标志的值自动增减。如果 DF = 1，(E)DI 减少，如果 DF = 0，(E)DI 增加

指令格式：

```
SCASB                   SCASW
SCASD                   SCAS ES:dest
SCAS dest
```

SET*condition* 条件设置（Set Conditionally）

O	D	I	S	Z	A	P	C

如果给定的条件为真，目的操作数指定的字节被赋予值 1。如果标志条件为假，目的被赋以值 0。条件的可能值列在本附录前面的表 B.2 中

指令格式：

```
SETcond reg8            SETcond mem8
```

SHL 逻辑左移(Shift Left)

O	D	I	S	Z	A	P	C
*			*	*	?	*	*

将目的操作数的每位左移，使用源操作数决定要移位的数目。最高位拷贝到进位标志中，最低位以0填充(与SAL相同)。在使用8086/8088处理器时imm8操作数必须为1

指令格式：

```
SHL  reg,imm8          SHL  mem,imm8
SHL  reg,CL            SHL  mem,CL
```

SHLD 双精度左移(IA-32)(Double-Precision Shift Left)

O	D	I	S	Z	A	P	C
*			*	*	?	*	*

将第二个操作数的若干位移到第一个操作数中。第三个操作数表示要移位的数目。第一个操作数移出的位由第二个操作数的高位填充。第二个操作数必须是寄存器，第三个操作数可以是立即数或CL寄存器。第二个操作数保持不变

指令格式：

```
SHLD reg16,reg16,imm8      SHLD mem16,reg16,imm8
SHLD reg32,reg32,imm8      SHLD mem32,reg32,imm8
SHLD reg16,reg16,CL        SHLD mem16,reg16,CL
SHLD reg32,reg32,CL        SHLD mem32,reg32,CL
```

SHR 右移(Shift Right)

O	D	I	S	Z	A	P	C
*			*	*	?	*	*

目的操作数中的每一位右移，使用源操作数决定移位的数目。最高位以0填充，最低位拷贝到进位标志中。在使用8086/8088处理器时imm8必须是1

指令格式：

```
SHR  reg,imm8          SHR  mem,imm8
SHR  reg,CL            SHR  mem,CL
```

SHRD 双精度右移(IA-32)(Double-Precision Shift Right)

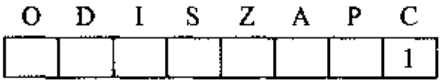
O	D	I	S	Z	A	P	C
*			*	*	?	*	*

将第二个操作数的若干位移到第一个操作数中。第三个操作数表示要移位的数目。第一个操作数移出的位由第二个操作数的低位填充。第二个操作数必须是寄存器，第三个操作数可以是立即数或CL寄存器。第二个操作数保持不变

指令格式：

```
SHRD reg16,reg16,imm8      SHRD mem16,reg16,imm8
SHRD reg32,reg32,imm8      SHRD mem32,reg32,imm8
SHRD reg16,reg16,CL        SHRD mem16,reg16,CL
SHRD reg32,reg32,CL        SHRD mem32,reg32,CL
```

STC 设置进位标志（Set Carry Flag）

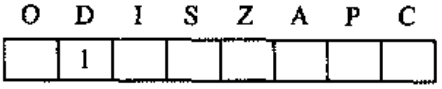


设置进位标志

指令格式：

```
STC
```

STD 设置方向标志（Set Direction Flag）

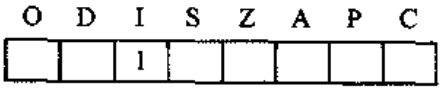


设置方向标志，导致字符串操作指令自动减少(E)SI 和/或(E)DI 的值，这样，字符串处理就能从高地址向低地址进行

指令格式：

```
STD
```

STI 设置中断标志（Set Interrupt Flag）

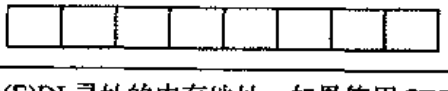


设置中断标志，以允许可屏蔽硬件中断。中断发生时自动禁止中断，因此中断处理程序应使用 STI 立即重新允许中断

指令格式：

```
STI
```

STOS
STOSB
STOSW
STOSD 存储字符串数据（Store String Data）



将累加器内容存储到由 ES:(E)DI 寻址的内存地址。如果使用 STOS，必须指定目的操作数。STOSB 拷贝 AL 到内存中，STOSW 拷贝 AX 到内存中，STOSD 拷贝 EAX 到内存中。(E)DI 根据操作数的尺寸和方向标志的状态值增减。如果 DF = 1，(E)DI 增加；如果 DF = 0，(E)DI 减少

指令格式：

```
STOSB                    STOSW
STOSD                    STOS ES:mem
STOS mem
```

SUB

减法(Subtract)

O	D	I	S	Z	A	P	C

从目的操作数中减去源操作数

指令格式:

```
SUB   reg,reg          SUB   reg,imm
SUB   mem,reg          SUB   mem,imm
SUB   reg,mem          SUB   accum,imm
```

TEST

测试(Test)

O	D	I	S	Z	A	P	C
*			*	*	?	*	0

测试目的操作数的单个位。指令执行逻辑与操作，影响标志值但不改变目的操作数的内容

指令格式:

```
TEST  reg,reg          TEST  reg,imm
TEST  mem,reg          TEST  mem,imm
TEST  reg,mem          TEST  accum,imm
```

WAIT

等待协处理器(Wait for Coprocessor)

O	D	I	S	Z	A	P	C

挂起 CPU，直到协处理器完成当前的指令

指令格式:

```
WAIT
```

XADD

加交换(Exchange and Add)(Intel 486)

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

源操作数与目的操作数相加。同时，原始的目的操作数送至源操作数

指令格式:

```
XADD  reg,reg          XADD  mem,reg
```

XCHG

交换(Exchange)

O	D	I	S	Z	A	P	C

交换源操作数和目的操作数的内容

指令格式:

```
XCH   reg,reg          XCH   mem,reg
XCH   reg,mem
```

XLAT
XLATB
	字节转换（Translate Byte）
	O D I S Z A P C
	使用 AL 中的值作为由 DS:BX 指向的一张表的索引，该索引指向的字节送至 AL。可以使用操作数指定一个段超越前缀。XLATB 可以用 XLAT 代替
	指令格式：
	`XLAT`　　`XLAT segreg:mem`
	`XLAT mem`　　`XLATB`

XOR	异或（Exclusive OR）
	O D I S Z A P C
	源操作数的每位同目的操作数的对应位进行异或操作。只有当原始操作数的数据位与目的操作数的对应位不同时结果才为 1
	指令格式：
	`XOR reg,reg`　　`XOR reg,imm`
	`XOR mem,reg`　　`XOR mem,imm`
	`XOR reg,mem`　　`XOR accum,imm`

附录 C　BIOS 和 MS-DOS 中断

本章要点：

- C.1 简介
- C.2 PC 中断
- C.3 中断 21h 的功能（MS-DOS 服务）
- C.4 中断 10h 的功能（视频 BIOS 服务）
- C.5 键盘 BIOS INT 16h 的功能
- C.6 鼠标功能（INT 33h）

C.1　简介

本附录列出了更多的常用中断，内容共分成 4 组：

- PC 中断号列表。这些中断号对应于存储在内存低端 1024 字节内的中断向量
- INT 10h 视频 BIOS 功能
- INT 16h 键盘 BIOS 功能
- INT 21h MS-DOS 功能

事实上，PC 中断的文档化是一项艰巨的任务，这是由于有许多不同版本的 MS-DOS，各种 DOS 扩展以及多种 PC 硬件控制器。关于中断最好的资源是 Ralf Brown 的中断列表，可在互联网上以各种形式获得。当然，由于相关的网址和内容经常改变，读者可以通过访问作者的站点获取 Ralf Brown 中断列表以及其他汇编网站的最新链接，网址是：

http://www.nuvisionmiami.com/asmsources

C.2　PC 中断

表 C.1　PC 中断号列表

中断号	描述
0	除法错误。CPU 产生：当试图除零时发生
1	单步。CPU 产生：当陷阱标志设置时发生
2	不可屏蔽中断。外部硬件：当发生内存错误时产生
3	断点：CPU 产生：当执行机器码 0CCh（INT 3）时发生
4	除法溢出。CPU 产生：溢出标志设置的条件下执行 INTO 指令时发生
5	打印屏幕。执行 INT 5 或按下 Shift-PrtSc 键时发生
6	无效操作码（80286+）
7	处理器扩展不可用（80286+）

