《微型计算机原理与接口技术》
第版

第4章
汇编语言程序设计

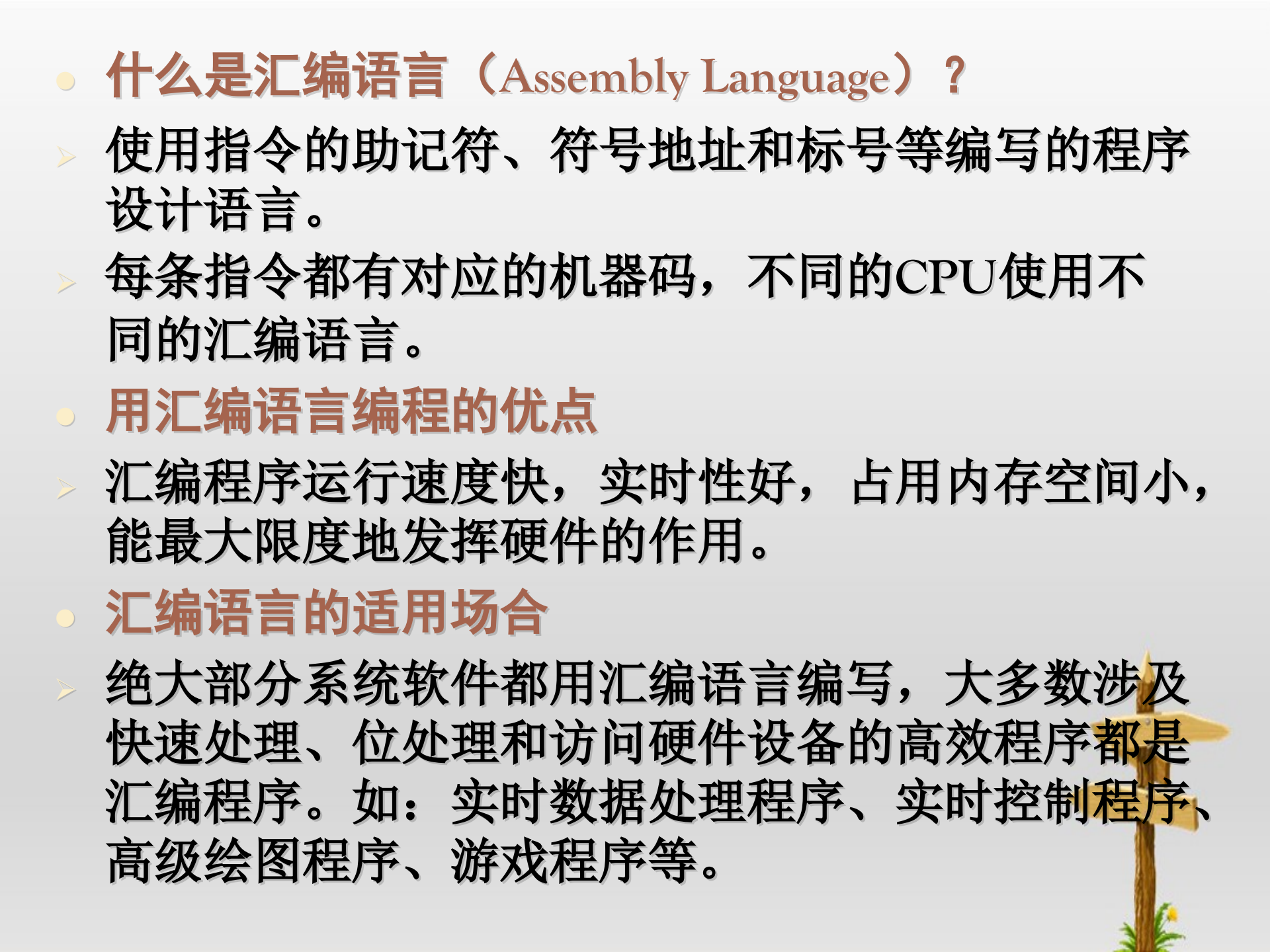
本章主要内容：

§4.1 汇编语言程序格式和伪指令

§4.2 DOS系统功能调用和BIOS中断调用

§4.3 汇编语言程序设计方法与实例

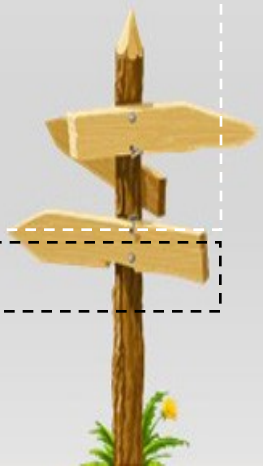
- **什么是汇编语言（Assembly Language）？**
  - 使用指令的助记符、符号地址和标号等编写的程序设计语言。
  - 每条指令都有对应的机器码，不同的CPU使用不同的汇编语言。
- **用汇编语言编程的优点**
  - 汇编程序运行速度快，实时性好，占用内存空间小，能最大限度地发挥硬件的作用。
- **汇编语言的适用场合**
  - 绝大部分系统软件都用汇编语言编写，大多数涉及快速处理、位处理和访问硬件设备的高效程序都是汇编程序。如：实时数据处理程序、实时控制程序、高级绘图程序、游戏程序等。



- **汇编语言编程的缺点和难点**
  - 编程效率低，程序设计的技巧性强，要求编程人员熟悉计算机的硬件结构和指令系统；
  - 编程和调试程序周期长，在一种机器上编写的程序，不能移植到别的类型机器上。

