第四章　汇编语言程序组织

4．1 汇编语言语句格式

- 指令语句格式
- 伪指令语句格式

（1）指令语句格式

- 标号：　指令助记符　操作数　；注释

- 例：
- L1：ADD　AL，BL　；寄存器内容相加

（1）指令语句格式

- **标号字段**：任选字段，性质为符号偏（位）移量，用于标记转移指令的目的地址.
- 在汇编过程中，被引用的标号会被替换为数值偏（位）移量.
- 标号位于语句开头，必须使用冒号结尾。

（1）指令语句格式

- **指令助记符与操作数**：组成汇编指令本身，必不可少的字段。

- 汇编过程中，语句中仅这两个字段被替换为机器指令。

（1）指令语句格式

- **注释字段**：任选字段，用于说明程序片段或指令的功能，提高源程序的可读性.
- 汇编过程中不处理该字段。
- 注释必须使用分号作为开始。

（１）指令语句格式

- LOP：　　MOV AX，0000H　　；将AX清零

- 在转移指令中引用标号：
- JMP　LOP

（2） 伪指令语句格式

- 伪指令语句：用于指示汇编程序如何汇编源程序。
- 与指令语句不同，伪指令不产生机器指令代码，其操作在汇编阶段完成。
- 伪指令的功能通常为分配存储单元、计算表达式、定义常量等。

（2） 伪指令语句格式

- 符号名　　伪指令符　操作数　；注释

- 例：
- VAR1　DB　54H；分配一个字节单元
- CON1　EQU　123；定义一个常量

（2） 伪指令语句格式

- 符号名字段：任选字段，可为常量名、变量名、子程序名称、结构名称、记录名称。

- 符号名在目标代码中不会出现，仅在汇编过程中使用。

（2） 伪指令语句格式

- 伪指令符：伪指令语句中必不可少的字段，由它指示具体操作。

- 操作数字段：由伪指令操作符决定。操作数可以是常数、字符串、常量名、变量名、表达式等。

- 注释字段：与指令语句中的注释字段含义相同。

（2） 伪指令语句格式

- VAR1 DB 12H ；定义变量VAR1

- 在指令中引用变量：
- MOV AL，VAR1

- 汇编后指令中的VAR1会被替换为相应的数值位移量。

- 伪指令中的VAR1不会在目标代码中存在。

（3）标识符

- 标识符是汇编语言中所有用户自定义符号的总称，它有一定的构成规则：
- （a）字符个数在1个到31个之间。
- （b）第一个字符必须是字母或特殊字符（特殊字符：？ @ _ . $）

（3）标识符

- （c）除第一个字符外，其他字符可以是字母、数字、特殊字符

- （d）不能使用保留字（汇编程序定义的符号名称），包括各种指令助记符、伪指令符、寄存器名称等等。

4.2 汇编语言中使用的数据
4.2.1 数据表示

- 数据表示是指汇编语言所规定的用于表示各种数据的具体方式。

- 数据表示可分为**数值表示**和**字符串表示**两类。

（1）数值表示

- 二进制整数：由数字0和1组成，以字母B结尾。例如，10110101B。
- 八进制整数：由0－7组成，以字母Q或O结尾。例如，35761Q。
- 十进制整数：由0－9组成，以字母D结尾或结尾不带任何字母。

（1）数值表示

- 十六进制整数：由0－F组成，以字母H结尾。例如，0A845H。

- 若十六进制整数以字母为最高位，前面必须添一个0，避免与标识符混淆。

（2）字符串表示

- 使用单引号或双引号括起来的任意字符序列称为字符串。例如：

- ‘Hello, world!’
- ‘A’
- ‘678’

（3）数据表示在程序中的用途

- 1）在指令语句中作为立即数

- MOV　AX，45F3H
- MOV　BL，‘C’

- 2）在指令语句中作为偏移量中的位移量分量

- MOV　AX，[1000H]
- ADD　10H[BX]，AX

（3）数据表示在程序中的用途

- 3）在数据定义伪指令中，作为存储单元存放的初值

- DB　10H，0F4H
- DW　0D3E5H
- DB　‘Hello’

