第三部分　微机原理与接口

第13章　汇编语言程序设计

13.1 汇编语言的基本概念

- 将汇编语言源程序“翻译”成机器语言（目标代码）的过程称为汇编。
- 目前使用较多的汇编程序是宏汇编（MASM）程序。
- 不同种类的CPU具有不同的汇编语言源程序（指令系统不同），互相之间不一定通用。

13.1.1 汇编语言源程序的结构

- 一个完整的汇编语言源程序通常由若干个逻辑段组成，包括数据段、附加段、堆栈段和代码段。
- 每个逻辑段以SEGMENT语句开始，以ENDS语句结束。
- 整个源程序用END语句结尾。

```
段名1    SEGMENT
           ⋮
段名1    ENDS
段名2    SEGMENT
           ⋮
段名2    ENDS
…
段名n    SEGMENT
           ⋮
段名n    ENDS
END
```

例，将数据段中两个字单元数据相加，结果存入附加段字单元中。

```
DSEG    SEGMENT
        DATA1   DW   0F865H
        DATA2   DW   360CH
DSEG    ENDS
ESEG    SEGMENT
        SUM     DW   2   DUP(0)
ESEG    ENDS
CSEG    SEGMENT
        ASSUME  CS:CSEG,DS:DSEG,ES:ESEG
START:  MOV    AX,    DSEG
        MOV    DS,    AX
        MOV    AX,    ESEG
        MOV    ES,    AX
        MOV    AX,    DATA1
        ADD    AX,    DATA2
        MOV    SUM,   AX
        HLT
CSEG    ENDS
        END    START
```

13.1.2 汇编语言语句类型及格式

- 汇编语言源程序语句：指令性语句和指示性语句。
- 指令性语句：由指令系统中助记符组成的语句，汇编后生成可被CPU执行的目标代码。
- 指示性语句：用于告诉汇编程序如何对源程序进行汇编，不生成可执行的目标代码，又称为伪操作语句或伪指令。

指令性语句格式：

{标号:} {前缀} 操作码 {操作数，操作数}{;注释}

指示性语句格式：

{名字} 伪指令 操作数 {，操作数，…}{;注释}

数据项

13.1.3 数据项及表达式

- 操作数
  - 寄存器
  - 存储器
  - 数据项
    - 常量
    - 标号
    - 变量名/段名
    - 表达式

1 常量

数字常量：

- 二进制数：以字母B结尾，如01001001B。
- 八进制数：以字母O或Q结尾，如631Q。
- 十进制数：以字母D结尾或不加结尾。
- 十六进制数：以字母H结尾，若以字母A～F开始的十六进制数，前面要加一个数字0，如0FEH。

字符串常量：

- 字符串常量：用单引号括起来的一个或多个字符，其值为这些字符的ASCII码值。

2 标号

标号放在一条指令的前面，它就是该指令在内存中存放地址的符号表示，即指令地址的别名。

每个标号具有三种属性：

（1）段值属性（SEG）

标号所在段的段地址。

（2）偏移量属性（OFFSET）

标号所在段的段首到定义标号的地址之间的字节数，即偏移地址。

（3）类型属性

NEAR和 FAR两种类型。NEAR型地址指针为2个字节，只能段内引用；FAR型4个字节，可以在其他段引用。

3 变量

变量是用来表示存放数据的存储单元，程序中以变量名的形式来访问变量，而变量名就是存放某数据块的存储单元首地址。

格式：

```
                DB                                  ；定义字节变量
                DW                                  ；定义字变量
变量名          DD      表达式1，表达式2……；定义4字节变量
                DQ                                  ；定义8字节变量
                DT                                  ；定义10字节变量
```