- 学会一种汇编语言，就能举一反三，触类旁通。学会8086汇编语言编程，就打好了学习32位高档机程序设计的基础，也便于从事单片机和嵌入式系统的设计开发。

```
DATA    SEGMENT
        DATA1   DB  0F8H, 60H,  0ACH, 74H, 3BH
        DATA2   DB  0C1H, 36H, 9EH, 0D5H, 20H
DATA    ENDS
CODE    SEGMENT
    ASSUME CS：CODE，DS：DATA
      START：  MOV    AX, DATA
               MOV    DS, AX
               MOV    CX, 5
               MOV    SI, 0
               CLC
    LOOPER：   MOV    AL, DATA2[SI]
               ADC    DATA1[SI]，AL
               INC    SI
               DEC    CX
               JNZ    LOOPER
               MOV    AH, 4CH
               INT    21H
CODE    ENDS
END   START
```



- 汇编语言源程序采用的是分段结构，即一个汇编语言源程序由若干个段组成。每个段以SEGMENT语句开始，以ENDS结束，整个程序的结尾是END语句。
- 在代码段中下面的内容是不可缺少的：
- （1）定义段
- （2）约定段寄存器和段的关系（即物理段和逻辑段的关系。使用一个或多个ASSUME语句实现）
- （3）装填段寄存器（只装填数据型段寄存器）
- （4）完成所需功能程序段
- （5）设置返回DOS的方法。

```
MYDARA   SEGMENT                          ;定义数据段起始语句
            … …                           ;定义数据
MYDATA   ENDS                             ;定义数据段终止语句
MYCODE   SEGMENT                          ;定义代码段起始语句
     ASSUME  CS ：MYCODE，  DS ：MYDATA    ;说明段寄存器和段的关系
     START ：
             MOV  AX，  MYDATA            ;装填相应的段寄存器
             MOV  DS，  AX
               … …                        ;完成所需功能的程序段
             MOV  AH，  4CH               ;设置返回DOS
             INT   21H
MYCODE   ENDS                         ;定义代码段终止语句
END   START                               ;程序结束
```

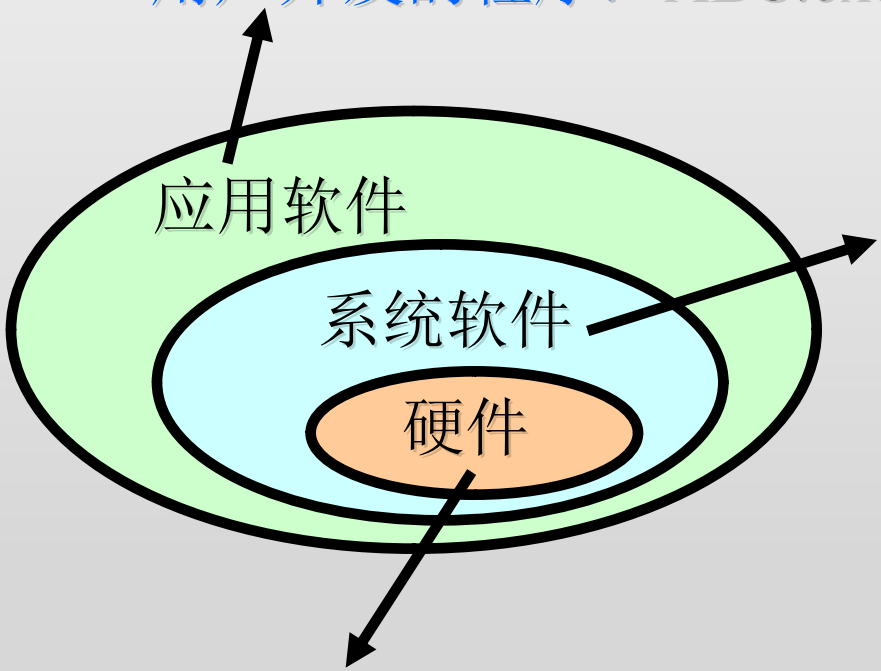
第5章　汇编语言程序设计

```
DSEG SEGMENT
    STRING1 DB 1,2,3,4,5
DSEG  ENDS
ESEG  SEGMENT
    STRING2 DB 5 DUP(?)
ESEG  ENDS
SSEG SEGMENT
    STACK'STACK'
    DW 10 DUP(?)
SSEG  ENDS
CSEG  SEGMENT
    ASSUME
   CS:CSEG,DS:DSEG
    ASSUME
   ES:ESEG,SS:SSEG
     START: MOV  AX，DSEG
            MOV  DS，AX
            MOV  AX，ESEG
            MOV  ES，AX
            MOV  AX，SSEG
            LEA  SI，  STRING1
            LEA  DI，  STRING2
            MOV  CX，5
            CLD
            REP  MOVSB
            MOV  AH，4CH
            INT  21H
CSEG ENDS
END START
```