数据表示与编码间的关系

- ；下面指令中的立即数，程序员使用了带符号的
- ；数据表示，汇编后生成二进制补码
- MOV　AL, -12

- ；下面指令中的立即数，程序员使用了无符号的
- ；数据表示，汇编后生成二进制的无符号数编码
- ADD　BH, 250

数据表示与编码间的关系

- ；下面指令中的立即数具有多义性，既可解释为
- ；补码，也可解释为无符号数编码，具体解释为
- ；何种编码，只能由程序员所使用的标志位确定

- SUB　BL，34
- MOV　AH，0E2H

4.2.2 变量

- 变量在汇编语言中的概念与高级语言中一致，指某一个特定的内存单元或内存区域。

- 在汇编阶段，变量在段中的起始偏移量（段内偏移量）、所占用的空间已经完全被确定。

4.2.2 变量

- 在程序执行阶段，变量中的数据随时可能发生变化。

- 在程序中引用变量本质上是引用内存单元的段内偏移量，通常访问的信息是变量中的数据。

（1）数据定义伪指令

- 语句格式：
- 变量名　DB（DW、DD）表达式1，表达式2，…；注释
- 变量名：可选字段，代表当前所定义内存单元的段内偏移量。

（1）数据定义伪指令

- 数据定义伪指令：必选字段，用于指明当前所定义变量所占用的空间大小。
- DB：字节
- DW：字
- DD：双字
- DQ：8字节
- DT：10字节

（1）数据定义伪指令

- 下面是一个完整的数据段定义：

- DATA1　SEGMENT
- VAR1　DB　10H
- VAR2　DW　3A4DH
- VAR3　DD　？
- DATA1　ENDS

- 可用这个例子来解释变量的三种固有属性。

变量的固有属性

- 1）段属性（SEG）

- VAR1、VAR2、VAR3定义在同一个逻辑段DATA1中，确定了访问这三个变量正确的段基值。

- 关于变量的段属性，需注意指令中隐含使用的段寄存器，要保证正确的段基值是放在当前使用的段寄存器中。

- MOV　AL，VAR1
- 指令默认DS中存放DATA1段基值。

变量的固有属性

- 2）偏移量属性（OFFSET）

- 偏移量属性是指变量相对于段起始地址的字节距离。

- 变量名称即符号偏移量；定义一个变量名即标识其段内偏移量；引用一个变量名即引用对应的偏移量。

变量的固有属性

- 段基值：偏移量构成完整的逻辑地址。
- 变量的段属性和偏移量属性完全确定了变量的起始地址。
- 上例中，VAR1的偏移量为0000H，VAR2为0001H，VAR3为0003H。

变量的固有属性

- 3）类型属性（TYPE）

- 类型属性表示变量所占用的字节数，该属性由伪指令符决定。

- DB：单字节
- DW：字（双字节）
- DD：双字（4字节）

变量的固有属性

- 在汇编过程中，汇编程序会检查变量类型，如果操作数类型不匹配，会提示语法错误。
- 例如：MOV AL，VAR2
- VAR2是字单元，AL是字节寄存器，会出现语法错误。

变量的固有属性

- 通过段属性、偏移量属性、类型属性，可以把变量在内存中的确切起始地址、所占用的字节数确定下来。

- 在汇编语言程序中访问的变量就是一个具体的、实在的变量，而不象高级语言中那么抽象。

分配内存空间而不定义变量名

- DATA1　SEGMENT
- VAR1　　DB　12H
- 　　DB　24H
- VAR2　　DW　5F4BH
- DATA1　ENDS

- 未定义变量名的内存空间在指令中如何寻址？

分配内存空间而不定义变量名

- （1）使用数值形式的段内偏移量

- 直接寻址（数值位移量）
- MOV　AL, [0001H]

- 寄存器间接寻址（基址或变址分量）
- MOV　BX, 0001H
- MOV　AL, [BX]

