ASM386

ASM386

ASM386

汇编语言

ASM386HUIBIANYUYAN

李宝山 主编

· 中国铁道出版社

ASM386 汇编语言

李宝山　主编

中　国　铁　道　出　版　社
1989年·北京

内 容 简 介

本书主要说明 ASM386 汇编语言的语法和功能。ASM386 是用于美国英特尔公司最新推出的 80386 微处理机的汇编语言。全书共分 8 章：ASM386 简介、段、指令定义和初始化数据、存取数据（操作数和表达式）、程序链接命令、80386 指令集、浮点指令、宏和宏功能等。除此之外，还在附录中介绍了代码宏、80386 标志、保留字、同高级语言模块链接、程序举例、ASM386 和 ASM286 之间的不同、ASM386 的构成元素、ASCII 表等内容。

本书可供应用计算机的工程技术人员及操作员、程序员、程序设计者等软件技术人员学习参考。

ASM386 汇编语言

李宝山 主编

中国铁道出版社出版、发行

责任编辑 魏京燕 封面设计 刘景山

各地新华书店经售

中国铁道出版社印刷厂印

开本：787×1092 毫米 $\frac{1}{16}$ 印张：15 字数：376 千

1989 年 5 月 第 1 版 第 1 次印刷

印数：1—8,000 册 定价：5.95 元

前　言

ASM386 是用于美国英特尔公司最新推出的 80386 微处理机的汇编语言。本书主要说明 ASM386 汇编语言的语法和功能，是使用 ASM386 汇编语言编制适用于 80386 微处理机汇编程序的工具书。

本书共分八章，前六章介绍 ASM386 的主要构成元素，包括段和数据定义、数据的存取、程序的链接和 80386 指令。第七章介绍浮点操作指令，这是在 80386 同 80287 协处理机一起应用时使用的部分指令。第八章介绍宏和宏功能，用于扩展汇编语言功能。

本书供有关计算机专业技术人员、特别是操作员、程序员、程序设计者等软件技术人员参考。

参加本书编写工作的还有常泽、单大明等同志。

本书编写过程中得到中国英特尔微型计算机用户协会理事长庄梓新先生的指导并审阅全书，在此深表感谢。

由于本人经验不足，水平有限，书中可能存在一些错误和缺点，恳请读者批评指正。

编者　1988 年 5 月

目　录

第一章　ASM386 简介

本章介绍 INTEL80386 汇编语言。ASM386 为 80386 微处理机指定机器指令并为程序数据分配存储器空间。ASM386 代码源文件可由 ASM386 宏汇编程序转换为机器可读的目标文件。汇编程序可产生包含程序和符号表的打印文件。可使用汇编程序调用行或源文件指定的控制调整汇编程序的输出。