（续表）

中断号	描述
8	IRQ0：系统时钟中断。更新 BIOS 数据区，每秒 18.2 次。读者自身的程序需要使用时钟中断时请参见 1Ch
9	IRQ1：键盘硬件中断。键盘按下的时候发生，从键盘端口读入按键并将它存储在键盘缓冲区中
0A	IRQ2：可编程中断控制器
0B	IRQ3：串行通信口（COM2）
0C	IRQ4：串行通信口（COM1）
0D	IRQ5：固定硬盘
0E	IRQ6：磁盘中断
0F	IRQ7：并行打印口
10	视频服务。控制视频显示的例程（完整列表见表 C.3）
11	设备检查。返回一个字，显示连接到系统中的所有外围设备
12	内存大小。在 AX 中返回内存的数量（以 1024 字节的块为单位）
13	软盘服务。重设软盘控制器，获取最近磁盘访问的信息，读写物理扇区，格式化磁盘
14	异步端口（串行）服务。初始化、读写异步通讯信，返回端口的状态
15	磁带控制器
16	键盘服务。读取或检查键盘输入（完整列表参见表 C.4）
17	打印机服务。初始化、打印以及返回打印机的状态
18	ROM BASIC。执行 ROM 中固化的 BASIC 解释程序
19	引导加载器。重启 MS-DOS
1A	每天的时间。获取自机器启动时开始的时间计数，或将该计数设为新值。计数每秒发生 18.2 次
1B	键盘中断。当 Ctrl-Break 按下的时候中断处理程序由 INT 9h 执行
1C	用户可以使用的时钟中断。空例程，每秒执行 18.2 次。可由自己的程序使用
1D	视频参数。指向包含视频控制芯片初始化信息的一张表
1E	磁盘参数。指向包含磁带控制芯片初始化信息的一张表
1F	图形字体表
20	结束程序。结束一个 COM 程序（应该尽量使用 INT 21h 4Ch 功能代替）
21	MS-DOS 服务
22	MS-DOS 终结地址。指向父程序或进程中的地址。如果当前程序结束，则跳转到该地址
23	MS-DOS 中断地址
24	MS-DOS 关键错误地址。如果当前程序中有严重错误，比如磁盘介质错误时，则跳转到该地址
25	绝对磁盘读（已过时）
26	绝对磁盘写（已过时）
27	结束并驻留（已过时）

（续表）

中断号	描述
28~FF	保留
33	Microsoft 鼠标。跟踪和控制鼠标的功能
34-3E	浮点模拟器
3F	覆盖管理器
40~41	固定磁盘服务。固定磁盘控制器
42~5F	保留：特殊用途
60~6B	应用程序可用
6C~7F	保留：特殊用途
80~F0	保留：由 ROM BASIC 使用
F1~FF	应用程序可用

C.3　中断 21h 的功能（MS-DOS 服务）

中断 21h 在 MS-DOS 服务中有许多功能，在这里仅列出一些最常用的功能。

表 C.2　中断 21h 的功能（MS-DOS 服务）

功能	描述
1	从标准输入读取字符。如果没有字符则等待输入。返回：AL = 字符
2	在标准输出上显示字符。接收：DL = 字符
3	从标准辅助输入读取字符（串口）
4	向标准输助输出写字符（串口）
5	向打印机写字符。接收：DL = 字符
6	直接控制台输入/输出。如果 DL = FFh，则从标准输入上读取字符。如果 DL 是其他值，则在标准输出上显示该字符
7	无回显直接字符输入。等待标志输入设备输入一个字符。返回：AL = 字符
8	无回显字符输入。等待标志输入设备输入一个字符。返回：AL = 字符。字符不回显，可以用 Ctrl-Break 结束等待
9	在标准输出上显示字符。接收：DS:DX = 字符串地址
0A	缓冲键盘输入。从标准输入设备读取一个字符串。接收：DS:DX 指向预定义的键盘结构。
0B	检查标准输入状态。检查是否有输入字符在等待。返回：如果字符在等待，AL = 0FFh，否则 AL = 0
0C	清除键盘缓冲区并调用输入功能。清除控制台输入缓冲，并执行一个输入功能。接收：AL = 期望的功能（1, 6, 7, 8 或 0Ah）
0E	选择默认驱动器。接收：DL = 驱动器号（0 = A，1 = B，等）。
0F-18	FCB 文件功能（已过时）
19	获取当前的默认驱动器。返回：AL = 驱动器号（0 = A，1 = B 等）
1A	设置磁盘传输地址。接收：DS:DX 包含磁盘传输区域的地址

（续表）

功能	描述
25	设置中断向量。将中断向量表中的表项设置为新的地址。接收：DS:DX 指向要插入到表中的中断处理程序地址；AL = 中断向量号
26	创建新的程序段前缀。接收：DX = 新的 PSP 的段地址
27-29	FCB 文件功能（已过时）
2A	获取系统日期。返回：AL = 星期（0~6，Sunday = 0），CX = 年，DH = 月，DL = 日
2B	设置系统日期。接收：CX = 年，DH = 月，DL = 日。返回：如果日期有效 AL = 0
2C	获取系统时间。返回：CH = 小时，CL = 分钟，DH = 秒，DL = 百秒数
2D	设置系统时间。接收：CH = 小时，CL = 分钟，DH = 秒。返回：如果时间有效，AL = 0
2E	设置校验标志。接收：AL = MS-DOS 校验标志的新值（0 = 关闭，1 = 打开），DL = 00h
2F	获取磁盘传输地址（DTA）。返回：ES:BX = 地址
30	获取 MS-DOS 的版本号。返回：AL = 主版本号，AH = 次版本号，BH = OEM 序列号，BL:CX = 24 位用户序列号
31	结束并驻留。结束当前的程序或进程，将其部分留在内存中。接收：AL = 返回码，DX = 请求的小节数
32	获取 MS-DOS 驱动器参数块。接收：DL = 驱动器号。返回：AL = 状态，DS:BX 指向驱动器参数块
33	扩展中断检查。指示 MS-DOS 是否检查 Ctrl-Break
34	获取 INDOS 标志（未公开）
35	获取中断向量。接收：AL = 中断向量号。返回：ES:BX = 中断处理程序的段/偏移地址
36	获取磁盘的空闲空间（只用于 FAT16）。接收：DL = 驱动器号（0 = 默认，1 = A 等）。返回：AX = 每簇扇区数，如果驱动器号无效 AX 返回 FFFFh；BX = 可用的簇数，CX= 每扇区字节数，DX = 每驱动器簇数
37	获取 switch 字符（命令行参数的前导字符）（未公开）
38	获取或设置国家信息（细节参见 Duncan 的书或 Brown 的中断列表）
39	创建子目录。接收：DS:DX 指向包含路径和目录名的 ASCII 字符串。返回：如果设置了进位标志 AX 中返回错误码
3A	删除子目录。接收：DS:DX 指向包含路径和目录名的 ASCII 字符串。返回：如果设置了进位标志 AX 中返回错误码
3B	改变当前目录。接收：DS:DX 指向包含断路径的 ASCII 字符串。返回：如果设置了进位标志 AX，则返回错误码
3C	创建或前裁文件。创建一个新文件或将已存文件剪裁为 0 字节。接收：DS:DX 指向包含文件名的 ASCII 字符串，CX = 文件属性。返回：如果设置了进位标志 AX 中返回错误码；否则 AX 中返回新的文件句柄
3D	打开已存在的文件。打开文件进行输入、输出或输入输出。接收：DS:DX 指向包含文件名的 ASCII 字符串，AL = 访问码（0 = 读，1 = 写，2 = 读/写）。返回：如果发生错误，设置进位标志并在 AX 中返回错误码；否则 AX 中返回新的文件句柄

（续表）

功能	描述
3E	关闭文件句柄。关闭文件句柄指定的文件或设备。接收：BX = 前面创建或打开的文件句柄。返回：如果进位标志设置，则AX = 错误码
3F	读文件或设备。从文件或设备读取指定的字节数。接收：BX = 文件句柄，DS:DX指向输入缓冲区，CX = 要读的字节数。返回：如果进位标志设置，AX = 错误码；否则，AX = 读取的字节数
40	写文件或设备。向文件或设备写指定数目的字节。接收：BX = 文件句柄，DS:DX指向输入缓冲区，CX = 要写的字节数。返回：如果进位标志设置，AX = 错误码；否则AX = 写入的字节数
41	删除文件。删除指定的文件。接收：DS:DX指向包含文件名的ASCII字符串。返回：如果设置进位标志AX = 错误码
42	移动文件指针。根据指定的方法移动文件的读/写指针。接收：CX:DX = 文件指针的移动距离，AL = 方式码，BX = 文件句柄。方式码如下：0 = 偏移相对于文件开始，1 = 偏移相对于当前位置，2 = 偏移相对于文件的末尾
43	获取/设置文件属性。获取或设置文件的属性。接收：DS:DX = 指向包含文件名的ASCIIZ串，CX = 属性，AL = 功能号（1 = 设置属性，0 = 获取属性）。返回：如果进位标志被设置，则AX = 错误码
44	设备的I/O控制。获取或设置与打开的文件句柄关联的设备信息。或者向设备句柄发送一个控制字符串，或从设备句柄接收一个控制字符串
45	复制文件句柄。为当前打开的文件返回一个新的文件句柄。接收：BX = 文件句柄。返回：如果进位标志设置，则AX = 错误码
46	强制复制文件句柄。强制CX内的文件句柄与BX内的文件句柄引用同一文件的同一位置。接收：BX = 已存文件的句柄，CX = 第二个文件句柄。返回：如果设置进位标志，则AX =错误码
47	获取当前目录。获取当前目录的全路径名。接收：DS:SI 指向存放存放目录路径的64字节区域，DL = 驱动器号。返回：DS:SI指向的缓冲区以路径填充，如果进位标志设置AX = 错误码
48	分配内存。分配请求的内存，按小节数（16字节的块）计算。接收：BX = 请求的节数。返回：AX = 已分配块所在的段，BX = 可用的最大块（按节计算），如果设置了进位标志AX = 错误码
49	释放已分配的内存块。释放前面使用功能48h分配的内存。接收：ES = 要释放块所在的段。返回：如果进位标志设置AX = 错误码
4A	修改内存块。修改已分配的内存块的大小。内存块可能会增大或减小。接收：ES = 块所在的段，BX = 请求的节数。返回：如果设置进位标志AX = 错误码，BX = 最多可用块
4B	加载或执行程序。为其他程序创建程序段前缀，将其加载进内存并执行。接收：DS:DX指向包含路径名的ASCIIZ字符串，ES:BX指向参数块，AL = 功能值。AL内的功能值如下：0 = 加载并执行程序，3 = 加载但不执行（覆盖程序）。返回：如果设置了进位标志，则AX = 错误码