汇编语言源程序的上机过程

上机环境

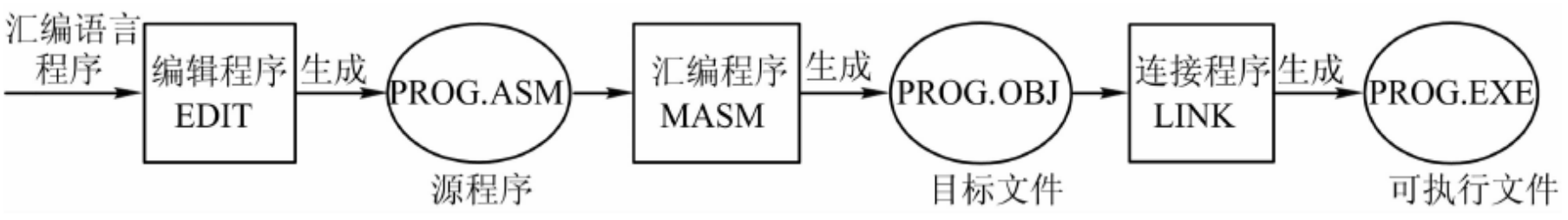
用户开发的程序：ABC.exe等



操作系统：DOS系统
编 辑 器：EDIT.exe
汇编程序：MASM.exe
连接程序：LINK.exe

CPU、存储器(ROM、RAM)、I/O接口、输入、输出设备

汇编语言的汇编处理过程



1）按语法规则编写源程序PROG.ASM；

2）用汇编程序将源程序（汇编语言源程序）翻译成目标文件PROG.OBJ；

3）用连接程序对1个或几个.OBJ模块连接后，生成能在机器上执行的程序PROG.EXE。

- 如果汇编过程中出错，要在纠错后重新汇编；
- 连接过程也会出现新的错误，需要反复修改。

用户程序的装入

完成以下操作：

- 确定内存可用部分

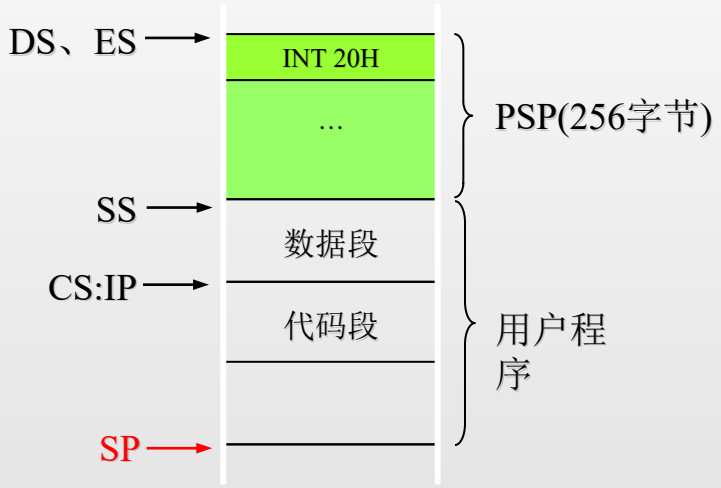
  以便存放要执行的 .exe 文件。

- 建立程序段前缀 PSP

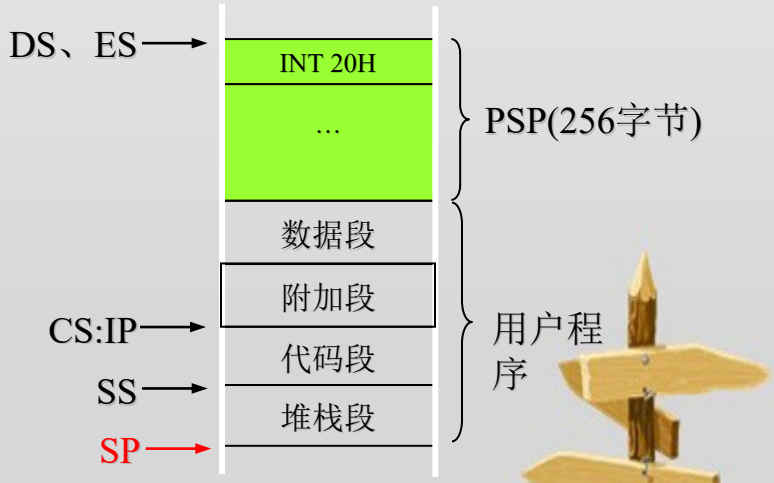
  （Program Segment Prefix）

  - 程序段前缀大小100H，即256个字节。
  - 存放执行过程中的控制信息。
  - PSP最开始的两个字节 CD20H，这是一条INT 20H中断指令。

- 装入可执行程序.exe



没有定义堆栈段的用户程序装入情况



定义了堆栈段的用户程序装入情况

§4.1 汇编语言程序格式和伪指令

4.1.1 汇编语言程序格式

4.1.2 伪指令语句

4.1.3 完整的汇编语言程序框架

4.1.1 汇编语言程序格式

汇编语言程序由指令语句和伪指令语句组成。

1. 指令语句

- 指令语句由4部分组成，格式：

  标号：指令助记符　操作数　；注释

1）标号

- 标号是指令的符号地址，后面须加冒号“:”。
- 可作标号的字符：英文字母、数字或某些特殊字符，如@　*　_　?　.等。
- 标号以英文字母或特殊字符打头，系统保留字不能做标号，如：4AB、MOV、DW、LOOP、M-D等都不是合法的标号。
- 标号具有段基址、偏移量及类型三种属性。

指令语句

2）指令助记符

- 它是指令语句中不可缺少的部分，表示指令的操作码。

3）操作数

- 1条指令可包含1个或2个操作数，也可没有操作数。
- 操作数的组成：
  - 常数 二进制数，加B；
    10进制数，可加D或省略；
    16进制数，加H，A~F前要加0；
    2-10进制BCD数，加H，要用调整指令
  - 字符或字符串 用单引号‘ ’ 括起来
  - 变量 程序运行期间可修改，数值可由DB、DW、DD等来定义
  - 标号 如JMP NEXT
  - 存储器 指令系统中已介绍
  - 表达式 将专门介绍