一、80386 结构概述

为使用 ASM386 汇编语言进行编程，需要熟悉 80386 的结构。80386 结构的基础部分是寄存器，如图 1-1 所示。

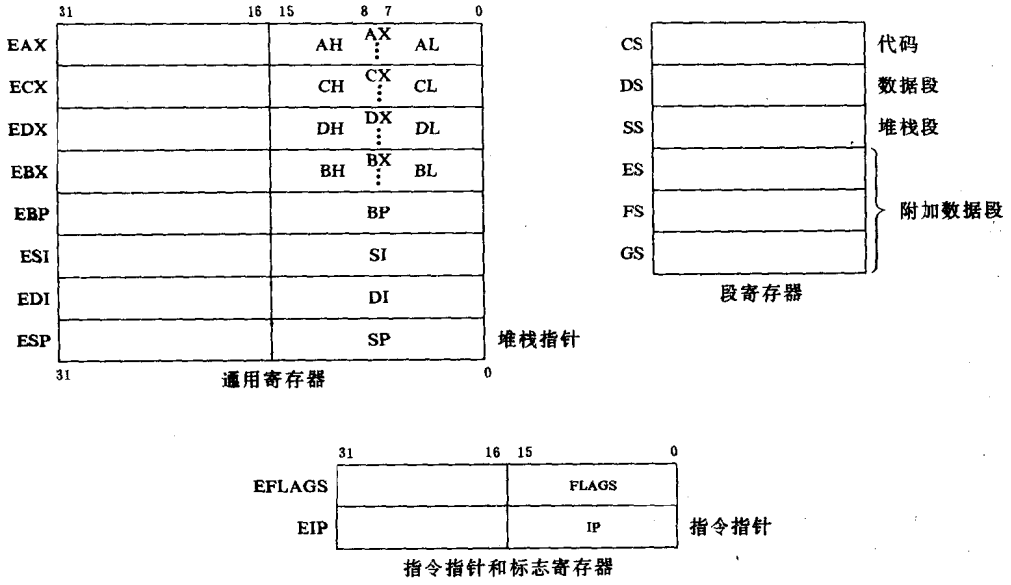

图1－1　80386结构中的寄存器

1. 寄存器

· 通用寄存器：如图 1-1 所示，80386 结构包括 8 个 32 位通用寄存器：EAX、ECX、EDX、EBX、EBP、ESI、EDI 和 ESP。每一通用寄存器的低 16 位可以单独存取，因而提供 8 个 16 位寄存器：AX、CX、DX、BX、BP、SI、DI 和 SP。有 4 个 16 位寄存器(AX、CX、DX、BX)的高位字节和低位字节可单独存取，因而提供 8 个 8 位寄存器：AH、AL、CH、CL、DH、DL、BH 和 BL。

通用寄存器用以保存中间数据。在 ASM386 源码中，它们可用作 80386 指令的操作数。

· 段寄存器：图 1-1 也示出了段寄存器：CS、DS、SS、ES、FS 和 GS，每一段寄存器间接指向调用程序段程序的基地址。每一段包含程序的不同部分：CS 指向的段包含代码；DS 指向的段

包含数据;SS 指向的段包含运行时的堆栈;ES、FS 和 GS 寄存器可指向附加数据段。

· 标志寄存器:标志寄存器 EFLAGS 包含 80386 的标志。EFLAGS 的低 16 位包含 80286 标志寄存器 FLAGS。80386 标志受 80386 指令影响而变化。

· 指令指针:80386 指令指针是 32 位的 EIP 寄存器,它包含欲执行的下一条指令的地址。EIP 的低 16 位包含 80286 指令指针 IP。

· 附加寄存器:控制寄存器和系统编址寄存器包含影响系统所有任务的机器状态。有四个控制寄存器:

CR0 保持机器状态字(MSW),它定义处理机的状态。CR0 可由具体的 80386 指令存取。

CR1 保留。

CR2 和 CR3 使 80386 具有编页功能。CR2 包含线性地址;CR3 包含指向页目录的地址。

有四个系统编址寄存器,它们包含 4 个特殊段的地址,这 4 个特殊段保存支持 80386 保护模式的描述符表。它们是:

IDTR(或 IDT),它包含中断描述符表的选择器。

GDTR(或 GDT),它包含整体描述符表的选择器。

LDTR(或 LDT),它包含局部描述符表的选择器。

TR(或 TSS),它包含任务状态段的选择器。

某些 80386 指令可以存取系统编址寄存器。

80386 结构还有一组调试寄存器(DR0～DR7)和一组测试寄存器(TR6 和 TR7)。

2. 实模式和保护模式操作

80386 有两种操作模式:实模式和保护虚拟地址模式(一般称作保护模式)。在实模式,存储器地址由指令偏移量和包含在段寄存器中的值形成。段寄存器中的值是段的基址。这种寻址方式可使处理机寻址到 4GB。

在保护模式,段寄存器不直接保存段的基址。每一寄存器保存一个选择器,选择器指向地址转换表(亦称描述符表)中的某一位置,由此定义段的基址。存储器地址由选择器和指令偏移量构成。