说明：其中表达式1、表达式2是给存储单元赋的初值。

举例：

```
VAR_DATA    SEGMENT
DATA1    DB     12H
DATA2    DB     20H,30H
DATA3    DW     5678H
VAR_DATA    ENDS
```

每个变量也有三种属性：

（1）段值属性（SEG）

变量所在段的段地址。

（2）偏移量属性（OFFSET）

变量所在段的段首到变量地址之间的字节数，即偏移地址。

（3）类型属性

BYTE（字节）、WORD（字）、DWORD（双字）、QWORD（四字）等，变量类型必须满足指令要求。

4 表达式

表达式是指令或伪指令语句操作数的常见形式。它由常数、变量、标号等通过操作运算符连接而成。

注意：任何表达式的值在程序被汇编的过程中进行计算确定，而不是到程序运行时才计算。

例，**MOV AX, OADH AND 0CCH**

MOV BX, OFFSET DMem1

13.2 伪指令

伪指令在汇编过程中由汇编程序执行，如定义数据、分配存储区、定义段以及定义过程等。

13.2.1 数据定义伪指令

1 格式

变量名 伪指令 操作数1，操作数2…

伪指令：

（1）DB，字节类型；

（2）DW，字类型；

（3）DD，双字类型；

（4）DQ，四字类型；

（5）DT，十字节类型；

2 操作数

操作数的值不能超过伪指令所定义的范围。

（1）数值表达式

例，DATA1　　DB　32H，30H

例，DATA2　　DW　1000H，100*6+88

（2）？表达式：表示可以预置任意内容

例，DA-BYTE　DB　？，？，？

（3）字符串表达式

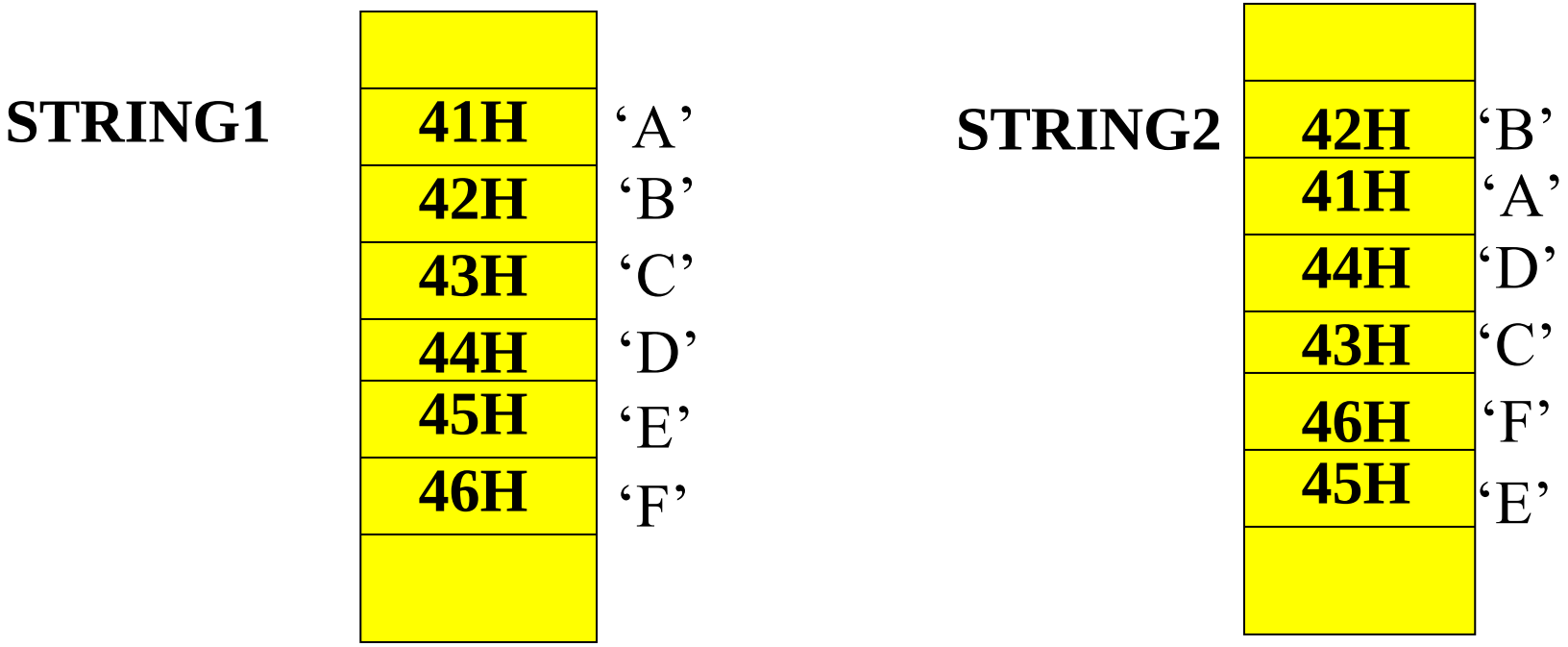
例，STRING1　DB　'ABCDEF'　;41H,42H…

例，STRING2　DW　'AB','CD'；4142H,4344H

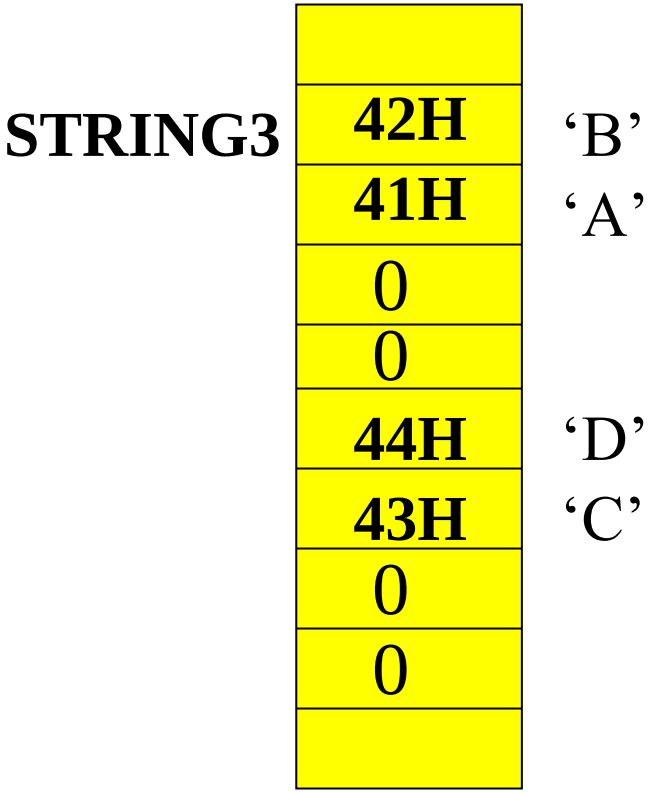
STRING3　DW　'ABCD'；错误,DW/DD不允许2个以上字符

例如：STRING1 DB ‘ABCDEF’

例如：STRING2 DW ‘AB’,‘CD’,’EF’