（续表）

功能	描述
4C	结束进程。结束程序返回到 MS-DOS 或调用程序的常用方法。接收：AL = 8 位的返回码，可以用 MS-DOS 的 4Dh 功能或批处理文件中的 ERRORLEVEL 命令查询
4D	获取进程的返回码。获取进程或程序的返回码，错误码由 31h 或 4Ch 功能调用产生。返回：AL = 程序的 8 位返回码，AH = 产生退出的类型：0 = 正常退出，1 = 按下 Ctrl-Break 退出，2 = 由关键设备的错误退出，3 = 调用功能 31h 退出
4E	查找第一个匹配文件。查找与给定文件名匹配的第一个文件。接收：DS:DX 指向要查找的文件名；CX = 搜索时使用的文件属性。返回：如果设置了进位标志 AX = 错误码；否则当前的 DTA 以文件的名字、属性、时间、日期、大小填充。通常在该功能调用之前要调用 DOS 功能 1Ah（设置 DTA）
4F	查找下一个匹配文件。查找与给定文件名匹配的下一个文件。应该在 DOS 功能 4Eh 调用之后调用该功能。退回：如果进位标志设置，AX = 错误码；否则当前的 DTA 填充文件的相关信息
54	获取 verify 标志。退回：AH = 磁盘 I/O 的 verify 标志（0 = 关闭，1 = 开）
56	重命名/移动文件。重命名文件或将其移动到其他目录中。接收：DS:DX 指向包含文件名的 ASCIIZ 字符串，ES:DI 指向新的路径和文件名。返回：如果进位标志设置，则 AX = 错误码
57	获取/设置文件的日期/时间。获取或设置文件的时间、日期戳记。接收：若要获取日期时间，AL = 0；若要设置日期时间，AL = 1；BX = 文件句柄，CX = 当前文件的时间，DX = 当前文件的日期
58	获取或设置内存分配策略（详细信息见 Duncan 的书或 Brown 的中断列表）
59	获取扩展错误信息。返回 MS-DOS 错误的其他信息，包括错误类别、位置和推荐动作。接收：BX = MS-DOS 版本号（版本 3.xx 为 0）。返回：AX = 扩展错误码，BH = 错误类别，BL = 建议动作，CH = 位置
5A	创建临时文件。在指定目录内生成一个独一无二的文件。接收：DS:DX 指向以反斜线（\）结尾包含路径的 ASCIIZ 串；CX = 期望的文件属性。返回：如果设置了进位标志，AX = 错误码；否则，DS:DX 指向创建的文件的全路径
5B	创建新文件。试图创建新文件，但是如果文件名已经存在则失败，这防止了覆盖已存在的文件。接收：DS:DX 指向包含文件名 ASCIIZ 串。返回：如果设置了进位标志，则 AX = 错误码，否则 AX = 文件句柄
5C~61	省略
62	获取程序段前缀（PSP）地址。返回：BX = 当前程序段前缀的段值
7303h	获取磁盘剩余空间。填充一个包含磁盘空间详细信息的结构。接收：AX = 17303h，ES:DI 指向 ExtGetDskFreSpcStruc 结构，CX = ExtGetDskFreSpcStruc 结构的大小，DS:DX 指向一个包含驱动器名的 ASCIIZ 串。返回：ExtGetDskFreSpcStruc 结构填充了磁盘信息。细节参见 14.5.1 小节
7305h	绝对磁盘读写。读写单个或一组磁盘扇区。在 Widnows NT/2000/XP 下不能工作接收：AX = 7305h，DS:BX = DISKIO 结构变量的段/偏移地址，CX = 0FFFFh，DL = 驱动器号（0 =默认，1 = A，2 = B，3 = C 等），SI = 读写标志。细节参见 14.4 节

C.4　中断 10h 的功能（视频 BIOS 服务）

表 C.3　中断 10h 的功能（视频服务）

功能	描述
0	设置显示模式。将视频显示设为单色、文本、图形或彩色模式。接收：AX= 显示模式
1	设置光标线。设置光标的起始和结束扫描线。接收：CH = 光标起始线，CL = 光标的结束线
2	设置光标位置。在屏幕上定位光标。接收：BH = 视频页，DH = 行，DL = 列
3	获取光标位置。获取光标的屏幕位置和大小。接收：BH = 视频页。返回：CH = 光标的起始扫描线，CL = 光标的结束扫描线，DH = 行，DL = 列
4	读光笔。读取光笔的位置和状态信息。返回：CH = 像素行，CL = 像素列，DH = 字符行，DL = 字符列
5	设置显示页。选择要显示的视频页。接收：AL = 期望的视频页号
6	上卷窗口。上卷当前视频页上的窗口，空出的行以空白填充。接收：AL = 卷动的行数，BH = 卷动行的属性，CX = 左上角的行列位置，DX = 右下角的行列位置
7	下卷窗口。下卷当前视频页上的窗口，空出的行以空白填充。接收：AL = 卷动的行数，BH = 卷动行的属性，CX = 左上角的行列位置，DX = 右下角的行列位置
8	读取字符及其属性值。读取当前光标位置的字符及其属性值。接收：BH = 显示页。返回：AH = 属性字节，AL = ASCII 码
9	显示指定属性的字符。在当前光标位置显示属性及字符。接收：AL = ASCII 字符，BH = 视频页，BL = 属性，CX = 重复计数
0A	显示字符。在当前光标位置显示字符（无属性字节）。接收：AL = ASCII 字符，BH = 视频页，CX = 重复计数
0B	设置调色板。为 CGA 适配卡的选择一组可用的颜色。接收：BH = 显示方式，BL = 调色板 ID
0C	显示像素。在彩色图形模式下写像素。接收：AL = 像素值，CX = X 坐标，DX = Y 坐标
0D	读取像素。读取指定位置像素点的色彩值。接收：CX = X 坐标，DX = Y 坐标。返回：AL = 像素值
0E	显示字符。在屏幕上显示字符并前进光标。接收：AL = ASCII 字符码，BH = 视频页，BL = 属性或颜色
0F	获取当前视频模式。获取当前视频模式。返回：AL = 视频模式，BH = 活跃的视频页
10	设置视频调色板（只用于 EGA 和 PCjr）。设置视频调色板寄存器，边界颜色或闪烁/亮度位。接收：AL = 功能码（00 = 设置调色板寄存器，01 = 设置边界颜色，02 = 设置调色板和边界颜色，03 = 设置/重设亮度位），BH = 颜色，BL = 要设置的调色板寄存器。如果 AL = 2，ES:DX 指向一个颜色列表
11	字符发生器。为 EGA 显示选择字符大小。比如，8 乘 8 字体用于 43 行的显示，8 乘 14 的字体用于 25 行的显示
12	其他选择功能。返回 EGA 显示的技术信息

（续表）

功能	描述
13	显示字符串（用于 PC/AT）。在视频显示上显示字符串文本。接收：AL = 模式，BH = 页号，BL = 属性，CX = 字符串的长度，DH = 行，DL = 列，ES:BP 指向字符串（在 IBM-PC 或 PC/XT 上不能工作）

C.5 键盘•BIOS INT 16h 的功能

表 C.4 键盘 BIOS 中断 16h 功能

功能	描述
03h	设置击键重复速率。接收：AH = 03h，AL = 5，BH = 重复延迟，BL = 重复速率。BH 中的延迟值可以是：0 = 250 ms，1 = 500 ms，2 = 750 ms，3 = 1000 ms。BL 中的重复速率可以是 0（最快）到 1Fh（最慢）之间的值。返回值：无
05h	按键送入缓冲区。将键盘字符和其对应的扫描码送入键盘缓冲区。接收：AH = 05h，CH = 扫描码，CL = 字符。如果缓冲区已满，设置进位标志，AL = 1
10	等待按键。等待输入字符及其扫描码。接收：AH = 10h。返回：AH = 扫描码，AL = 字符的 ASCII 码（功能 00h 同 10h 功能相同，不过只用于老式键盘）
11	检查按键缓冲区。检查是否有字符在键盘缓冲区中等待。接收：AH = 01h。返回：如果有按键在等待，扫描码在 AH 中返回，ASCII 码在 AL 中返回，清除零标志（但字符仍在缓冲区中）。如果没有按键等待，设置零标志（功能 01h 同 11h 功能相同，不过只用于老式键盘）
12	获取键盘标志。返回在 RAM 低端存储的键盘标志字节。接收：AH = 12h。返回：在 AX 中返回键盘标志（功能 02h 同 12h 功能相同，不过只用于老式键盘）