实模式用以执行 8086 目标码。它是处理机加电或复位时所处的模式。当处理机控制寄存器中的一位置位时开始执行保护模式,保护模式可提供虚拟寻址、多级保护、多任务和调试功能,这些功能在实模式中是没有的。

3. 段

每一个段寄存器(CS、DS、SS、ES、FS 和 GS)的内容说明一个连续的物理存储器空间——段。将存储器划分为段是 80386 存储器寻址的基础。

当处理机在实模式操作时,段寄存器的值是段的基地址。在保护模式中,每一个段寄存器保存一个称作段选择器的 16 位值。段选择器指向描述符表的某一项,描述符表驻留在存储器中。描述符表中的项包含存储器段的基地址。实模式和保护模式段的选择请参见图 1-2、图 1-3 和图 1-4。

段分别提供程序的代码、数据和堆栈部分,这有助于防止程序执行时的故障。例如,如果代码、数据、堆栈混到一起,数据可能被当作代码执行,堆栈可能被变量覆盖。80386 编程允许直接指定程序的哪一部分属于哪一个段。

· 存储器地址格式

80386 存储器地址由段机构形成。每一个地址有两部分:段选择器和偏移量。段选择器

指向段的基地址。偏移量是相对于段基址的位移量。选择器和偏移量一起给出完整的存储器地址。

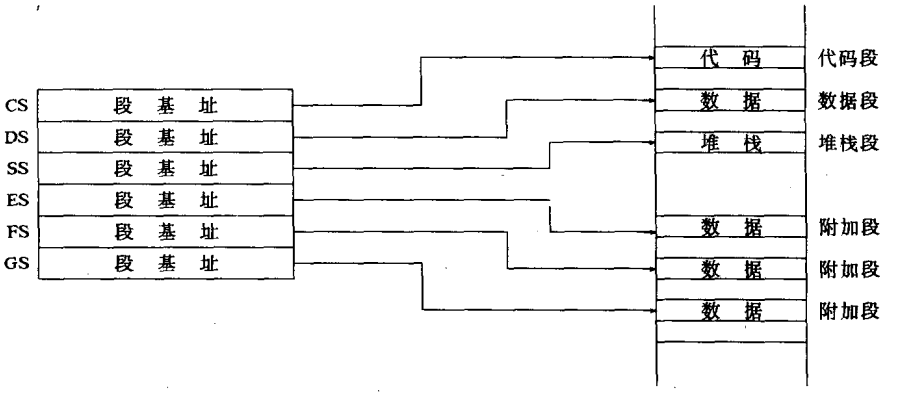

图1-2 实模式段选择

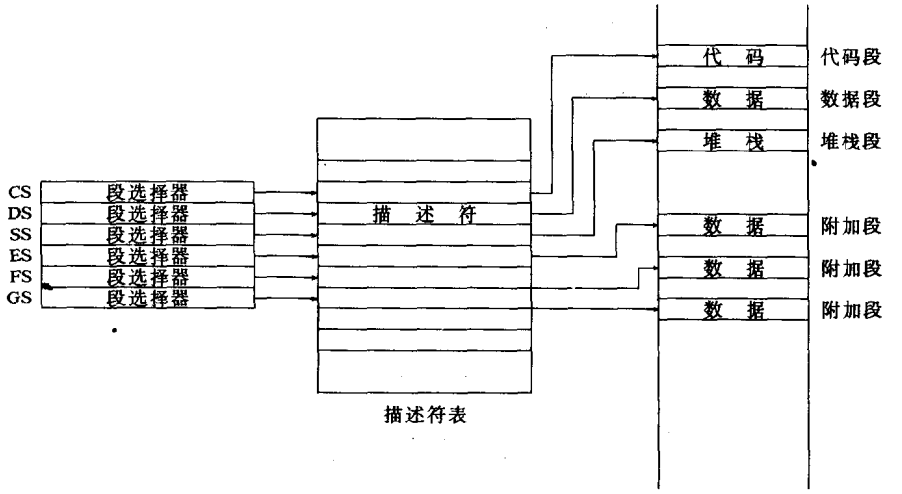

图1-3 保护模式段选择

例如，考虑变量 VAR_1 的地址。因为变量是一组数据，所以程序员将它定义为数据段的一部分。假定 VAR_1 定位在数据段开始的 6 个字节，那么它的偏移量是 6 字节。VAR_1 的基地址是数据段的基地址。假定 DS 寄存器中的选择器指向基地址，那么，速记符 DS：6 可用以说明 VAR_1 的地址，它指出了指向段基址的段寄存器(DS)和段内单元的偏移量(6 字节)。图 1-5 用 VAR_1 做例子说明基址:偏移量编址方式。