分配内存空间而不定义变量名

- （2）通过引用其它变量实现间接引用

- MOV　AL，VAR1+1
- 或
- MOV　AL，VAR2-1

变量初值的设定

- 程序装载到内存中以后，程序执行前变量的取值称为初值。

- 变量初值的设定通过表达式来完成。

变量初值的设定

- 1）数值表达式

```
data  segment
varb1  db    10h
varb2  db    10,11,12
          db    0ffh,0
varw   dw    1234h,5678h
vard    dd    12345678h
data  ends
```

变量初值的设定

- 2）字符串表达式

- data segment
- string1 db ‘ABCD’
- string2 dw ‘AB’,’CD’,’A’
- string3 dd ‘AB’
- data ends

变量初值的设定

- 3）问号表达式

- 用问号定义初值，仅为变量分配空间，其初值为内存单元中原有的随机值。

- 例：
- varb db ?,?

变量初值的设定

- 4）带DUP的表达式
- 变量名　DB等　表达式1　DUP（表达式2）

- DUP（Duplication）指多次重复分配内存空间，并且为每次分配的空间定义相同的初值。

- 表达式1：定义重复分配空间的次数。

- 表达式2：定义每次分配所用的初值。

变量初值的设定

- 例：

- aryb1 db 10h dup(2)
- aryb2 db 20h dup('ABC')
- aryw dw 10h dup(?)

（2）变量的引用

- 1）变量在指令语句中的引用（位移量）

- 在指令中引用变量名作为符号位移量可实现**直接寻址、基址寻址、变址寻址、基址变址寻址**。

- 通过各种寻址方式，指令可对变量或变量数组中的数据进行读写操作。

（2）变量的引用

- 2）在伪指令中引用变量（**段内偏移量、逻辑地址**）
- 这里不是指定义变量，而是指在伪指令语句中引用变量。
- 和指令中引用变量相似，在伪指令中引用变量也是引用变量的段内偏移量或者完整逻辑地址。

（2）变量的引用

- ADR1　DW　VAR1+3　　；存放VAR1+3偏移量
- ADR2　DD　VAR1-2　　；存放VAR1-2逻辑地址

- 注意，在伪指令中引用变量时不能使用DB伪指令。

4.2.3 标号

- 变量是用于存放数据的内存单元所对应的偏移量，是一种符号地址。
- 标号同样是一种符号地址，用于指示特定内存单元的偏移量。
- 但与变量不同，标号所指示的地址位于代码段。

4.2.3 标号

- 标号就是程序中某一条特定指令的符号地址。

- 转移指令引用标号，即是引用标号所标识的指令在代码段中的偏移量或完整的逻辑地址。

4.2.3 标号

- 标号的一般定义和引用
- …
- L1: INC SI ；在指令前定义标号L1
- …
- JMP L1 ；跳转到标号L1处
- …

（1）标号的属性

- 和变量一样，标号也有三种固有属性
- 1）段属性（SEG）
- 标号是某一条指令的符号地址，一条指令总是位于某一个特定的代码段。该代码段的段基值就是标号的段基值。
- 在程序执行过程中，当前代码段的段基值总是由CS段寄存器指出。