例如：STRING3 DD ‘AB’, ‘CD’



注意：DW和DD伪指令不能用两个以上字符构成的字符串赋初值，否则将出错。

3 重复操作符

当同样的操作数重复多次时，可用DUP表示。

格式：

变量名　伪指令　表达式1 DUP （表达式2）

表达式1为重复次数；

表达式2为重复内容；

例，　DATA_A DB　10H DUP(?)

DATA_B DB　20H DUP(30H,31H)

DATA_B DW　20H DUP(3000H,3100H)

4 变量的使用

（1）在指令语句中引用

在指令语句中直接引用变量名则表示该操作数为存储器操作数，其表现形式为段地址：[偏移地址]。

例，
```
DA1   DB   0FEH
DA2   DW   52ACH, 1234H
……
MOV  AL, DA1        ;将0FEH传送到AL中
MOV  BX, DA2        ;将52ACH传送到BX中
MOV  BX, DA2+1      ;将3452H传送到BX中
```

必须注意两操作数类型的一致。

当变量出现在寄存器间接寻址的操作数中时表示该变量的偏移地址。

例：

```
DA3  DB  10H  DUP(?)
DA4  DW  10H  DUP（1）
MOV  DA3[SI]，AL
ADD  DX，DA4[BX][DI]
```

将AL的内容送入从DA3 开始再偏移(SI)的存储单元中

将从DA4开始再偏移(BX)+(DI)的字存储单元的内容与DX的内容相加，结果送回DX中。

指令语句中，变量名代表→段地址：[偏移地址]

[ ]既指明存储器操作数又有加号的作用。

（2）在伪指令中引用

```
NUM     DB   75H
ARRAY   DW   20H DUP(0)
ADR1    DW   NUM
ADR2    DW   ARRAY[2]
ADR3    DD   NUM
```

表示取变量的偏移地址

前两个字节存偏移地址，后两个字节存段地址

伪指令中，变量名属性→段地址：偏移地址，[ ]加号的作用

5 属性定义伪指令LABEL

```
例如：  SUB1_FAR   LABEL   FAR
        SUB1: MOV  AX，30H
              ……
```