· 32 位和 16 位地址

80386 地址可由 32 位长的偏移量或 16 位长的偏移量形成。每个程序段都有一个称为 USE 属性的特性，它决定段内定义的存储器地址是用 32 位还是用 16 位偏移量形成。USE32 属性指定建立 32 位偏移量；USE16 指定建立 16 位偏移量。这些属性在第二章和第六章中讨论。

· 段的长度

程序段的长度可以变化。段的最大长度由段的 USE 属性决定。USE16 段可长达 64K 字节；USE32 段可达 4GB(2^{32} 字节)长。

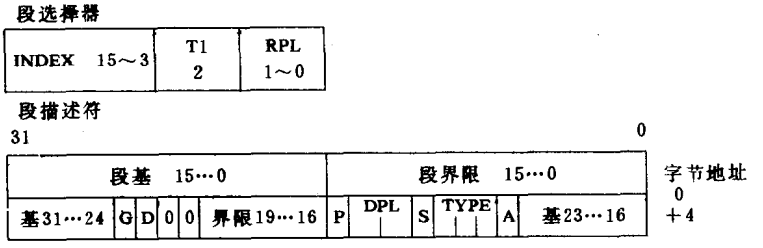

图1-4 段选择器和描述符格式

T1——表标识符，0=GDT，1=LDT；
INDEX——表中的描述符项；
P——表示位，1=表示，0=无表示；
DPL——描述符优先级0～3；
S——段描述符，0=系统描述符，1=代码或数据段描述符；
G——粒度位，1=段长度是页粒度，0=段长度是字节粒度；
D——缺省操作长度(只有代码段描述符能识别)，1=32位段，0=16位段；
0——此位必须为0，以便同未来的处理机兼容。
RPL——申请优先级(0～3)；
基——段基址；
界限——段长度；
TYPE——段类型；
A——存取位；

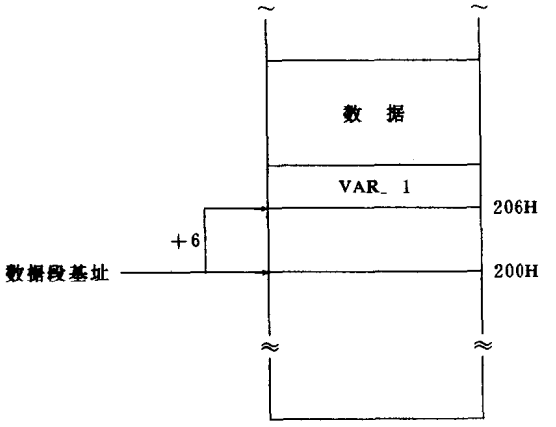

图1-5 相对于数据段基址的VAR_ 1地址

4. 页

80386 提供了一个称之为页的可选择地址结构。页只在 80386 保护模式下应用，它将处理机物理地址空间分为一系列 4K 字节页或页框架。页的存取分两步：页目录指向 1K 页表，然后每一页表可存取 4K 字节的 1K 页框架。由此，每一页目录可存取 4GB 存储器地址空间。