（1）标号的属性

- 2）偏移量属性（OFFSET）

- 偏移量是指带标号指令在代码段中相对于段起始地址的字节距离。

- 通过段属性和偏移量属性的结合，就能够完全确定带标号指令在内存中的确切起始地址。

（1）标号的属性

- 3）类型属性（TYPE）

- 变量的类型是指变量在内存中占用的字节空间大小。

- 标号的类型不是指令的字节长度，而是指标号的引用类型，指地址的长度。

（1）标号的属性

- NEAR型：使用关键字NEAR说明的标号仅标识指令的段内偏移量，只能用于段内引用。

- FAR型：使用关键字FAR说明的标号标识指令的完整逻辑地址，可用于段内或段间引用。

（1）标号的属性

- 标号的隐含属性为NEAR，当定义标号未指明类型时，汇编程序默认它为NEAR型。

- 如果需要定义FAR类型的标号，必须使用LABEL伪指令显式定义其类型。

（2）LABEL伪指令

- LABEL伪指令与指令语句连用时，用于定义指令的标号；与数据定义伪指令连用时，用于定义变量。

（2）LABEL伪指令

- 1）用LABEL伪指令定义标号

- NEXTF　LABEL　FAR
- NEXT：　ADD　AL，BL

- NEXT为NEAR型，NEXTF则为FAR型；但是两个标号均指向同一条指令。

（2）LABEL伪指令

- 若在段内使用转移指令，应引用NEXT标号，因为它占用的空间小；

- 若在其它代码段使用转移指令，应引用NEXTF标号，因为只有它能给出段基值。

（2）LABEL伪指令

- 2）用LABEL伪指令定义变量

- VARB　LABEL　BYTE
- VARW　DW　　10H　DUP（1122H）

- VARB和VARW两个变量的段属性和偏移量属性完全相同，具有相同的逻辑地址。

- 但是，两个变量的类型却不同，VARB是字节类型，VARW是字类型。

（2）LABEL伪指令

- 通过LABEL伪指令与数据定义伪指令连用，可以为同一个逻辑地址定义不同类型的变量。

- 程序可使用不同类型来访问相同的内存单元。

4.3 符号定义伪指令

- 如果一个常数或表达式会在源程序中经常使用，可以定义一个符号来表示它。
- 如果符号所对应的表达式很复杂，那么在源程序中使用符号会使程序描述更简单、清晰。
- 汇编语言中符号的概念和高级语言中常量的概念相似。

4.3.1 等值语句

- 语句格式：
- 符号　EQU　表达式

- EQU伪指令把语句右边的表达式赋值给左边的符号。

- 语句中的表达式可以为常数、数值表达式、地址表达式、变量名、标号、寄存器名称、指令助记符等。

4.3.1 等值语句

- 定义：
- NUM　　EQU　10H
- CONT　EQU　123＋34－67

- 引用：
- VAR　　DB　　NUM
- MOV　AX，　NUM

4.3.1 等值语句

- 使用EQU伪指令定义符号后，不能再次使用EQU伪指令修改符号定义。
- 在指令语句或伪指令语句中引用该符号相当于引用它对应的数值、标识符或保留字。
- 汇编过程中，源程序中所有符号都会被替换为它代表的内容。

4.3.1 等值语句

- 符号仅在源程序汇编阶段有效，在程序执行阶段无符号概念。
- 在目标代码中符号都已经全部被替换为相应的内容。
- 与变量不同，符号定义并没有分配任何存储单元。

4.3.2 等号语句

- 语句格式：
- 符号 = 表达式

- 等号语句的作用和等值语句完全一致，但用等号定义过的符号可以再次使用等号修改其定义。

4.3.2 等号语句

- CONT = 10
- M = MOV
- ……
- CONT = CONT +10
- M = MUL

4.4 表达式与运算符

- 指令和伪指令中都可使用表达式来描述操作数。
- 表达式为汇编语言中的高级语法成分，可以简化复杂操作数的描述。
- 表达式：由常数、变量、标号等元素通过各种运算符连接而成的算式。
- 在汇编语言中，表达式一般分为数值表达式和地址表达式两种。

4.4 表达式与运算符

- 表达式的**计算**都在**汇编阶段**完成，最后生成的目标代码中，表达式被替换为具体的数值。

- 运算符：汇编语言中的运算符有五种，包括算术运算符、逻辑运算符、关系运算符、数值返回运算符、属性与分离运算符。

4.4.1 算术运算符

- 可用于数值表达式的运算符：+、-、*、/、MOD、SHL、SHR
- 可用于地址表达式的运算符：+、-、[ ]
- 注：这里所有算术运算都是整数运算。

4.4.1 算术运算符

- 地址表达式中，两个变量间的运算只能是减法。

- MOV　　AX，VAR2－VAR1

4.4.1 算术运算符

- ＮＵＭ　ＥＱＵ　１１１１Ｂ
- ……
- MOV　AL, NUM SHL 4
- ADD　BL, NUM SHR 4

- 汇编后：
- ＭＯＶ　　ＡＬ，１１１１００００Ｂ
- ＡＤＤ　　ＢＬ，００００００００Ｂ

- 注：ＳＨＬ，ＳＨＲ运算符与移位指令有区别。

4.4.1 算术运算符

- 注：表达式的计算一定是在汇编过程中完成的，不是在程序的执行阶段来完成。

- 例：MOV　AL，ARRY[3]