指令语句

4）注释

- 说明指令或程序的功能，增强程序可读性，可省略。
- 注释前必须加分号“；”。

2. 伪指令语句

- 伪指令语句的格式如下：

  名字 伪指令指示符 操作数 ；注释

1）名字

- 是给伪指令语句起的名称，格式要求与标号类似，名字后不能跟冒号“：”。

2）伪指令指示符

- 是伪指令语句中不可缺少部分，常用伪指令语句：

  段定义语句 SEGMENT和ENDS

  段分配语句 ASSUME

  过程定义语句 PROC和ENDP

  变量定义语句 DB、DW、DD、DQ、DT

  程序结束语句 END

伪指令语句

3）操作数

- 有的伪指令不允许带操作数，有的可带1个或多个操作数。

4）注释

- 注释部分与指令语句的要求类似。

3. 表达式和运算符

- 将常数、符号、寄存器等通过运算符连接起来的式子叫做表达式。
- 不论是常数、变量还是标号，都可用表达式的形式给出。
- 表4.1给出了常用表达式的运算符，还给出了一些简单的例子。

常用表达式的运算符

表 4.1　MASM 表达式中的运算符

类型	符号	名称	运算结果	举例
算术运算符	+	加法	和	3+6=9
	−	减法	差	8−3=5
	*	乘法	乘积	4 * 6=24
	/	除法	商	28/5=5
	MOD	模除	余数	28 MOD 5=3
	SHL	按位左移(n 次)	左移后 2 进制数	0010B SHL 2=1000B
	SHR	按位右移(n 次)	右移后 2 进制数	1100B SHR 1=0110B
逻辑运算符	NOT	非运算	逻辑非结果	NOT 1100B=0011B
	AND	与运算	逻辑与结果	1011B AND 0011B=0011B
	OR	或运算	逻辑或结果	1011B OR 1100B=1111B
	XOR	异或运算	逻辑异或结果	1011B XOR 0010B=1001B

常用表达式的运算符(续)

表 4.1　MASM表达式中的运算符

类型	符号	名称	运算结果	举例
关系运算符	EQ	相等	结果为真,输出全 1 结果为假,输出全 0	5 EQ 10B=全 0
	NE	不等		5 NE 10B=全 1
	LT	小于		5 LT 2=全 0
	LE	小于等于		5 LE 101B=全 1
	GT	大于		5 GT 011B=全 1
	GE	大于等于		5 GE 110B=全 0
数值返回符	SEG	返回段基址	段基址	SEG N1=N1 所在段基址
	OFFSET	返回偏移地址	偏移地址	OFFSET N1=N1 的偏移地址
	LENGTH	返回变量单元数	单元数	LENGTH N1=N1 的单元数
	TYPE	返回变量类型	(见表 4.3)	
	SIZE	返回变量总字节数	总字节数	SIZE N1=N1 的总字节数

常用表达式的运算符(续)

(续)表 4.1

类型	符号	名称	运算结果	举例
修改属性符	PTR THIS 段寄存器名	修改类型属性 指定类型属性 段超越前缀	修改后类型 指定后类型 修改段	BYTE PTR [BX] ALPHA EQU THIS BYTTE ES:[BX]
其它运算符	HIGH LOW SHORT () [ ]	分离高字节 分离低字节 短转移说明 圆括号 方括号	取高字节 取低字节 -128~127 字节间转移 改变运算优级先级 下标或间接寻址	HIGH 1234H=12H LOW 1234H=34H JMP SHORT LABEL (8-3)*6=30 MOV AX,[BX]

运算符的优先级

- 如果一个表达式中有多个运算符，则要根据优先级别从高到低的顺序进行运算，优先级别相同的运算符，则按从左到右的顺序进行运算。

表 4.2　运算符的优先级别

优先级		运算符
高级	0	(),[ ],LENGTH,SIZE
	1	PTR,OFFSET,SEG,TYPE,THIS,CS:,DS:,ES:,SS:(4 个段超越前缀)
	2	HIGH,LOW
	3	* ,/,MODE,SHL,SHR
	4	+,-
	5	EQ,NE,LT,LE,GT,GE
	6	NOT
	7	AND
	8	OR,XOR
低级	9	SHORT

运算符

1）算术运算符

例4.1 利用现行地址符“$”和减法运算符“-”求数组的长度。程序段：

```
DATA    SEGMENT                ；数据段
LIST    DB  12，38，5，29，74   ；LIST数组（变量）
COUNT   EQU  $-LIST            ；COUNT=现行地址-
                               ；LIST的偏移地址
DATA    ENDS
                ⋮
        MOV  CX，COUNT         ；CX←LIST数组长度
```

- LIST变量的起始地址偏移量为0，“$”符表示本指令的现行地址偏移量，它等于5，所以$-LIST=5-0=5，并赋予COUNT，这样可很方便地求得变量长度。

运算符

2）逻辑运算符和关系运算符

例4.2 将表达式的运算结果送到寄存器中。

```
MOV AL，NOT 10110101B
            ；AL←01001010B
MOV BL，10H GT 20H
            ；BL←00H，因10H>20H为假，输出全0
MOV BX，6 EQ 0110B
            ；BX←FFFFH，因6=6为真，输出全1
```

运算符

3）数值返回运算符

- 数值返回运算符OFFSET和SEG

例4.3 将TABLE变量的段基址：偏移量送入 DS：BX。

```
TABLE  DB  40H，79H，24H，30H，19H    ;数字0~9的
           12H，02H，78H，00H，18H    ;七段代码表
            ⋮
       MOV  BX，OFFSET TABLE ;BX←TABLE的偏址
       MOV  AX，SEG TABLE        ;AX←TABLE的段址
       MOV  DS，AX               ;DS←TABLE的段址
```