当 CR0 寄存器中的页位(PG)置位时允许编页。编页机构对应用程序是透明的。

二、ASM386 程序结构

ASM386 程序由段组成，段由语句组成。语句由命令、指令、操作符和操作数组成。本节说明如何将这些元素组合到一起。

1. 逻辑段

ASM386 程序的基本逻辑元素是程序段。源文件中的汇编语言语句分别定义各逻辑区

段，这些区段与段寄存器内容指向的物理段一致。每一逻辑段用段的符号名后跟字 SEGMENT 开始，并用它的符号名加字 ENDS 结束。对 SEGMENT/ENDS 定义的代码和数据来说，逻辑段与物理段在形成存储器地址方面是一致的，它们都使用同样的段基址。

图 1-6 表示一个程序代码区段，它定义了一个名为 PROG_DATA 的逻辑段。

```
PROG_DATA   SEGMENT
    VAR_1   DB 0
    VAR_2   DW 0
    VAR_3   DW 1000
PROG_DATA   ENDS
```

图 1-6　段定义举例

PROG_DATA 包含定义变量 VAR_1、VAR_2、VAR_3 的语句。因为这些变量定义在 SEGMENT/ENDS 确定的段 PROG_DATA 内，所以它们将定位在同样的物理存储器段内。它们的地址用同样的段基址，但偏移量不同。

在 ASM386 程序中，大多数语句必须放在 SEGMENT/ENDS 对之间。某些语句可以放在段块之外，但只用于特殊目的。

2. 语　　句

ASM386 程序由语句组成，语句分两类：指令语句和命令语句。指令语句包括 80386 和 80287 数字处理机指令，有时称为指令记忆符。例如，MOV、ADD、JMP 是指令记忆符。另外，指令语句可包括由用户提供的代码宏定义的指令，代码宏列在附录 A 中。命令语句包括 ASM386 命令，它是由汇编程序解释的记忆符。

指令和命令语句一般有下列格式：

指令语句

〔label：〕〔prefix〕mnemonic〔argument〔,…〕〕

命令语句

〔name〕directive argument〔,…〕

这里　label 和 name：是用户定义的标识符。

prefix：是 80386 指令前缀(LOCK 或 REP)。

mnemonic：是 80386 或 80287 指令或用户定义的代码宏。

directive：是汇编程序命令。

argument：是指定的指令或命令的修改量或操作数。

ASM386 用户定义的标识符必须用一个字母、问号(？)、符号@或记号(　)开头，剩下的字符可以是字母、数字或前面的三个字符。标识符只考虑到 31 个字符，但就其长度来说可以到 255 个字符。每一个标识符在程序模块中有全局作用域。

图 1-7 中是这两种语句的举例。这一节程序包含两个以上的逻辑段，它们是：运行时的堆栈 PROG_STACK，它由 STACKSEG 命令定义且包含 200 字节；PROG_CODE 包含建立程序代码的指令语句；PROG_DATA 包括命令 DB 和 DD。PROG_CODE 包括指令 MOV、INC、PUSH、ADD、POP、SUB 和 JNZ。80386 指令和 ASM386 命令在下一节中叙述。

另外，ASM386 程序可以包含注解，注解用分号(;)开始，还可以包含具有汇编程序控制的控制行。

```
PROG_STACK  STACKSEG  200
PROG_DATA  SEGMENT  RW
    VAR_1   DB  0
    VAR_2   DD  0
    VAR_3   DD  1000
PROG_DATA  ENDS
PROG_CODE  SEGMENT  ER
    START:MOV  ESP,STACKSTART  PROG_STACK
    MAIN:INC VAR_1
        PUSH  EAX
        MOV  EAX,VAR_2
        ADD  EAX,5
        MOV  VAR_2,EAX
        POP  EAX
        MOV  ECX,VAR_3
        SUB  ECX,VAR_2
        JNZ  MAIN
PROG_CODE  ENDS
```

图 1-7　数据段、堆栈段、代码段举例

三、指　令

80386 指令分为几类，其中包括数据传送、地址传送、算术、逻辑、堆栈、标志、数据调整、串、位、控制传递、中断、过程实施和处理机控制指令。当处理机在保护模式操作时，保护模式控制和保护参数校验指令允许保护特性被控制和被校验。