C.6 鼠标功能（INT 33h）

INT 33h 鼠标功能在 AX 寄存器中接收功能号，关于这些功能的更多信息，参见 15.6 节。其他鼠标功能参见表 15.7。

表 C.5 NT 33h 鼠标功能

功能	描述
0000h	重设鼠标并获取鼠标状态。接收：AX = 0000h。重设鼠标并确认其可用。鼠标（如果找到了）被定位在屏幕的中央，其显示页被设为页 0，指针被设置为隐藏状态，mickey 到像素的比率和速度设为默认值。鼠标的移动范围设置为整个屏幕
0001h	显示鼠标指针。接收：AX = 0001h。返回值：无。鼠标驱动保留一个该功能调用次数的计数
0002h	隐藏鼠标指针。AX = 0002h。返回值：无。在鼠标不可见的时候仍然跟踪其位置
0003h	获取鼠标的位置和状态。AX = 0003h。返回值：BX = 鼠标按钮的状态，CX = X 坐标（像素），DX = Y 坐标（像素）
0004h	设置鼠标位置。AX = 0004h，CX = X 坐标（像素），DX = Y 坐标（像素）返回值：无

（续表）

功能	描述
0005h	获取按钮的下压信息。接收 AX = 0005h，BX = 按钮 ID（0 = 左键，1 = 右键，2 = 中键）。返回 AX = 鼠标按键状态，BX = 按钮下压计数，CX = 最后按下按钮的 X 坐标，DX = 最后按下按钮的 Y 坐标
0006h	获取按钮的释放信息。接收 AX = 0006h，BX = 按钮 ID（0 = 左键，1 = 右键，2 = 中键）。返回 AX = 鼠标按键状态，BX = 按钮释放计数，CX = 最后释放按钮的 X 坐标，DX = 最后释放按钮的 Y 坐标
0007h	设置水平限制。接收 AX = 0007h，CX = 最小 X 坐标（像素），DX = 最大 X 坐标（像素）。返回值：无
0008h	设置垂直限制。接收 AX = 0008h，CX = 最小 Y 坐标（像素），DX = 最大 Y 坐标（像素）。返回值：无

附录 D　MASM 参考手册

本章要点：

- 简介
- MASM 的保留关键字
- 寄存器名称
- Microsoft 汇编稿译器（ML）
- 链接（LINK）
- CodeView 调试器
- MASM 伪指令

D.1　简介

Microsoft MASM 6.11 手册最后一次印刷是在 1992 年，它由 3 卷构成：

- 程序员指南
- 参考手册
- 环境和工具

遗憾的是，印刷的手册已经很多年没有再版了，但是 Microsoft 在 Platform SDK 包中提供了手册的电子版（MS-Word 文件）。到现在，印刷的手册肯定早已是收藏家的收藏品了。

本章中的信息摘自参考手册的第 1 章至第 3 章，也包含了 MASM 6.14 readme.txt 文件中更新的内容。

格式符号　本附录使用一致的符号表示法：大写的单词表示是MASM的保留字，但是保留字在程序中可能以大写形式也可能以小写形式出现。在下例中，DATA 是一个保留字：

.DATA

斜体的单词表示是已定义的术语或类型。在下例中，*number* 是指一个整数常量：

ALIGN [*number*]

当使用方括号括起一个项目的时候，表示项目是可选的。在下例中，*text* 是可选的：

[*text*]

当两个或多个项目之间出现垂直分隔符时，必须选择其中的一个项目。下例表示要在 NEAR 和 FAR 之间进行选择：

NEAR|FAR

省略号（…）表示项目列表中的最后一个项目要进行重复。在下例中，*initializer* 之后的逗点

表示它要多次重复：

[*name*] BYTE *initializer* [,*initializer*] …

D.2　MASM的保留关键字

下表是各种MASM伪指令的操作数，它们也是保留字。也就是说它们不能用做标识符（标号、常量等）。

$	PARITY?
?	PASCAL
@B	QWORD
@F	REAL4
ADDR	REAL8
BASIC	REAL10
BYTE	SBYTE
C	SDWORD
CARRY?	SIGN?
DWORD	STDCALL
FAR	SWORD
FAR16	SYSCALL
FORTRAN	TBYTE
FWORD	VARARG
NEAR	WORD
NEAR16	ZERO?
OVERFLOW?	

D.3　寄存器名称

AH	CR0	DR1	EBX	SI
AL	CR2	DR2	ECX	SP
AX	CR3	DR3	EDI	SS
BH	CS	DR6	EDX	ST
BL	CX	DR7	ES	TR3
BP	DH	DS	ESI	TR4
BX	DI	DX	ESP	TR5
CH	DL	EAX	FS	TR6
CL	DR0	EBP	GS	TR7

D.4 Microsoft 汇编编译器（ML）

ML 程序（ML.EXE）可以用来编译并链接一个或多个汇编语言源文件，它的命令行选项是大小写敏感的，格式为：

ML [*options*] *filename* [[*options*] *filename*]…[/link *linkoptions*]

命令行中要求至少有一个 *filename* 参数，其中 *filename* 是汇编语言编写的源文件名。例如下面的命令行编译源文件 AddSub.asm 并生成目标文件 AddSub.obj：

```
ML -c AddSub.asm
```

options 参数由 0 到多个命令行选项构成，每个选项以一个斜线或减号开始，多个选项之间必须以至少一个空格分隔。表 D.1 列出了全部命令行选项，命令行选项是大小写敏感的。

表 D.1 部分 ML 的命令行选项

选项	含义
/AT	允许支持微型内存模式。对与.COM 文件格式相冲突的代码格式发出错误信息。注意这和.MODEL TINY 伪指令并不完全相同
/Bl*filename*	选择一个链接器
/c	只编译，不链接
/Cp	保留所有用户定义标识符的大小写
/Cu	映射所有标识符为大写
/Cx	保留公共和外部符号的大小写（默认）
/D*sysmbol*[=*value*]	定义给定名字的文本宏。如果没有 *value*，文本宏为空。以空格隔开的多个符号必须以引号引起来
/EP	生成一个预定义的源文件列表（发送到 STDOUT）。参见/Sf
/F *hexnum*	将堆栈大小设为 *hexnum* 字节（这与/link /STACK:number 是相同的）。value 必须以 16 进制格式表示。F 和 *hexnum* 之间必须有一个空格
/Fe*filename*	指定可执行文件名
/Fl[*filename*]	生成一个汇编代码列表文件。参见/Sf
/Fm[*filename*]	创建一个链接映像文件
/Fo*filename*	指定目标文件名
/FPi	为浮点运算生成模拟代码（只用于混合语言编程）
/Fr[*filename*]	生成.SBR 源浏览文件
/FR[*filename*]	生成扩展形式的.SBR 源浏览文件
/Gc	指定使用 FORTRAN 或 Pascal 格式的函数调用和命名约定。与 OPTION LANGUAGE:PASCAL 是相同的
/Gd	指定使用 C 格式的函数调用和命名约定。与 OPTION LANGUAGE:C 是相同的
/H *number*	外部名字限制为 number 个有效字符，默认是 31 个字符
/help	调用 ML 的快速帮助
/I *pathname*	设置包含文件路径。最多允许 10 个/I 选项
/nologo	在编译成功的情况下屏蔽编译器输出的信息

（续表）

选项	含义
/Sa	打开所有可用信息列表
/Sc	在列表文件中增加指令执行时间信息
/Sf	在列表文件中增加第一遍编译列表
/Sg	打开汇编生成的代码列表
/Sl *width*	设置列表文件中行的宽度，按每行字符数计算。范围在60~255之间，或者为0。默认情况下是0，同PAGE *width*
/Sn	生成列表文件时关闭符号表
/Sp *length*	设置列表文件每页的长度，按每页行数计算。范围从10~255，或者为0，默认情况下是0。同PAGE *length*
/Ss *text*	为列表文件指定文本，同SUBTITLE *text*
/St *text*	为列表文件指定标题，同TITLE *text*
/Sx	打开列表文件中的错误条件信息
/Ta *filename*	汇编不以.ASM扩展名结尾的源文件
/w	同/W0
/W*level*	设置警告级别，level = 0, 1, 2, 3
/WX	如果产生警告则返回错误
/Zd	在目标文件中生成行号信息
/Zf	使所有符号变成公共符号
/Zi	在目标文件中生成CodeView信息
/Zm	允许M510选项，以最大程度地同MASM 5.1兼容
/Zp[*alignment*]	将结构按指定的字节边界对齐。*alignment*可以是1，2或4
/Zs	只进行参数检查
/?	显示ML命令行选项的帮助信息