运算符

- 数值返回运算符LENGTH返回变量单元数，SIZE返回变量的总字节数。
- TYPE加在变量前，返回变量的类型属性；加在标号前，返回标号的距离属性。TYPE运算符的返回值如表4.3

表 4.3　TYPE 运算符返回值

	类型	返回值
变量	DB	1
	DW	2
	DD	4
	DQ	8
	DT	10
标号	NEAR	-1(FFH)
	FAR	-2(FEH)

运算符

例4.4 LENGTH、SIZE和TYPE运算符返回值举例

```
A1          DB    20H，30H
A2          DW    1234H，5678H
A3    DD    ?
L1:   MOV   AH，TYPE A1        ；AH←1（字节）
      MOV   BH，TYPE A2        ；AH← 2（字）
      MOV   AL，TYPE A3        ；AL←4（双字）
      MOV   BL，TYPE    L1     ；BL← 0FFH（近标号）
      MOV   BH，SIZE    A2
                               ；BH ←4（A2变量的总字节数）
      MOV   CL，LENGTH A2
                               ；CL←2（A2变量的字单元数）
```

运算符

例4.5 用LENGTH设置堆栈。

```
STAPN    DB     100 DUP (?)
         ；定义100个字节空间
TOP      EQU   LENGTH  STAPN
         ；TOP 100（变量STAPN的单元数为100字节）
```

运算符

4）修改属性运算符

例4.6 对存储单元的属性进行修改。

```
INC     BYTE PTR [BX]
                ；将字节存储单元的内容增1
```

- 用“BYTE PTR”指明存储单元[BX]为字节单元。

```
MOV     BX，ES：[DI]
                ；BX ← （16×ES+DI）的内容
```

- 源操作数[DI]也是存储单元，未加段超越前缀ES时，默认DS为段基地址，加了ES操作符后，段基地址修改成了ES。

4.1.1 汇编语言程序格式

4.1.2 伪指令语句

4.1.3 完整的汇编语言程序框架

4.1.2 伪指令语句

1.段定义语句

- 段定义语句SEGMENT和ENDS，用来定义一个逻辑段。

例4.7 用段定义语句定义一个数据段，段名为DATA，段中包含X、Y两个变量。

```
DATA    SEGMENT      ；数据段开始，DATA为段名
                     ；表示该段的基址
X       DW  1234H    ；变量X的段基址：偏移量
                     ；=DATA：0000，内容为1234H
Y       DB  56H      ；变量Y的段基址：偏移量
                     ；=DATA：0002，内容为56H
DATA    ENDS         ；数据段结束
```

1. 段定义语句

- 段定义语句的一般形式：

```
段名   SEGMENT [定位类型]    [组合类型]    [‘分类名’]
                 PAGE(页)    *NONE         ‘STACK’
               *PARA(节)     PUBLIC        ‘CODE’
                 WORD(字)    STACK
                 BYTE(字节)  COMMON
                             AT
                             MEMORY
        ┆                  ；段中内容
段名   ENDS
```

- 加“[ ]”项可省略，但堆栈段的组合类型是STACK，不可省略。
- 省略项不写时，其值用带“*”的项，它们是隐含用法，用的是默认值。

1. 段定义语句

- [ ]内3个参数的功能：

1）定位类型（Align Type）

- 用LINK程序将程序中的段相互衔接时，用定位类型来确定该段存储器的起始边界要求。
- 定位类型有四种：

PAGE（页类型）　该段起始地址能被256（页）整除 XXXX XXXX XXXX 0000 0000B

PARA（节类型）　该段起始地址能被16（节）整除XXXX XXXX XXXX XXXX 0000B

WORD　该段起始地址能被2（字节）整除

XXXX XXXX XXXX XXXX XXX0B

BYPE（字节类型）　起始地址可从任何地方开始

XXXX XXXX XXXX XXXX XXXXB

1. 段定义语句

2）组合类型（Combine-Type）

- 组合类型告诉LINK程序本段与其它段关系，包括：
  - NONE 与其它段不连接，各段有独立段基址和偏移量。
  - PUBLIC 同名同类别模块段连接成一段，段基址同, 偏移量不同。
  - COMMON 本段与其它段覆盖，偏移地址名称不同。
  - STACK 这是堆栈段，不可省略。
  - MEMORY 连接时该段放在所有段最后（最高地址）。
  - AT 定义本段的段基地址。如AT 2000H定义该段的段基地址为20000H。

3）分类名（‘class’）

- LINK将分类名相同的逻辑段组成1个段组，分类名有‘STACK’，‘CODE’和‘DATA’等。

2. 段分配语句

- 段分配语句ASSUME告诉汇编程序，4个段寄存器CS、DS、SS、ES分别与哪些段有关。格式如下，也可分两行书写。

ASSUME CS：代码段名，DS：数据段名

SS：堆栈段名，ES：附加段名

3. 过程定义语句

- 将结构和功能相同，仅有一些变量赋予的值不同的程序段独立编写，用过程定义伪指令PROC和ENDP进行定义，并把这些程序段称为过程（Procedure）或子程序，由主程序中的CALL语句来调用它们。
- 过程定义格式：

```
过程名    PROC        [NEAR]/FAR
            ┆                          ；过程内容
          RET
过程名    ENDP
```

3. 过程定义语句

- 在PROC伪语句中，必须说明是近过程NEAR还是远过程FAR，NEAR可以省略不写。在过程内部必须安排一条返回指令RET或RET *n*，以便返回主程序。
- 过程像标号一样，有3种属性：段基址、偏移地址和距离属性（NEAR或FAR），它可作为CALL指令的操作数。
- 用CALL语句调用过程，无需说明是近调用还是远调用。