表 1-1 列出了 80386 指令，更详细的说明请参阅第六章。

80287 数字协处理机可同 80386 处理机连接使用以提供浮点计算。80287 有它自己的指令，这些指令可用同 80386 指令一样的方式在 ASM386 指令语句中使用。第七章介绍 80287 指令集。

指令操作数

Arguments 一项随具体的命令或指令而变化。对指令来说这一项通常是一个或几个操作数。在指定两个操作数的情况下，右边的操作数是源，左边的操作数是目标。例如：

MOV VAR_2,EAX

在这里，EAX 是源，VAR_2 是目标。源包含指令操作的值，目标是操作存放结果的单元。

指令操作数可以是变量名、寄存器名、数字或其组合，它同 ASM386 操作符一起形成表达式。表达式在汇编程序汇编时求值，产生的结果用作操作数。ASM386 的操作符和表达式在第四章中详述。

四、ASM386 命令

表 1-2 按下列作用分类列出了 ASM386 命令：

1. 段命令用于定义程序的逻辑段。

2. 当程序由一个以上的模块组成时，需要程序链接命令。这些命令在第五章中讨论。

3. 数据初始化命令用于定义程序变量。每一个命令指定一个实际大小和类型的变量。例如，BIT、BYTE、WORD、DWORD 是 ASM386 数据类型。这些命令在第三章中叙述。

80386 指 令 表 1-1

数据传送指令	
MOV	传送数据
MOV	到/从指定寄存器传送
MOVZX	具有 0 扩展的传送
MOVSX	具有符号扩展的传送
IN	从端口输入
OUT	向端口输出
XCHG	寄存器/存储器同寄存器交换
XLAT/XLATB	查表传送
地址传送指令	
LEA	装入有效地址偏移量
LDS	装入全指针
LES	装入全指针
LFS	装入全指针
LGS	装入全指针
LSS	装入全指针
算术指令	
ADD	加法
ADC	进位加
SUB	整数减法
SBB	借位整数减法
MUL	AL 或 AX 的无符号乘法
IMUL	符号乘
DIV	无符号除
IDIV	符号除
INC	加 1
DEC	减 1
NEG	二进制补码
CMP	比较两个操作数
逻辑指令	
NOT	二进制反码
AND	逻辑与
OR	逻辑或
XOR	逻辑异
TEST	逻辑比较
SHL	逻辑左移
SHR	逻辑右移
SAL	算术左移
SAR	算术右移
SHLD	双精度左移
SHRD	双精度右移
ROL	左旋转
ROR	右旋转
RCL	通过 CF 标志左旋转
RCR	通过 CF 标志右旋转

续上表

堆栈指令	
ENTER	为过程参数建立堆栈框架
LEAVE	高级过程退出
PUSH	将操作数压入堆栈
POP	从堆栈中弹出一个字
PUSHF/PUSHFD	将 FLAGS 或 EFLAGS 寄存器压入堆栈
POPF/POPFD	将堆栈弹出到 FLAGS 或 EFLAGS 寄存器
PUSHA/PUSHAD	将所有通用寄存器压栈
POPA/POPAD	弹栈所有通用寄存器
标志指令	
STC	设置进位标志
CLC	清除进位标志
CMC	进位标志求反
STD	设置方向标志
CLD	清除方向标志
STI	设置中断标志
CLI	清除中断标志
LAHF	将标志装入到 AH
SAHF	将 AH 存入到标志
SETcc	依条件设置字节
数据调整指令	
AAA	加法之后 ASCII 调整 AL
AAS	减法之后 ASCII 调整 AL
DAA	加法之后十进制调整 AL
DAS	减法之后十进制调整 AL
AAM	乘法之后 ASCII 调整 AX
AAD	除法之前 ASCII 调整 AX
CBW	将字节转换为字
CWD	将字转换为双字
CWDE	将字转换为扩展双字
CDQ	将双字转换为四字
串指令	
MOVS	将数据从串传送到串
CMPS	比较串操作数
SCAS	比较串数据
LODS	装串操作数
STOS	存储串数据
INS	从端口输入到串
OUTS	将串输出到端口
位指令	
BT	位测试
BTS	位测试和置位
BTR	位测试和复位
BTC	位测试和求反
BSF	位向前扫描
BSR	位反向扫描
IBTS	嵌入位串
XBTS	抽取位串