D.5 链接（LINK）

下列信息适用于16位的Microsoft链接器。LINK实用工具将目标文件链接成一个可执行文件或动态链接库。格式为：

LINK *options objfiles*[,[*exefile*][,[*mapfile*][,[*libraries*][,[*deffile*]]]]][;]

选项 表D.2描述了LINK的命令行选项，其中包括了一些帮助文档中很少提到的选项。

表D.2 16位LINK的命令行选项

选项	含义
/A:size	选项名：/A[LIGNMENT]。指示LINK在分段的可执行文件中按照size指定的字节数对齐段数据。size必须是2的幂
/B	选项名：/B[ATCH]。禁止显示库或目标文件未找到的信息
/CO	选项名：/CO[DEVIEW]。增加Mcirosoft的CodeView调试器需要的符号数据和行号信息。这个选项同/EXEPACK选项是不相容的

（续表）

选项	含义
/CP:*number*	选项名：/CP[ARMAXALLOC]。设置程序需要的最大内存数量为 number，以节（16 字节）为单位
/DO	选项名：/DO[SSEO]。以 Microsoft 的高级语言使用的默认顺序组织段
/DS	选项名：/DS[ALLOCATE]。指示链接器加载数据段高端开始的所有数据。该选项用于创建 MS-DOS .EXE 可执行文件
/E	选项名：/E[XEPACK]。压缩可执行文件，该选项同/INCR 和/CO 不相容。不要将/EXEPACK 选项用于基于 Windows 的应用程序
/F	选项名：/F[ARCALLTRANSLATION]。优化远程调用，使用/TINY 选项时该选项自动打开。在链接基于 Windows 的应用程序时不推荐和/PACKC 选项同时使用该选项
/HE	选项名：/HE[LP]。调用 LINK 的快速帮助
/HI	选项名：/HI[GH]。尽可能将可执行文件加载到内存高端。/HIGH 和/DSALLOC 联合使用。该选项用于创建 MS-DOS .EXE 可执行文件
/INC	选项名：/INC[REMENTAL]。准备使用 ILINK 进行增量链接。该选项同/EXEPACK 和/TINY 不相容
/INF	选项名：/INF[ORMATION]。在标准输出上显示链接阶段以及正被链接的目标文件名
/LI	选项名：/LI[NENUMBERS]。增加源文件的行号以及与 MAP 文件关联的地址信息。目标文件创建时必须带有行号信息。即使未指定 MAP 文件名，该选项也会创建一个 MAP 文件
/M	选项名：/M[AP]。在 MAP 文件中加入公共符号
/NOD[:libraryname]	选项名：/NOD[EFAULTLIBRARYSEARCH]。忽略指定的库，如果未指定 *libraryname*，忽略所有的默认库
/NOE	选项名：/NOE[XTDICTIONARY]。禁止 LINK 在库中查找扩展字典。当符号重定义导致错误 L2044 时使用该选项
/NOF	选项名：/NOF[ARCALLTRANSLATION]。关闭远程调用优化
/NOI	选项名：/NOI[GNORECASE]。保留标识符的大小写
/NOL	选项名：/NOL[OGO]。禁止显示 LINK 的版权信息
/NON	选项名：/NON[ULLSDOSSEG]。以与/DOSSEG 相同的顺序组织段，但是在_TEXT 段（如果定义了）的开始没有附加的字节。该选项覆盖/DOSSEG 选项
/NOP	选项名：/NOP[ACKCODE]。关闭代码段压缩选项
/PACKC[:*number*]	选项名：/PACKC[ODE]。将相邻的代码段压缩在一起，可指定使用/PACKC 组合段的物理尺寸最大为 *number* 字节
/PACKD[:*number*]	选项名：/PACKD[ATA]。将相邻的数据段压缩在一起，可指定使用/PACKD 组合段的物理尺寸最大为 *number* 字节。该选项仅用于 Windows
/PAU	选项名：/PAU[SE]。LINK 任务中换盘暂停
/PM:type	选项名：/PM[TYPE]。指定基于 Windows 应用程序的类型，type 可以是下列值：PM（或 WINDOWAPI），VIO（或 WINDOWCOMPAT），NOVIO（NOTWINDOWCOMPAT）

（续表）

选项	含义
/ST:*number*	选项名：/ST[ACK]。将堆栈大小设为 *number* 字节，从 1~64 KB
/T	选项名：/T[INY]。创建微型内存模式的 MS-DOS 程序，扩展名为.COM 而不是.EXE。同/INCR 选项是不相容的
/?	选项名：/?。显示 LINK 的命令行格式摘要

环境变量：

变量	描述
INIT	指定 TOOLS.INI 文件的路径
LIB	指定库文件的搜索路径
LINK	指定默认的命令行选项
TMP	指定 VM.TMP 文件的路径

D.6　CodeView 调试器

Microsoft 的 CodeView 调试器在运行程序的同时显示程序源代码、程序变量、内存位置、处理器寄存器以及其他相关信息。它的使用格式为：

CV [*options*] 可执行文件名 [*arguments*]

表 D.3 列出了运行于 MS-DOS 下版本的 CodeView 的命令行选项。

表 D.3　CodeView 的命令行选项

选项	含义
/2	允许使用双监视器
/25	25 行模式
/43	43 行模式
/50	50 行模式
/B	黑白模式
/C*commands*	启动时执行 commands 指定的命令
/F	通过改变视频页切换屏幕
/G	消除 CGA 监视器上的刷新雪花
/I[0|1]	打开（I1）或关闭（I0）不可屏蔽硬件中断和 8259 中断陷阱
/K	不允许为正被调试的程序安装键盘监视
/M	不允许 CodeView 使用鼠标。在调试 Windows 3.x 下支持鼠标的应用程序时使用该选项
/N[0|1]	N0 通知 CodeView 跟踪不可屏蔽硬件中断。N1 通知调试器不要跟踪
/R	允许 80386/486 调试寄存器
/S	通过改变缓冲区切换屏幕（主要用于图形程序）
/TSF	指定 TOOLS.INI 的某项读取/不读取 CURRENT.STS 文件

D.7 MASM 伪指令

name = expression

将表达式 *expression* 的值赋给 *name*。这种符号后面可重复定义。

.186

允许汇编 80186 处理器指令，禁止其后处理器引入的汇编指令。也允许汇编 8087 指令。

.286

允许汇编 80286 处理器非特权指令，禁止其后处理器引入的汇编指令。也允许汇编 80287 指令。

.286P

允许汇编80286处理器所有指令（包括特权指令），禁止其后处理器引入的汇编指令。也允许汇编 8087 指令。

.287

允许汇编 80287 协处理器指令，禁止其后协处理器引入的新处理器。

.386

允许汇编 80386 处理器非特权指令，禁止其后处理器引入的汇编指令。也允许汇编 80387 指令。

.386P

允许汇编 80386 处理器所有指令（包括特权指令），禁止其后处理器引入的汇编指令。也允许汇编 8087 指令。

.387

允许汇编 80387 协处理指令。

.486

允许汇编 80486 处理器非特权指令。

.486P

允许汇编 80486 处理器的所有指令（包括特权指令）。

.586

允许汇编奔腾处理器的非特权指令。

.586P

允许汇编奔腾处理器的所有指令（包括特权指令）。

.686

允许汇编奔腾 Pro 处理器的非特权指令。

.686P

允许汇编奔腾 Pro 处理器的所有指令（包括特权指令）。

.8086

允许汇编 8086 指令（以及等价的 8088 指令），禁止汇编其后处理器引入的指令。也允许汇编 8087 指令，这是处理器的默认模式。

.8087

允许汇编 8087 指令，禁止汇编其后处理器引入的指令。这是协处理器的默认模式。

ALIAS <alias>=<actual-name>

将旧函数名映射为新名字。Alias 是过程或函数的别名，actual-name 是实际名字。ALIAS 可用于创建允许连接器（LINK）将旧函数映射为新函数的库。

ALIGN [*number*]

将下一个变量或下一条指令按 *number* 对齐。

.ALPHA

按字母顺序将段排序。

ASSUME *segregister*:*name* [, *segregister*:*name*]…

ASSUME *dataregister*:*type* [,*dataregister*:*type*]…

ASSUME *register*:**ERROR** [, *register*:**ERROR**]…

ASSUME [*register*:] **nothing** [, *register*:**nothing**]…

允许对寄存器值进行错误检查。在ASSUME生效之后，汇编器监视给定寄存器值的改变。如果寄存器被使用，ERROR 会产生错误。NOTHING 消除寄存器的错误检查。可在一条语句中结合不同类型的 ASSUME。

.BREAK [**.IF** *condition*]

如果 condition 为真，则退出.WHILE 或.REPEAT 块。

[*name*]**BYTE** *initializer* [, *initializer*]

分配 1 字节存储空间并可选地为每个字节指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。

name **CATSTR** [*textitem1* [,*textitem2*]…]

链接文本。每个文本项都可以是一个字符串，以%开头的常量或宏函数返回的字符串。

.CODE [[*name*]]

在和.MODEL 伪指令联用的时候，表示名字是 *name* 的代码段的开始（默认情况下 tiny，small，compact 和 flat 模式的代码段名是_TEXT，其他模式下段名是 *module*_TEXT）。