- 源操作数用地址表达式给出，汇编过程中会完成ARRY+3的计算，并用计算结果替换表达式，寻址方式为直接寻址方式。

4.4.1 算术运算符

- MOV　AL，ARRY[BX]

- 源操作数不是表达式，因为偏移量的计算是在指令执行阶段完成的。

- ARRY是符号位移量，在汇编过程中它被替换为具体数值，ARRY这一部分可看作表达式，但[BX]这一部分决不是一个表达式。

- 寻址方式为基址寻址。

4.4.2 逻辑运算符

- 包括AND、OR、XOR、NOT四个运算符，只能用于数值表达式。

- 注：逻辑运算符与逻辑指令有区别。

4.4.3 关系运算符

- 包括EQ、NE、LT、LE、GT、GE六个运算符。
- 数值表达式中，对左右两个表达式的值进行比较；地址表达式中，对左右两个变量的偏移量进行比较。
- 注：关系运算符左右两个表达式必须性质相同。
- 如果关系成立，运算结果为全1；如果不成立，运算结果为全0。

4.4.3 关系运算符

- DA1　　DB　　3　LT　8
- DA2　　DB　　10　NE　0AH

- ……
- MOV　　BX，DA2　GE　DA1

4.4.4 数值返回运算符

- 这类运算符只能针对变量或者标号，用于返回变量或标号的段基值、偏移量、类型等固有属性。

- 注：与其它运算符一样，返回运算是在汇编阶段完成，而不是在执行阶段。

（1）SEG运算符

- 加在引用的变量名或标号前面，运算结果是返回该变量或标号的段基值。

- ＭＯＶ　　ＡＸ，ＳＥＧ　ＶＡＲ１
- ＭＯＶ　　ＤＳ，ＡＸ

（2）OFFSET运算符

- 加在引用的变量名或标号前，运算结果为返回该变量或标号的偏移量。

- mov ax, offset var1
- lea   ax, var1

（3）TYPE运算符

- 加在引用的变量名或标号前，运算结果为返回该变量或标号占用的字节数。

- var1 db ?
- var2 dw ?
- ……
- mov ax, type var1
- mov bx, type var2

（4）LENGTH运算符

- 返回数组中的数据个数（注：与单个数据占用的字节数无关）。

- 如果定义该变量时使用了DUP关键字，那么返回重复的次数；如果没有使用DUP，那么返回1。

- LENGTH运算符只有针对使用DUP关键字定义的变量才有实际意义。

（4）LENGTH运算符

- arry1 db 100 dup(0)
- var1 db 12, 32, 0
- len1 dw length arry1
- len2 dw length var1