续上表

控制传送指令	
Jcc	如果条件满足则跳转
JMP	跳转
CALL	调用过程
RET	从过程返回
LOOP/LOOPcond	用 CX 计数器控制循环
中断指令	
INT	调用中断过程
INTO	溢出则调用中断过程
IRET	中断返回
IRETD	中断返回
处理机控制指令	
HLT	停机
WAIT	等待，直至 BUSY 待用
保护模式控制指令	
LGDT	装入 GDT 寄存器
LIDT	装入 IDT 寄存器
LLDT	装入 LDT 寄存器
LTR	装入任务寄存器
LMSW	装入机器状态字
ARPL	调整选择器的 RPL 区
CLTS	清除 CRO 中的 TS 标志
保护参数校验指令	
BOUND	检查阵列索引是否超限
LAR	装入存取权字节
LSL	装入段界限
VERR	校验段是否可读
VERW	校验段是否可写
无操作指令	
NOP	无操作(执行一个字节，增加指令指针)
指令前缀	
LOCK	置起 BUSLOCK 信号前缀
REP	重复串操作

4. 过程和标号定义命令用来定义过程和标号。标号是给特定程序地址的符号名。过程是给定符号名的逻辑段内的一段代码，它可由程序的其它部分调用。第三章讨论这些命令。

5. 位置计数器命令改变位置计数器，它是一个标号，它保持跟踪汇编时的位置。位置计数器和它们的命令将在第三章中讨论。

6. 符号命名和清除命令是 EQU 和 PURGE。它们将在第三章中讨论。

命令的操作数可以是名字。例如，同 SEGMENT 或 ENDS 命令一起使用的段名是一个操作数。数字数据也可用作命令操作数。

五、运算符和表达式

如前所述，指令和命令操作数可以是表达式，表达式由变量名、寄存器、数字和运算符组成。运算符作用于表达式操作数以产生指令或命令操作数使用的值。例如：

A_BYTE－1 这是一个表达式。运算符是减号（－），操作数是 A_BYTE 和 1。这个表达式

ASM386 命 令 表 1-2

段命令	
SEGMENT/ENDS	定义逻辑段。指定段的属性(类型、存取权和使用属性
STACKSEG	定义堆栈段,分配指定的字节数给运行时的堆栈
ASSUME	将数据和堆栈段寄存器的内容通知汇编程序
程序链接命令	
NAME	指定模块名字
END	模块需要的最后语句。对汇编程序模块的有效结束。可选择初始化 CS、SS、DS
PUBLIC	指定一个命名的符号是公用的,即在程序中的其它模块中也是可用的
EXTRN	指定的命名符号在其它程序模块中已被说明为是公用的
COMM	指定的命名符号既未在当前模块中定义也未在另外的模块中定义为公共符号
数据初始化命令	
DBIT	初始化位串变量(类型 BIT)
DB	初始化一字节变量(类型 BYTE)
DW	初始化二字节变量(类型 WORD)
DD	初始化四字节变量(类型 DWORD)
DP	初始化六字节变量(类型 PWORD)
DQ	初始化八字节变量(类型 QWORD)
DT	初始化十字节变量(类型 TBYTE)
RECORD	定义一个记录(一种 ASM386 位编码数据结构类型 RECORD)
STRUC	定义一个结构(一种模块数据值类型 STRUCTURE)
过程和标号定义命令	
Label_name	定义一个近标号(段内跳转标号)
PROC/ENDP	定义一个近(段内跳转/调用)或远(段间跳转/调用)过程
LABEL	定义一个已说明类型的标号
位置计数器命令	
ORG	将位置计数器设置为指定的值
EVEN	为后序代码或数据将位置计数器设置为偶数字节值
符号命名和清除命令	
EQU	为 ASM386 结构定义新的符号名
PURGE	删除指定符号的定义