  例如：

  CALL　　过程名

4. 变量定义语句

- 变量定义语句的一般形式为：

  变量名 伪指令指示符 操作数 ；注释

- 变量名用符号表示，也可以省略。
- 伪指令包括DB、DW、DD、DQ和DT，分别定义字节、字、双字、4字和10字节变量。
- 操作数可以有具体的字节、字和双字等初值，也可以不指定具体数值，而用一个问号“？”来表示，此时仅为变量留出存储单元。

4. 变量定义语句

例4.8 变量定义语句举例。

```
FIRST     DB    ?                  ；定义一个字节变量
                                   ；初始值不确定
SECOND  DB    20H，33H           ；定义两个字节变量
THIRD     DW   1122H，3344H      ；定义两个字变量
FOUR      DQ   12345678H         ；定义一个双字变量
```

4. 变量定义语句

- 还可用复制操作符DUP来定义重复变量，其格式为：

  变量名　伪指令指示符　　*n* DUP (操作数)

其中n为重复变量的个数。

例4.9　用重复操作符DUP定义变量。

```
N1    DB   100       DUP (?)
                ；分配100个字节单元，初值不确定
N2    DW   10        DUP (0)
                ；定义10个字单元，初值均为0
N3    DB   100 DUP (3 DUP(8), 6)
                ；定义100个“8，8，8，6”的数据项
```

4. 变量定义语句

- 数据项也可写成单个字符或字符串的形式，通常用字节来表示。

例4.10　字符串变量举例。

```
DB    ‘Welcome’
      ；在内存中顺序存放各字符的ACSII码
```

4. 变量定义语句

例4.11 如数据在存储单元中的存放形式如图4.2，试给出相应的变量定义语句。

```
DATA1    DB    '3', 'A'
DATA2    DW    98，100H，-2
DATA3    DD    12345678H
DATA4    DB    100 DUP (0)
```

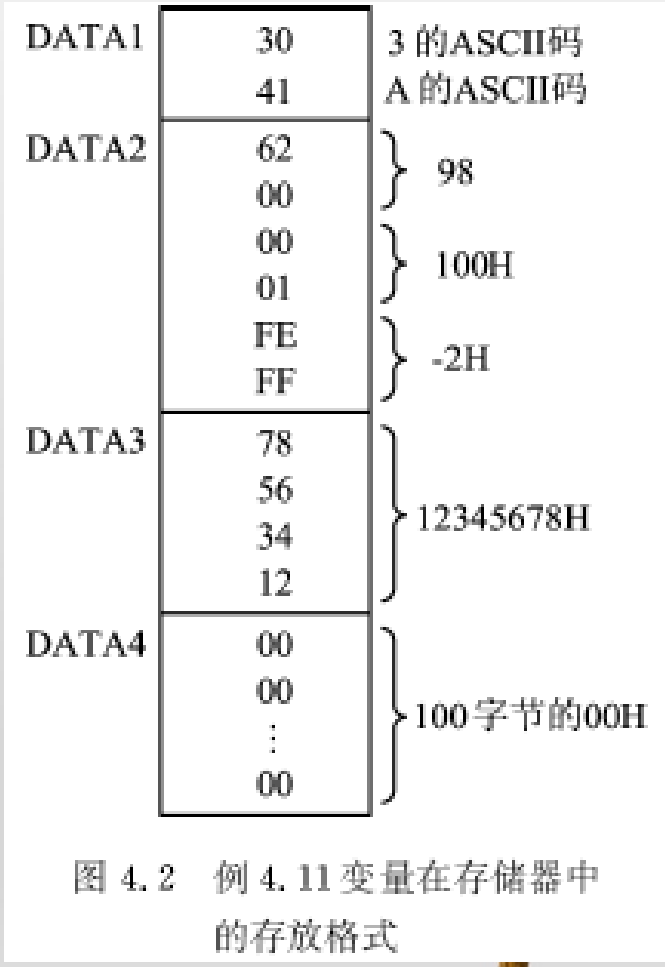


图 4.2　例 4.11 变量在存储器中的存放格式

5. 程序结束语句

- 程序结束语句的格式为：

  END　[标号名或名字]

- 它位于程序的最后一行，指示源程序结束，遇到END伪指令则停止汇编。
- 标号名或名字可省略。

6. 其它伪指令

1）等值伪指令EQU

- 使用EQU语句可使程序更清晰、易读，其格式为：

符号名　EQU　变量、标号、常数等

例4.12　EQU伪指令语句举例。

```
Profit    EQU   10 ；常数值10赋给符号名Profit
CNT1      EQU   41H      ；常数值41H赋给符号名CNT1
COUNT     EQU   8        ；常数值8赋给COUNT
```

6. 其它伪指令

2）定义类型伪指令LABEL

- 其作用与PTR类似，格式为：

  名字　LABEL　类型

  它将LABEL左边的名字定义为右边的类型。

例4.13　LABEL伪指令举例。

```
BARRY   LABEL     BYTE
                  ；将BARRY定义为字节变量
TOP       LABEL   WORD
                  ；将TOP定义为字变量
SUBRT   LABEL     FAR
                  ；将SUBRT定义为FAR标号
```

6. 其它伪指令

3）对准伪指令EVEN

- 将下一语句指向的地址调整为偶地址，确保存取一个字数据只要进行一次操作。

例4.14 对准伪指令举例。

```
DATA   SEGMENT
X      DB ‘M’          ；X变量的偏移地址为0
       EVEN     ；将下一语句指向地址调整为偶数
Y      DW       100 DUP（？）
                ；Y变量从地址为02H处开始存放
DATA   ENDS
```