COMM *definition* [,*definition*]…

定义公共变量，每个变量及其属性在 *definition* 中定义，每个定义的格式如下：

[*langtype*] [**NEAR**|**FAR**] *label*:*type* [:*count*]

label 是变量的名字。*type* 可以是任何类型关键字（BYTE，WORD 等）或一个指定字节数的整数。*count* 指定数据对象的数量（默认为 1）。

COMMENT *delimiter* [[*text*]]

[*text*]

[*text*] *delimiter* [*text*]

将分隔符之间或同一行上的文本作为注释对待。

.CONST

在和.MODEL 伪指令联用时，表示常量数据段的开始（段名是 CONST）。CONST段的属性是只读的。

.CONTINUE [[**.IF** *condition*]]

如果条件 *condition* 为真，跳转到.WHILE 或.REPEAT 块的开始。

.CREF

允许在符号表的符号部分和浏览文件中列出符号的交叉引用。

.DATA

在和.MODEL 伪指令联用时，定义已初始化数据段的开始（段名是_DATA）。

.DATA?
在和.MODEL 伪指令联用时，定义未初始化数据段的开始（段名是_BSS）。

.DOSSEG
根据 MS-DOS 的段约定组织段的顺序：代码段最先，然后是不在 DGROUP 段组中的段，最后是 DOROUP 段组中的段。DGROUP 段组中的段顺序如下：首先是非 BSS 和非 STACK 段，然后是 BSS 段，最后才是 STACK 段。主要用于确保 CodeView 支持 MASM 单独编译的程序。同 DOSSEG。

DOSSEG
同.DOSSEG，.DOSSEG 是推荐格式。

DB
用于和 BYTE 一样地定义数据。

DD
用于和 DWORD 一样地定义数据。

DF
用于和 FWORD 一样地定义数据。

DQ
用于和 QWORD 一样地定义数据。

DT
用于和 TBYTE 一样地定义数据。

DW
用于和 WORD 一样地定义数据。

[*name*]DWORD *initializer* [, *initializer*] …
分配双字存储并可选地为每个双字指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。

ECHO message
在标准输出设备（默认是屏幕）上显示消息 *message*。同%OUT。

.ELSE
参见.IF

ELSE
标记条件块中可选块的开始。参见 IF。

ELSEIF
将 ELSE 和 IF 合并成一条伪指令。参见 IF。

ELSEIF2
如果 OPTION：SETIF2 为真，每趟汇编时 ELSEIF 块都会被求值。

END [*address*]
标记模块的结束，可选地还可以指定程序的入口点为 *address*。

.ENDIF
参见.IF。

ENDIF
参见 IF。

ENDM
结束宏或重复块。参见 MACRO，FOR，FORC，REPEAT 或 WHILE。

***name* ENDP**
标记前面以 PROC 开始的过程 *name* 的结束。参见 PROC。

***name* ENDS**
标记前面以 SEGMENT，STRUCT，UNION 开始的段、结构或联合 *name* 的结束。

.ENDW
参见.WHILE。

name* EQU *expression
将 *expression* 的值赋给 *name*。*name* 在后面不能再重复定义。

***name* EQU <*text*>**
将特定的文本 *text* 赋给 *name*。*name* 后面仍可以赋不同的文本值。参见 TEXTEQU。

.ERR [*message*]
产生一个错误信息。

.ERR2 [*message*]
如果 OPTION:SETIF2 为真，.ERR 块每趟汇编时都会被求值。

.ERRB <*textitem*> [,*message*]
如果 *textitem* 为空则产生一个错误。

.ERRDEF *name* [,*message*]
如果 *name* 是已定义的标号、变量或符号就产生一个错误。

.ERRDIF[I]<*textitem1*>,<*textitem2*> [,*message*]
如果 *textitem1* 和 *textitem2* 不相等就产生一个错误。如果使用了 I，比较是大小写不敏感的。

.ERRE *expression* [,*message*]
如果 *expression* 为假（0）则产生一个错误。

.ERRIDN[I]<*textitem1*>,<*textitem2*> [,*message*]
如果 *textitem1* 和 *textitem2* 相等就产生一个错误。如果使用了 I，比较是大小写不敏感的。

.ERRNB <*textitem*> [,*message*]
如果 *textitem* 非空则产生一个错误。

.ERRNDEF *name* [,*message*]
如果 *name* 是未定义的标号、变量或符号就产生一个错误。

.ERRNZ *expression*，[*message*]
如果 *expression* 为真（非 0）则产生一个错误。

EVEN
以偶数字节边界对齐下一个变量或下一条指令。

.EXIT [*expression*]
生成结束码。可以选择向外壳程序返回 *expression* 的值。

EXITM [*textitem*]
结束当前的重复块或宏块的展开，开始汇编块外的指令。在宏函数中，*textitem* 是返回值。

EXTERN [*langtype*] ***name:type*** [, *langtype*] ***name:type*** …]

定义一个或多个名字为 *name*，类型为 *type* 的外部变量、标号或符号。类型 *type* 可以是 ABS，将 *name* 作为常量引入。同 EXTRN。

EXTERNDEF [*langtype*] ***name:type*** [, *langtype*] ***name:type*** …]

定义一个或多个名字为 *name*，类型为 *type* 的外部变量、标号或符号。

如果 *name* 在模块中定义了，就被作为 PUBLIC 符号对待；如果 *name* 在模块中被引用，被作为外部符号对待；如果名字未被引用，就被忽略掉。类型 *type* 可以是 ABS，把 *name* 作为常量引入。通常用于包含文件中。

EXTRN

参见 EXTERN。

.FARDATA [***name***]

在和**.MODEL** 伪指令联用的时候，定义初始化远程数据段的开始（段名是 FAR_DATA 或 *name*）。

.FARDATA?[***name***]

在和.MODEL 伪指令联用的时候，定义未初始化远程数据段的开始（段名是 FAR_BSS 或 *name*）。

FOR parameter [:REQ|:=default], < argument [,argument] …>

statements

ENDM

标记一个要为每个参数 *argument* 重复一次的语句块，每次重复的时候以当前 *argument* 替换 *parameter*。同 IRP。

FORC

parameter,<string>statements

ENDM

标记一个要为 *string* 中的每个字符重复一次的语句块，每次重复的时候以 *string* 中的当前字符替换 *parameter*。同 IRPC。

[*name*]FWORD *initializer* [, *initializer*] …

分配 6 字节存储并可选地为每个 FWORD 指定初始值 *initializer*。在可以使用类型的地方也可以用作类型关键字。

GOTO *macrolabel*

将汇编控制转移到:*macrolabel* 标记的行。GOTO 只允许出现在 MACRO，FOR，FORC，REPEAT 和 WHILE 块中。标号必须单独成行，而且必须以冒号开头。

***name* GROUP *segment* [, *segment*]**

将段 *segment* 添加到名为 *name* 的段组中。在 32 位平坦内存模式先使用该伪指令的时候无效果，如果命令行指定了/coff 选项时会产生一个错误。

.IF *condition1*

statements

[**.ELSEIF** condition2

statements]

[.ELSE

statements]

.ENDIF

生成代码测试条件 *condition1*（如 AX > 7），如果条件为真执行语句 *statements*。如果最后跟.ELSE 块，那么前面的条件都为假的情况下执行.ELSE 后的语句。注意条件是在运行时计算的。

IF expression 1

ifstatements

[**ELSEIF** *expression2*

elseifstatements]

[**ELSE**

elsestatements]

ENDIF

如果 *expression*1 为真(非零)或 *expression*1 为假(0),*expression2* 为真则确保汇编 *ifstatements* 或 *elseifstatements* 语句块。下面的伪指令可以代替 ELSEIF：ELSEIFB，ELSEIFDEF，ELSEIFDIF，ELSEIFDIFI，ELSEIFE，ELSEIFIDN，ELSEIFIDNI，ELSEIFNB，ELSEIFNDEF。如果前面的表达式都为假，则汇编 *elsestatements* 语句块。注意表达式是在汇编时求值的。

IF2 ***expression***

如果 OPTION:SETIF2 为真的话每趟汇编会对 IF 块进行求值。完整的格式参见 IF。

IFB ***textitem***

如果 *textitem* 为空则确保汇编其后的语句块。

IFDEF ***name***

如果 *name* 是前面已经定义的标号、变量或符号时确保汇编其后的语句块。完整格式参见 IF。

IFDIF[I] ***textitem1, textitem2***

如果 *textitem1* 和 *textitem2* 不同则确保汇编其后的语句块。如果使用了 I，比较是大小写不敏感的。完整格式参见 IF。

IFE ***expression***

如果 *expression* 为假，确保汇编其后的语句块。完整格式参见 IF。

IFIDN[I] ***textitem1, textitem2***

如果 *textitem1* 和 *textitem2* 相同则确保汇编其后的语句块。如果使用了 I，比较是大小写不敏感的。完整格式参见 IF。

IFNB ***textitem***

如果 *textitem* 非空则确保汇编其后的语句块。完整格式参见 IF。

IFNDEF ***name***

如果 *name* 是未定义的标号、变量或符号时确保汇编其后的语句块。完整格式参见 IF。

INCLUDE ***filename***

在汇编期间，在当前源文件中插入 *filename* 文件的源代码。如果 *filename* 包含了反斜线、分号、大于号、小于号、单引号或双引号时必须用尖括号括起。