可用作命令的变量。例如:

B_BYTE EQU A_BYTE−1

A_BYTE−1 是 EQU 命令的一个变量。

第四章将详细说明 ASM386 运算符。

六、初始化段寄存器

在执行 ASM386 程序之前,必须初始化段寄存器和定义运行时的堆栈。

1. 初始化 DS、ES、FS 和 GS 寄存器

变量或标号的地址由选择器和偏移量组成。选择器存放在段寄存器中。但是,在段寄存器用以形成地址之前,必须将其初始化为相应选择器的值。

DS、ES、FS 和 GS 寄存器按下列方法之一初始化:

· 使用 SEG 运算符。例如:

```
DATA SEGMENT RW
  VART DW 0
DATA ENDS
```

```
⋮
MOV AX,SEG VARI
MOV DS,AX
```

·使用定义的段名。例如：

```
DATA SEGMENT RW
 VAR1 DW 0
DATA ENDS
⋮
MOV AX,DATA
MOV DS,AX
```

上面的方法可能产生一个汇编程序警告，因为汇编语言不知道寄存器已经被初始化。如果已经将寄存器正确地初始化了，就可以忽略这个警告。DS 寄存器也可使用 END 语句初始化。

2. 初始化堆栈寄存器

在程序开始，必须初始化 ESP 或 SP 寄存器和 SS 寄存器。在这些寄存器被初始化之前，不能使用运行堆栈。

SS 寄存器保存堆栈的基址。堆栈指针寄存器保存在堆栈中的最后一项的偏移量。这个寄存器对 USE16 模式来说是 SP，对 USE32 模式来说是 ESP。堆栈增长方向是从最高地址开始向较低的存储器地址增长。每次压栈一个字或一个双字，SP 或 ESP 中的值减 2（字）或减 4（双字），这表示地址比原来低 2 或 4 个字节。从堆栈弹出时，每次一个字或一个双字，SP 或 ESP 的值加 2 或加 4。

图 1-8 说明具有属性 USE32 的模块的运行时堆栈。

在使用堆栈之前，必须将堆栈段的基址装入 SS。这由 END 语句完成。

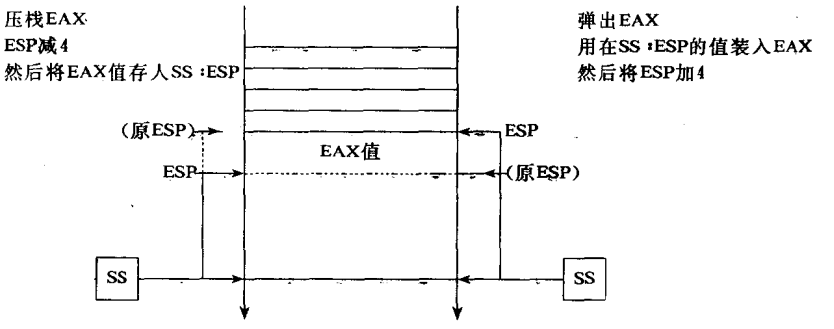

图1-8 运行时的堆栈

初始化(E)SP 以使第一条 PUSH 指令将(E)SP 设置为堆栈区中最高地址的字或双字的偏移量。因此，将用堆栈的第一个字或双字的偏移量装入(E) SP。这由 ASM386 完成。STACKSTART 运算符返回指定堆栈的第一个字或双字的偏移量值。例如：

```
MOV ESP,STACKSTART PROG_STACK
```

它初始化 ESP 为由 PROG_STACK 开始的第一个双字的偏移量。STACKSTART 在第四章中说明。