6. 其它伪指令

4）ORG伪指令

- 为下面一条语句指定起始偏移地址，可放在程序的任何位置上。

例4.15 ORG伪指令举例。

```
DATA    SEGMENT
          ORG  1200H
A1              DB     12H，34H
                                ；A1变量偏移地址为1200H
        ORG   2000H
A2              DW     3040H，2830H
                                ；A2变量偏移地址为2000H
DATA    ENDS
```

6. 其它伪指令

```
CODE    SEGMENT
        ORG   400H
    ；此段代码段起始地址偏移量为400H
        ASSUME  CS：CODE，DS：DATA
          ┇
CODE    ENDS
```

6. 其它伪指令

5）模块连接伪指令PUBLIC和EXTRN

- PUBLIC和EXTRN伪指令用于解决模块连接问题。
- PUBLIC将标号、变量或数据定义为公共的，可供其它模块使用；
- EXTRN引用其它模块中已用PUBLIC伪指令定义过的标号和变量。

6. 其它伪指令

例4.16 PUBLIC和EXTRN伪指令应用举例。

```
DATA    SEGMENT
A1              DB   30H，31H          ；定义变量
A2              DW  1234H
A3              DB   100  DUP (?)
DATA    ENDS
;
CODE    SEGMENT
        ASSUME CS：CODE，DS：DATA
START：MOV AX，DATA
        ┆
SUBRT：     ┆
SUBRT   LABEL FAR                ；SUBRT为远标号
PUBLIC A1，A2，SUBRT
                        ；声明A1，A2，SUBRT为公用
```

例4.16

```
        ⋮
CODE    ENDS
;
PDATA   SEGMENT
P1          DB   20H
P2          DW   3580H
PDATA   ENDS
;
PCODE   SEGMENT
EXTRA   A1：BYTE，A2：WORD，SUBRT：FAR
                ；其它模块中用PUBLIC定义过的
                ；A1、A2、SUBRT可供本模块引用
```

例4.16

```
MAIN：MOV  AX，PDATA
        ┆
       MOV BX，OFFSET A1    ；引用变量A1
       MOV DI，OFFSET A2    ；引用变量A2
        ┆
       JMP   SUBRT          ；引用其它模块
;
SUBRT
        ┆
PCODE  ENDS
       END  MAIN
            ；程序结束，从MAIN语句开始执行
```

4.1.1 汇编语言程序格式

4.1.2 伪指令语句

4.1.3 完整的汇编语言程序框架

4.1.3 完整的汇编语言程序框架

- 完整的汇编语言程序包含数据段、代码段、堆栈段和附加数据段。
  - 其中代码段是必须要有的；
  - 堆栈段根据情况设置；
  - 代码段中要用到变量或数据时，应该设置数据段。当代码段中有字符串操作指令时，不仅要设置数据，还必需设置附加段，而且必须将源串存放在数据段中，而把目的串放在附加段中。
- 下面先给出程序框架，再介绍如何设置堆栈段，以及程序结束后怎样返回DOS操作系统。

1. 完整的汇编语言程序框架

例4.17 汇编语言程序框架。

```
DATA    SEGMENT；数据段
 X      DB   ?
 Y      DW   ?
DATA    ENDS
;
EXTRA   SEGMENT          ；附加段
ALPHA   DB    ?
BETA    DW   ?
EXTRA   ENDS
;
STACK   SEGMENT PART STACK 'STACK'  ;堆栈段
STAPN   DB    100 DUP（？）     ;定义100字节空间
TOP     EQU  LENGTH  STAPN
STACK   ENDS
```

1. 完整的汇编语言程序框架

```
;代码段
CODE  SEGMENT
MAIN  PROC  FAR                          ;过程定义语句
                          ;说明4个段寄存器分别与哪些段有关
        ASSUME  CS：CODE，    DS：DATA
                    ES：EXTRA，SS：STACK
START：
        MOV   AX，STACK   ;设堆栈段寄存器SS：SP
        MOV   SS，AX
        MOV   SP，TOP
        PUSH  DS                         ;DS入栈保护
        SUB     AX，AX                   ;AX=0
        PUSH    AX                       ;段内偏移量“0”入栈
        MOV   AX，DATA                   ;AX ← 数据段基址DATA
        MOV   DS，AX                     ;DS ← 数据段基址DATA
```

1. 完整的汇编语言程序框架

```
        MOV   AX，EXTRA
        MOV   ES，AX          ;ES← 附加段基址EXTRA
                              ;用户要编写的程序内容
        RET                   ;返回DOS
MAIN    ENDP                  ;MAIN过程结束
CODE    ENDS                  ;代码段结束
        END   MAIN            ;整个源代码结束
```

- 代码段、数据段、附加段和堆栈段，都用段定义伪指令SEGMENT和ENDS定义。
- 数据段或附加段，用DB、DW等伪指令设置实际数值。堆栈段定义了100字节空间，其数值也可修改。

1. 完整的汇编语言程序框架

- 代码段用来存放可执行的指令序列。这里用PROC FAR和ENDP伪指令将整个程序编写成一个远过程的形式，过程名为MAIN。
- 最后一条指令语句为过程返回指令RET，使程序执行完后返回到调用它的地方。
- MAIN过程中，首先用段分配伪指令ASSUME告诉汇编程序，4个段寄存器分别与哪些段相对应，但不能将段基地址装入相应的段寄存器中，还要给DS、ES和SS寄存器赋初值，CS则由操作系统赋初值。
- 对于堆栈段，要给SS和SP赋初值，以设定堆栈。