- SUB1_FAR与SUB1两个标号具有相同的逻辑地址。被转移指令或调用指令使用时是指同一个入口地址。
- SUB1只能被段内调用，SUB1_FAR可以被段间指令调用。

```
例如：DATA_BYTE   LABEL  BYTE
      DATA_WORD  DW  20H  DUP（？）
```

- DATA_BYTE与DATA_WORD具有相同的段基址和偏移量。
- DATA_BYTE可以被用来存取一个字节数据，而DATA_WORD则不能。

6 表达式运算符

（1）算术运算符： +、 —、 *、 / (取整) 、MOD(取余)

例，MOV AL, 8+5

例，MOV AX, NUM+(9-1) *2

（2）逻辑运算符：AND、OR、NOT、XOR

用于对数值进行的按位逻辑运算并得到一个确定的数值结果。

例，MOV AX, OADH AND 0CCH

（3）关系运算符：

EQ（等于）、NE（不等于）、LT（小于）、 LE（小于等于）、GT（大于）、 GE（大于等于）

- 关系运算符用来比较两个表达式的大小。关系运算符比较的两个表达式必须同为常数或同一逻辑段中的变量。
- 若是常量的比较，则按无符号数进行比较。
- 若是变量的比较，则比较它们的偏移量的大小。
- 关系运算的结果只能是“真”（全1）或“假”（全0）。

例，　MOV　AX，0FH　EQ　1111B =>MOV　AX ,0FFFFH
　　MOV　BX，0FH　NE　1111B =>MOV　BX, 0

例，　VAR　DW　CNUM　LT　0ABH

该语句在汇编时，根据符号常量CNUM的大小来决定VAR存储单元的值，当CNUM<0ABH时，则变量VAR的内容为0FFFFH，否则VAR的内容为0。

（4）取值运算符：

①OFFSET：得到标号或变量的偏移地址。

例，MOV　BX,　OFFSET　DMem1

等同于，LEA　BX, DMem1

②SEG：得到标号或变量的段地址。

例，MOV　AX,　SEG　DMem1

MOV　DS,　AX

（5）属性运算符：

格式：　类型　PTR　表达式

作用：将表达式所指定的标号、变量或用其它形式表示的存储器地址的类型属性修改为“类型”所指的值。

类型可以是BYTE、WORD、DWORD、NEAR和FAR。这种修改是临时的，只在含有该运算符的语句内有效。

例，MOV　AX，WORD PTR DMem1

例，JMP　FAR PTR LAB1

（6）存储器操作数运算符[ ]：

格式：　表达式1[表达式2]

作用: 表明操作数为存储器操作数，同时对表达式还具有相加的运算作用，且表达式为操作数的偏移地址。

（7）段重设运算符：

格式：　段寄存器：[表达式]

作用：“:”用来指定一个存储器操作数的段属性。

例，　MOV　AX,　ES:[BX]

（8）TYPE运算符：

取变量或标号的类型属性，并用数字形式表示。对变量来说就是取它的类型字节长度。

变量	BYTE	1
	WORD	2
	DWORD	4
	QWORD	8
	TWORD	10

标号	NEAR	-1
	FAR	-2

例如：

```
V1   DB   'ABCDE'
V2   DW   1234H, 5678H
V3   DD   V2
     ……
     MOV  AL , TYPE  V1
     MOV  CL , TYPE  V2
     MOV  CH , TYPE  V3
```

经汇编后的等效指令序列如下：

```
MOV  AL，1
MOV  CL，2
MOV  CH，4
```

（9）LENGTH运算符：

该运算符用于取变量的长度。

- 如果变量是用重复数据操作符DUP说明的，则LENGTH运算取最外层DUP给定的值。
- 如果没有用DUP说明，则LENGTH运算返回值总是1。

例如

```
K1  DB   10H DUP（0）
K2  DB    10H，20H，30H，40H
K3  DW  20H DUP（0，1，2 DUP（0））
K4  DB   ‘ABCDEFGH’
……..
```

```
MOV   AL,  LENGTH   K1; (AL)=10H
MOV   BL,  LENGTH   K2 ; (BL)=1
MOV   CX,  LENGTH   K3 ;  (CX)=20H
MOV   DX,  LENGTH   K4 ;  (DX)=1
```