（5）SIZE运算符

- 只能加在引用变量名前，返回结果相当于LENGTH运算符和TYPE运算符的乘积，即数组变量总共占用的字节数目。

4.4.5 属性运算符

- 1）PTR运算符
- 语句格式：
- 类型　PTR　地址

- var1 dw 1234h
- ……
- mov al, byte ptr var1

4.5 程序的段结构

- 8086/8088系统中存储器的为分段管理机制。
- 这里从汇编语言语法的角度来讨论段的定义。

4.5.1 段定义伪指令

- 段名　SEGMENT　定位类型　组合类型
- 　　…
- 　　…
- 段名　ENDS

（1）段名

- 必选字段，具体名称由用户自己决定，需要满足标识符条件，段头和段尾名称必须一致。

（2）定位类型

- 可选的字段，定义段起始地址。从这里可以知道段基址和段起始地址的区别。

- 默认为PARA定位类型，以节边界为段起始地址。

（3）组合类型

- 可选字段，用于定义当前段与其他各段之间的重叠、邻接关系。

- 1）NONE：如果没有定义组合类型字段，那么NONE是隐含使用的组合类型。这种类型说明该段在逻辑上独立，和其他段没有重叠或邻接关系。

（3）组合类型

- ２）STACK：把段名和该段相同的所有段邻接在一起构成一个连续的段，并指定该段为堆栈段。

- 操作系统准备运行该程序时，根据这个连续段来设置SS和SP寄存器，从而初始化堆栈。

- 若未定义STACK段，SS和SP寄存器的初始化必须由用户自己使用指令来实现。

4.5.2 段寻址伪指令ASSUME

- 指定段与各段寄存器之间的对应关系，指示汇编程序如何确定指令隐含使用的段寄存器。

- data1 segment
- da1 db ?
- data1 ends

4.5.2 段寻址伪指令ASSUME

- data2 segment
- da2 db ?
- data2 ends

- code segment
- da3 db ?
- assume cs:code, ds:data1, es:data2
- ……
- mov da1, 10h
- mov da2, 10h
- mov da3, 10h
- code ends

4.5.2 段寻址伪指令ASSUME

- 若不使用ASSUME伪指令，指令大多数情况下隐含使用DS段寄存器。

- 这种前提下，如果要使用指令访问ES、CS等段寄存器所指示段中的变量，那么必须在指令中显式给出段前缀。

- 例如：MOV AX，ES：VAR1

4.5.2 段寻址伪指令ASSUME

- ASSUME语句的方便之处在于它可以指定指令中的隐含段寄存器，避免使用段前缀。

- 在程序设计中，一般ASSUME语句在代码段中位于所有指令语句之前。

4.5.2 段寻址伪指令ASSUME

- 把段基值装入段寄存器这一过程是在程序执行阶段完成。

- 即使程序中使用了ASSUME语句，但在执行阶段各段寄存器未装入正确的段基值，那么仍不能正确地访问变量。

4.5.3 段寄存器的装入
（1）CS的装入

- CS段寄存器和IP指令指针的初始化由操作系统完成。
- 当用户向操作系统提交一个程序时，操作系统会自动把程序中第一条指令的逻辑地址装入到CS和IP中。
- 此后，CPU从这条指令开始执行，并且不断修改IP的内容使它指向下一条指令。

（1）CS的装入

- 第一条指令的地址在源程序中是通过END伪指令来确定的。

- END伪指令一定是汇编语言源程序中的最后一条伪指令，而且一定只在源程序中出现一次。

（1）CS的装入

- END语句的功能：
- 1）标记源程序结束的位置：END语句以后的任何语句都不会被汇编程序解释。

- 2）指定程序中第一条指令的逻辑地址：
- END伪指令后一定要带一个标号，该标号指定程序中第一条指令的逻辑地址。

（2）DS、ES的装入

- 与CS的装入方式不同，DS、ES的段基值装入必须由程序员使用MOV指令在程序中完成。

- 对DS、ES段寄存器的初始化一般在代码段的最开始，因为之后的指令可能会频繁的访问数据段或附加段。

- 只有保证段寄存器中段基值正确，之后的数据操作才是正确的。

（3）SS的装入

- SS段寄存器的初始化过程分为两种类型，自动和手动初始化。
- 1）自动初始化
- 如果希望采用这种初始化方式，在定义堆栈段时必须使用STACK作为组合类型。
- 对ＳＴＡＣＫ段，操作系统将程序装入到内存准备运行时，会自动初始化SS和SP。

1）自动初始化

- stack1 segment stack
- dw 20h dup(0)
- stack1 ends

（3）SS的装入

- 2）手动初始化
- 与DS、ES的初始化过程相似，需要在程序中使用MOV指令来实现。
- 如果要使用组合类型不是STACK属性的段作为堆栈段，那么必须使用这种初始化方式。
- 采用手动初始化方式时，SS和SP需同时初始化。