1. 完整的汇编语言程序框架

- 除了程序框架中给出的设置堆栈的方法外，还可用以下语句来设置堆栈：

```
STACK   SEGMENT  STACK ;  设置堆栈段
        DW   50 DUP（？）        ;  定义50个字空间，
                                 ;  偏移地址为00~99
TOP     LABEL WORD               ;  将TOP定义为字类型
                                 ;  其偏址为100
STACK   ENDS
CODE    SEGMENT
          ┆
START:  MOV  AX，STACK
        MOV  SS，AX              ;  设置SS
        MOV  SP，OFFSET  TOP
          ┆                      ;  SP ← TOP的偏移地址100
CODE    ENDS
```

1. 完整的汇编语言程序框架

- 设置堆栈后，紧接着用下面3条指令，将DS推入堆栈保护起来，再使00H入栈，以便在程序结束时，能执行RET指令来返回DOS，即：

```
PUSH    DS                  ；DS入栈
SUB   AX，AX
PUSH    AX                  ；00H入栈
```

- 用户编写的程序的具体内容，放在初始化程序之后，RET指令之前。
- 代码段之后，再安排一条END　MAIN指令，汇编程序遇到这条指令后就结束汇编，并自动从MAIN过程开始往下执行程序。

2. 堆栈的设置

- 在代码段中为SS：SP赋予初值，就设置了一个堆栈段。

- 如果程序中没有定义堆栈段，连接时会给出一个警告信息：

  Warning: no stack segment

  此错误不影响连接过程的完成，这时，DOS会自动定义一个堆栈段，使程序仍可正常运行。

3. 返回DOS操作系统

- 汇编语言程序在DOS下运行结束后，应能正确返回DOS，否则其它程序将无法运行，还会导致死机。
- 返回DOS的3种方法：

1）按程序框架设定的方法返回。先将主程序定义为一个远过程，再执行3条指令：

```
PUSH    DS
SUB     AX，AX
PUSH    AX
  ┇
RET
```

- 将DS和00H推入栈，再执行RET指令，转去执行INT 20H指令，返回DOS。这是返回DOS的常规方法。
- 执行这几条指令后，为什么能正确返回DOS？具体过程请参看本教材。

3. 返回DOS操作系统

2）执行4CH号DOS功能调用。程序结束前按如下方法使用4CH号DOS功能调用指令，返回DOS。

```
MOV  AX，4C00H      ；AH=4CH，是DOS功能号
                    ；AL通常置为0
INT   21H
```

- 这种方法功能更强，更安全，使用也比较方便，建议使用这种方法返回DOS。

3）若编写的程序要以.COM文件的形式执行，可用INT 20H指令直接返回DOS。

返回DOS的方法

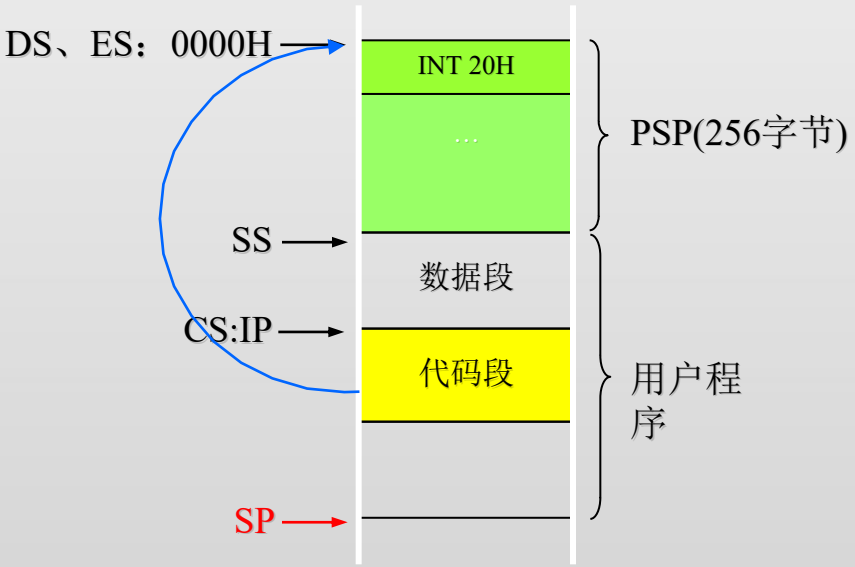
(1) 非标准方法

调用INT 21H的4CH功能，即

```
MOV AH，4CH
INT  21H
```

(2) 标准方法

借用PSP首单元的INT 20H返回DOS。



① 把主程序定义成一个远距离过程。即：

```
过程名   PROC   FAR
        …… ……

                RET
过程名   ENDP
```

② 在给DS、ES赋初值之前，用下列三条指令，把PSP首单元的逻辑地址压入堆栈，即

```
PUSH   DS              ；PSP段地址压栈
MOV    AX，0           ；或用XOR AX,AX指令
PUSH   AX              ；PSP段首单元的偏移地址
                       ；压栈(偏移地址为0000H)
```

例：

```
DATA   SEGMENT
    NUM  DB  82H,68H,88H
    SUM  DB  ?
DATA   ENDS
CODE  SEGMENT
    MAIN  PROC  FAR
    ASSUME CS:CODE,DS:DATA
    START: PUSH  DS
           XOR   AX,AX
           PUSH  AX
           MOV   AX,DATA
           MOV   DS,AX
           LEA    BX, NUM
           MOV   AL,[BX]
           INC    BX
           ADD    AL,[BX]
           INC    BX
           ADD    AL,[BX]
           MOV   SUM,AL
           RET
    MAIN   ENDP
CODE  ENDS
END   START
```