（10）SIZE运算符：

该运算符只能用于变量，SIZE取值等于LENGTH和TYPE两个运算符返回值的乘积。

例如，对于上面例子，加上以下指令：

```
K1  DB   10H DUP（0）
K2  DB    10H，20H，30H，40H
K3  DW  20H DUP（0，1，2 DUP（0））
K4  DB   ‘ABCDEFGH’
……..
```

```
MOV   AL，SIZE  K1；  （AL）=10H
MOV    BL，SIZE  K2；  （BL）=1
MOV    CL，SIZE  K3；  （CL）=20H*2=40H
MOV    DL，SIZE  K4；  （DL）=1
```

（11）HIGH/LOW运算符：

这两个运算符用来将表达式的值分离出高字节和低字节。

格式：

```
HIGH   表达式
LOW    表达式
```

- 如果表达式为一个常量，则将其分离成高8位和低8位
- 如果表达式是一个地址（段基值或偏移量）时，则分离出它的高字节和低字节。

```
DATA   SEGMENT
CONST  EQU   0ABCDH
DA1    DB    10H DUP （0）
DA2    DW    20H DUP （0）
DATA   ENDS
       …….
MOV    AH ,HIGH  CONST
MOV    AL, LOW  CONST
MOV    BH, HIGH  (OFFSET DA1)
MOV    BL,  LOW  (OFFSET  DA2)
MOV    CH,  HIGH (SEG  DA1)
MOV    CL,  LOW  (SEG   DA2)
```

设DATA段的段基值是0926H，则指令序列汇编后的等效指令为：

```
MOV   AH，  0ABH
MOV   AL，  0CDH
MOV   BH，  00H
MOV   BL，  10H
MOV   CH，  09H
MOV   CL，  26H
```

不能用来分离一个变量、寄存器或存储器单元的内容。

下面语句中的用法是错误的。

```
DA1   DW  1234H
……
MOV   AH，HIGH  DA1
MOV   BH，  LOW  AX
MOV   CH，HIGH [SI]
```

13.2.2 符号定义伪指令

符号定义伪指令将常数或表达式等形式用某个指定的符号来表示。

格式：符号名 EQU 表达式

功能：用符号名来表示EQU右边的表达式。后面的程序中一旦出现该符号名，汇编程序将把它替换成该表达式。

1.常数或数值表达式

```
COUNT  EQU  5
NUM    EQU  COUNT+6
```

2.地址表达式

```
ADR1   EQU  DS：[BP+14]
```

3.变量、寄存器名或指令助记符

```
CREG   EQU  CX
```

在同一源程序中，同一符号不能用EQU定义多次。

例，
```
CBD  EQU  DAA
CBD  EQU  ADD
```

13.2.3 段定义伪指令

伪指令SEGMENT和ENDS用于定义一个逻辑段。使用时必须配对，分别表示定义的开始与结束。

一般格式：

```
段名   SEGMENT   [定位类型] [组合类型] ['类别名']
          .......
          ........
          .......
段名    ENDS
```

1、段名

- 段名由用户自己任意选定，但要符合标识符定义规则
- 最好选用与该逻辑段用途相关的名称。如第一个数据段为DATA1,第二个数据段为DATA2等。
- 一对SEGMENT和ENDS前的段名必须一致。
- 段名代表段地址。

2、定位类型

定位类型决定了当前段起始数据边界，即第一个可存放数据的存储单元位置，再由起始数据边界决定当前段地址。定位类型可以有4种取值。

（1）PAGE：表示该段从一个页面的边界开始存放数据。

由于一个页面为256个字节，并且页面编号从0开始，因此，PAGE定位类型的段起始地址的最后8位二进制数一定为0，即以00H结尾的地址。

（2）PARA:表示该段从一个小节的边界开始存放数据。

如果用户未选定位类型，则缺省为PARA。

（3）WORD:表示该段从一个偶数字节地址开始存放数据，即段起始数据单元地址的最后一位二进制数一定是0。

（4）BYTE:表示该段起始数据单元地址可以是任一地址值。

定位类型为PAGE和PARA时，段基物理地址（0H结尾）直接选用段起始数据地址，即它们是重合的。定位类型为WORD和BYTE时，段基物理地址与段起始数据地址可能不同。