2）手动初始化

- stack2 segment
- dw 30h dup(0)
- top label word
- stack2 ends
- ……
- mov ax, stack2
- mov ss, ax
- mov sp, offset top

4.7 汇编语言源程序的基本结构框架

- 一个程序提交给操作系统后就能获得操作系统下放的CPU控制权。
- 程序执行完后必须返回操作系统，即把CPU控制权交还给操作系统。
- 对于DOS来说，如果应用程序不主动返还CPU控制权，那么它将永远失去对CPU的控制。
- 为实现应用程序正常返回，可采用多种程序框架。

使用特定的中断调用返回DOS

- 程序最后两条被执行的指令是固定的：

- MOV　AH，4CH
- INT　 21H

- 使用21H号中断调用的4CH号子功能，也能完成终止程序、返回DOS操作系统的功能。

4.8 其他伪指令
（1）定位伪指令ORG和位置计数器

- 汇编程序在翻译源程序过程中，会对每一个段设置一个位置计数器，用于表示当前解释的指令或数据在段中的偏移量。

- 注：位置计数器是汇编程序在汇编阶段中使用的一种工具，在程序执行阶段并没有这样一个计数器。

（1）定位伪指令ORG和位置计数器

- ORG伪指令可以修改位置计数器，以指定指令或变量位置，或者预留内存空间。

- ORG语句格式如下：

- ORG　数值表达式

（1）定位伪指令ORG和位置计数器

- ORG伪指令会把数值表达式的值赋给位置计数器，改变它的取值。

- 注：在表达式中可使用特殊符号“$“表示位置计数器当前的取值，即可以根据位置计数器当前取值计算一个新取值。

（1）定位伪指令ORG和位置计数器

```
data  segment
        org    10h
var1  dw     200h,300h
        org    $+5
var2  db     15h  dup(0)
cont  equ    $-var2
data  ends
```

简明的程序框架

- 数据段、堆栈段略去
- code segment
- assume cs:code,ds:data,ss:stack1
- begin: mov ax, data ;初始化DS、ES
- mov ds, ax
- …… ;程序主体
- mov ah, 4ch ;返回DOS
- int 21h
- code ends
- end begin ;结束源程序

另一种程序框架

- 将应用程序作为操作系统的子程序，采用子程序框架。
- 定义子程序的语句格式：
- 过程名　PROC　NEAR / FAR
- …
- RET
- …
- 过程名　ENDP

子程序定义

- 1）子程序名：不可缺少的字段，必须满足标识符的定义。

- 2）子程序名与标号类似，具有三个相同的固有属性。调用子程序或转移到某个标号这两个过程都要改变程序的执行流程。区别在于子程序调用需要保存返回点到堆栈。

- 3）一个子程序中至少有一条RET指令，在程序执行阶段，子程序中最后一条被执行的指令一定是RET指令。

PSP段

- DOS操作系统把任何一个程序装入到内存时，都会给它分配一个PSP（程序段前缀）空间。
- DS、ES未初始化时，它们都指向PSP的起始地址，该地址中存放了一条中断调用指令“INT 20H”。
- 第20H号中断调用，可以完成终止当前应用程序并返回DOS操作系统的功能。

程序框架原理

- 初始化DS、ES后，DS、ES中的内容会改变，不再指向PSP。
- 若希望使用PSP起始地址处存放的INT 20H指令返回DOS，则应把应用程序设置为一个FAR类型的子程序。
- 在程序开始时保存逻辑地址（DS）：0000H到堆栈，作为返回点。
- 程序执行完毕后使用RET指令，使流程转向PSP的起始地址，执行INT 20H后会返回到操作系统。

源程序结构

- ；数据段、堆栈段略去
- code segment
- assume cs:code, ds:data, ss:stack1
- main proc far
- push ds
- mov ax, 0
- push ax

源程序结构

- mov ax, data ;初始化DS、ES
- mov ds, ax
- …… ;程序主体
- ret ;返回PSP首地址
- main endp ;结束子程序定义
- code ends ;结束段定义
- end main ;结束源程序定义