INCLUDELIB *libraryname*

通知汇编器当前的模块应该和库 *libraryname* 相链接。如果 *libraryname* 包含了反斜线、分号、大于号、小于号、单引号或双引号时必须用尖括号括起。

name* INSTR [[*position*]], *textitem1*, *textitem2

查找 *textitem2* 在 *textitem1* 中首次出现的位置。开始位置 *position* 是可选的。每个文本项都可以是一个字符串、以%开头的常量或宏函数返回的字符串。

INVOKE *expression* [, *arguments*]

调用 *expression* 指定的地址处的过程，根据语言类型的标志调用约定在堆栈上或通过寄存器传递参数。每个传递给过程的参数都可以是一个表达式、一个寄存器或一个地址表达式（前面跟 ADDR）。

IRP

同 FOR。

IRPC

同 FORC。

name* LABEL *type

创建一个标号 *name*，赋以它当前地址指针计数器的值以及类型 *type*。

***name* LABEL [NEAR|FAR|PROC] PTR [*type*]**

创建一个标号 *name*，赋以它当前地址指针计数器的值以及类型 *type*。

.K3D

允许汇编 K3D 指令。

.LALL

参见 LISTMACROALL。

.LFCOND

参见 LISTIF。

.LISTALL

列出所有的语句。等价于.LIST，.LISTIF 和 LISTAMCROALL 的组合。

.LISTIF

列出假条件块内的所有语句。同.LFCOND。

.LISTMACRO

列出生成代码和数据的宏展开语句。这是默认的，同.XALL。

.LISTMACROALL

列出宏内的所有语句。同.LALL。

LOCAL *localname*[, *localname*]

在宏内部定义对每个宏实例而言都是惟一的标号。

LOCAL *label* [[*count*]] [:type] [, label [[count]] [:type]] …

在过程定义中（PROC）创建在过程生存期内基于堆栈的变量。*label* 可以是一个简单的变量或包含 *count* 个元素的数组。

***name* MACRO [parameter [:REQ|:=default|:VARARG]] …**

statements

ENDM [*value*]

定义名为 *name* 的宏语句块，并为调用宏时传递的参数创建名为 *parameter* 的容器。宏函数向调用语句返回 *value*。

.MMX

允许汇编 MMX 指令。

.MODEL *memorymodel* [, *langtype*] [, *stackoption*]

初始化程序的内存模式。内存模式 *memorymodel* 可以是 TINY，SMALL，COMPACT，MEDIUM，LARGE，HUGE 或 FLAT。*Langtype* 可以是 C，BASIC，FORTRAN，PASCAL，SYSCALL 或 STDCALL。*stackoption* 可以是 NEARSTACK 或 FARSTACK。

NAME *modulename*

忽略。

.NO87

禁止汇编所有的浮点指令。

.NOCREF [*name* [,name]…]

禁止在符号表和浏览文件中列出符号。如果指定了符号名 *name*，那么只有给定名字的符号名被禁止。同.XCREF。

.NOLIST

禁止其后代码列在程序列表中。同.XLIST

.NOLISTIF

禁止列出条件计算为假的条件块。这是默认的，同.SFCOND。

.NOLISTMACRO

禁止列出宏扩展。同.SALL。

OPTION *optionlist*

允许或禁止汇编器的某些特性。可用的选项包括：CASEMAP，DOTNAME，NODOTNAME，EMULATOR，NOEMULATOR，EPILOGUE，EXPR16，EXPR32，LANGUAGE，LJMP，NOLJMP，M510，NOM510，NOKEYWORD，NOSIGNEXTEND，OFFSET，OLDMACROS，NOOLDMACROS，OLDSTRUCTS，NOOLDSTRUCTS，PROC，PROLOGUE，READONLY，NOREADONLY，SCOPED，NOSCOPED，SEGMENT，SETIF2。

ORG *expression*

将地址指针计数器设为 *expression* 的值。

%OUT

同 ECHO。

[*name*]OWORD *initializer* [, *initializer*]…

分配 8 字（16 字节）存储并可选地为每个 OWORD 变量指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。该伪指令主要用于 SIMD 指令，它容纳了 4 个 4 字节实数。

PAGE [[length], width]

设置列表文件每页的行数和字符宽度。如果不指定参数，就生成一个分页符。

PAGE+

增加节号并将页号设为 1。

POPCONTEXT *context*

恢复全部或部分当前上下文（由 PUSHCONTEXT 伪指令保存）。*context* 可以是 ASSUMES，RADIX，LISTING，CPU 或 ALL。

label PROC [*distance*] [*langtype*] [*visibility*] [*<prologuearg>*]

[**USES** *reglist*] [*parameter* [*:tag*]] ···

statements

label **ENDP**

定义名为 label 的过程的开始和结束。块内的语句可以使用 CALL 指令或 INVOKE 指令调用。

***label* PROTO** [*distance*] [*langtype*] [, [*parameter*]*:tag*] ···

声明过程原型。

PUBLIC [*langtype*] *name* [,[*langtype*]*name*] ···

使每个名为 name 的标号、变量和绝对符号对程序中的其他模块可用。

PURGE *macroname* [, *macroname*] ···

删除指定的宏定义。

PUSHCONTEXT *context*

保存当前上下文的全部或部分内容：段寄存器名、基数值、列表和交叉引用标志或处理器/协处理器的值。*context* 可以是 ASSUMES，RADIX，LISTING，CPU 或 ALL。

[*name*]**QWORD** *initializer* [, *initializer*] ···

分配 8 字节存储并可选地为每个 QWORD 变量指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。

.RADIX *expression*

设置表达式默认的基数值，范围从 2~16。

***name* REAL4** *initializer* [, *initializer*] ···

为单精度（4 字节）浮点数分配存储并可选地为每个单精度变量指定初始值 *initializer*。

***name* REAL8** *initializer* [, *initializer*] ···

为双精度（8 字节）浮点数分配存储并可选地为每个双精度变量指定初始值 *initializer*。

***name* REAL10** *initializer* [, *initializer*] ···

为 10 字节的浮点数分配存储并可选地为每个变量指定初始值 *initializer*。

***recordname* RECORD** *fieldname*: *width* [= expression] [*fieldname:width* [= *expression*]] ···

声明一个由指定域构成的记录。*filedname* 指定域的名字，*width* 指定数据位的数目，*expression* 给出它的初始值。

.REPEAT

statements

.UNTIL *condition*

生成代码重复执行 *statements* 语句块，直到条件 *condition* 变为真。.UNTILCXZ 当CX 为 0 时为真，可由.UNTIL 代替。.UNTILCXZ 中条件 *condition* 是可选的。

REPEAT *expression*

statements

ENDM

标记一个要重复 *expression* 次的语句块。同 REPT。

REPT

参见 REPEAT。

.SALL

参见.NOLISTMACRO。

***name* SBYTE *initializer* [, *initializer*] …**

为有符号字节分配存储，可选为其指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。

***name* SDWORD *initializer* [, *initializer*] …**

为有符号双字(4字节)分配存储，可选为其指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。

***name* SEGMENT [READONLY] [align] [combine] [use] ['class']**

statemnets

name **ENDS**

定义名为 *name* 的段，段属性有对齐 *align*（BYTE，WORD，DWORD，PARA，PAGE），组合方式 *combine*（PUBLIC，STACK，COMMON，MEMORY，AT *address*，PRIVATE）和模式 *use*（USE16，USE32 和 FLAT），以及类别 *class*。

.SEQ

按顺序(默认顺序)排列段。

.SFCOND

参见.NOLISTIF。

name* SIZESTR *textitem

获取文本项的大小。

.STACK [*size*]

在和.MODEL 联合使用的时候，定义一个堆栈段（段名为 STACK）。可选的 *size* 指定堆栈的字节数(默认是 1024)。.STACK 伪指令自动结束堆栈段。

.STARTUP

生成程序启动代码。

STRUC

参见 STRUCT。

***name* STRUCT [*alignment*][,NONUNIQUE]**

fielddeclarations

name **ENDS**

声明一个结构类弄，每个域必须是一个有效的数据定义，同 STRUC。

***name* SUBSTR *textitem,position*, [, *length*]**

返回 *textitem* 从 *position* 开始的子串。*textitem* 可以是字符串、以%开始的常量或宏函数返回的字符串。

SUBTITLE ***text***

定义列表文件的子标题。同 SUBTTL。

SUBTTL

参见 SUBTITLE。

name **SWORD** ***initializer*** **[,** ***initializer*****]** **…**

为有符号字(2 字节)分配存储，可选为其指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。

[*name*]TBYTE ***initializer*** **[,** ***initializer*****]** **…**

分配 10 字节存储并可选地为其指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。

name **TEXTEQU [*textitem*]**

将文本 *textitem* 赋给 *name*。*textitem* 可以是字符串、以%开始的常量或宏函数返回的字符串。

.TFCOND

列出计算为假的条件块。

TITLE ***text***

定义程序列表文件的标题。

name **TYPEDEF** ***type***

定义一个与类型 *type* 等价的名为 *name* 的新类型。

name **UNION[*alignment*] [, NONUNIQUE]**

fielddeclarations

[*name*] ENDS

说明一个或多个数据类型的联合。*fielddeclarations* 必须是有效的数据定义。在嵌套 UNION 定义中要忽略 ENDS 前面的名字 *name*。

.UNTIL

参见.REPEAT。

.UNTILCXZ

参见.REPEAT。

.WHILE ***condition***

statements

.ENDW

生成代码在 *condition* 条件为真时重复执行 *statements* 语句块。

WHILE ***expression***

statements

ENDW

在 *expression* 为真时重复 *statements* 汇编语句块。

[*name*]WORD initializer[, ***initializer*****]** **…**

分配字存储并可选地为其指定初始值 *initializer*。在可以使用类型的地方也可以用做类型关键字。

.XALL

参见.LISTMACRO。

.XCREF
参见.NOCREF。

.XLIST
参见.NOLIST。

.XMM
允许汇编 SIMD 扩展指令。

D.8 预定义符号

$
地址指针的当前值。

?
在数据声明中，汇编器分配存储但并不初始化。

@@:
定义只能在 *label1* 和 *label2* 之间识别的代码标号，其中 *label1* 是代码的开始或前一个@@:标号，*label2* 是代码的结尾或下一个@@:标号。见@B 和@F。

@B
前一个@@：标号的地址。

@CatStr *string1* [, *string2* …]
链接一个或多个字符串的宏函数。返回一个字符串。

@code
代码段的名称（文本宏）。

@CodeSize
TINY，SMALL，COMPACT 和 FLAT 模式下该值为 0，MEDIUM，LARGE 和 HUGE 模式该值为 1。

@Cpu
指定处理器模式的位掩码。

@CurSeg
当前段的名字（文本宏）。

@data
默认数据段组的名字。除了 FLAT 模式之外，其余所有模式下该值都等于 DGROUP。FLAT 模式下该值等于 FLAT。

@DataSize
TINY，SMALL，COMPACT 和 FLAT 模式下该值为 0，MEDIUM，LARGE 模式该值为 1，HUGE 模式下该值为 2。

@Date
系统时间，格式是 mm/dd/yy（文本宏）。

@Environ（envvar）
环境变量 *envvar* 的值（宏函数）。

@F

下一个@@:标号的地址。

@ fardata

.FARDATA 伪指令定义的段名（文本宏）。

@fardata?

.FARDATA? 伪指令定义的段名（文本宏）。

@FileCur

当前文件的名字。

@FileName

正被汇编文件的基本名。

@InStr ([*position*], *string1*, *string2*)

找出 *string2* 在 *string1* 中从 *position* 开始首次出现位置的宏函数，如果没有 *position*，从 *string1* 的开始查找，如果未找到 *string2* 返回 0。

@Interface

关于语言参数的信息。

@Line

当前文件中的源代码行号。

@Model

TINY 模式该值为 1，SAMLL 模式该值为 2，COMPACT 模式该值为 3，MEDIUM 模式该值为 4，LARGE 模式该值为 5，HUGE 模式该值为 6，FLAT 模式该值为 7。

@SizeStr(*string*)

返回给定字符串长度的宏函数。返回一个整数。

@stack

近堆栈该值是 DGROUP，远堆栈该值是 STACK（文本宏）。

@SubStr(*string*, *position*,[, *length*])

返回 string 从 *position* 位置开始的子串。

@Time

24 小时格式的系统时间，格式是 hh:mm:ss（文本宏）。

@Version

版本号，MASM 6.1 该值为 610（文本宏）。

@WordSize

16 位段该值为 2，32 位段该值为 4。

D.9 操作符

expression1* + *expression2

expression1 加 *expression2*。

expression1*- *expression2

expression1 减 *expression2*。

expression1 * expression2
expression1 乘以 *expression2*。

expression1 / expression2
expression1 除以 *expression2*。

-expression
expression 符号取反。

expression1[expression2]
expression1 加 *expression2*。

segment:expression
以 *segment* 覆盖 *expression* 的默认段。*segment* 可以是段寄存器，组名，段名或段表达式。*expression* 必须是常量。

expression.field[.filed] ···
返回 *expression* 的地址值加上结构或联合内 *field* 偏移的和。

[register].field[.filed] ···
返回由 *register* 指向的地址值加上结构或联合内 *field* 偏移的和。

<text>
将 *text* 作为文本对待。

"text"
将 *text* 作为字符串对待。

'text'
将 *text* 作为字符串对待。

!character
将 *character* 作为字符而不是操作符或符号对待。

;text
将 *text* 作为注释对待。

;;text
text 只在宏内以注释的形式出现。列表文件中在宏展开的时候并不显示 *text*。

% expression
将宏内的 *expression* 值作为文本对待。

&parameter&
以对应的参数值替换 *parameter*。

ABS
参见 EXTERNDEF 伪指令。

ADDR
参见 INVOKE 伪指令。

expression1* AND *expression2
返回 *expression1* 和 *expression2* 进行位与操作的结果。

***count* DUP(*initialvalue*[,initialvalue] ···)**
指定 *count* 数量的 *intialvalue* 的声明。

expression1* EQ *expression2
如果 *expression1* 等于 *expression2* 返回真（-1），否则返回假（0）。

expression1* GE *expression2
如果 *expression1* 大于等于 *expression2* 返回真（-1），否则返回假（0）。

expression1* GT *expression2
如果 *expression1* 大于 *expression2* 返回真（-1），否则返回假（0）。

HIGH *expression*
返回 *expression* 的高字节。

HIGHWORD *expression*
返回 *expression* 的高字。

expression1* LE *expression2
如果 *expression1* 小于等于 *expression2* 返回真（-1），否则返回假（0）。

LENGTH *variable*
返回 *variable* 内初始化时创建的数据项的数目。

LENGTHOF *variable*
返回 *variable* 中的数据对象数目。

LOW *expression*
返回 *expression* 的低字节。

LOWWORD *expression*
返回 *expression* 的低字。

LROFFSET *expression*
返回 *expression* 的偏移。同 OFFSET，不过它生成一个加载器解决的偏移，以允许重定位代码段。

expression1* LT *expression2
如果 *expression1* 小于 *expression2* 返回真（-1），否则返回假（0）。

MASK | ***recordfiledname*|*record*** |
返回一个掩码，其中 *recordfiledname* 或 *record* 对应的位设置而其他位清除。

expression1* MOD *expression2
返回 *expression1* 除以 *expression2* 时的余数值（模）。

expression1* NE *expression2
如果 *expression1* 不等于 *expression2* 返回真（-1），否则返回假（0）。

NOT *expression*
expression 的所有位变反。

OFFSET *expression*
返回 *expression* 的偏移值。

OPATTR *expression*
返回一个定义 *expression* 模式和范围的字。低字节等于.TYPE 返回的字节，高字节包含了其它信息。

expression1* OR *expression2
返回 *expression1* 和 *expression2* 进行位或运算的结果。

type* PTR *expression
强制表达式被按照具有 *type* 类型对待。

[*distance*] PTR *type*
指定一个指针的类型为 *type*。

SEG *expression*
返回 *expression* 所在的段。

expression* SHL *count
返回表达式 *expression* 左移 *count* 个位的结果。

SHORT *label*
将 *label* 的类型设为短类型。所有跳转到 *label* 的跳转都必须是近跳转(从跳转指令到 *label* 的距离在–128 到+127 字节范围之内)。

expression* SHR *count
返回表达式 *expression* 右移 *count* 个位的结果。

SIZE *variable*
返回初始化时为 *variable* 分配的字节数。

SIZEOF {*variable*|*type*}
返回初始化时 *variable* 或 *type* 的字节数。

THIS *type*
返回一个指定的 *type* 类型的标号，其偏移和段值等于当前地址计数器的值。

.TYPE *expression*
参见 OPATTR。

TYPE *expression*
返回 *expression* 的类型。

WIDTH { *recordfiledname*|*record* }
返回当前 *recordfiledname* 或 *record* 内数据位的数目。

expression1* XOR *expression2
返回 *expression1* 和 *expression2* 进行位异或运算的结果。

D.10 运行时操作符

下面的操作符仅用于.IF，.WHILE 或.REPEAT 块中并在运行时进行求值，而不是在汇编时求值的。

expression1* == *expression2
等于。

expression1* != *expression2
不等于。

expression1* > *expression2
大于。

expression1* >= *expression2
大于等于。

expression1* < *expression2
小于。

expression1* <= *expression2
小于等于。

expression1* || *expression2
逻辑或。

expression1 && expression2
逻辑与。

expression1* & *expression2
位 AND。

!*expression*
逻辑非。

CARRY?
进位标志的值。

OVERFLOW?
溢出标志的值。

PARITY?
奇偶标志的值。

SIGN?
符号标志的值。

ZERO?
零标志的值。