即当定位类型为WORD和BYTE时，第一个存储单元的偏移地址不一定为0。

3、组合类型

指定段与段之间的连接关系和定位。它有六种取值选择。

（1）NONE：表示本段与其他逻辑段无连接关系，不同的程序模块中，即使具有相同的段名，也分别装入内存。默认情况下，为NONE类型。

（2）PUBLIC：表示对于不同程序模块中用PUBLIC说明的具有相同段名的逻辑段，汇编时将它们连接在一起，构成一个大的逻辑段。

（3）STACK：把不同程序模块中所有同名的堆栈段连接成一个连续堆栈段，且系统自动对SS段寄存器初始化为该连续段的段地址，并初始化堆栈指针SP。

（4）COMMON：对不同程序模块中用COMMON说明的同名逻辑段，连接时从同一个地址开始装入，即所有逻辑段重叠在一起，连接之后的长度等于最长的逻辑段。

（5）MEMORY：表示本段在存储器中应定位在所有其它段之后的最高地址上。如果有多个用MEMORY说明的段，则只处理第一个用MEMORY说明的段，其余的被视为COMMON。

（5）AT：根据表达式的值定位段地址。如AT 8000H，则段地址为8000H，即本段的起始物理地址为80000H。

4、类别名

- 类别名为某一个段或几个相同类型段设定类型名称。
- 类别名必须用单引号引起来。所用字符串可任意选定，但它不能使用程序中的标号、变量名或其它定义的符号。
- 系统进行连接处理时，把不同程序模块类别名相同的段存放在相邻的存储区，但段的划分与使用仍按原来的设定。

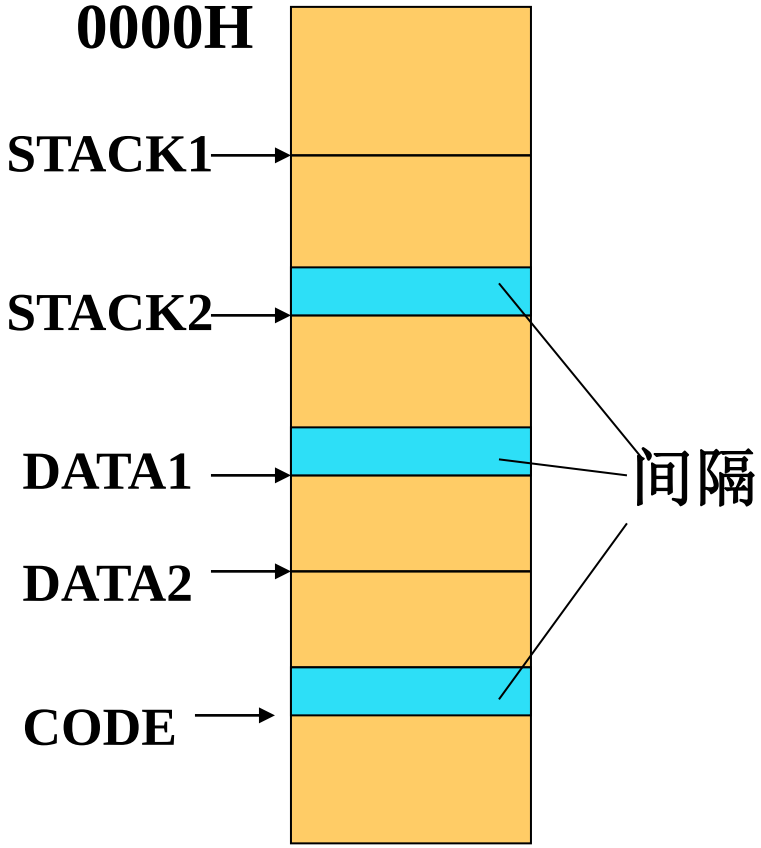
在定义一个段时，段名是必须有的项，而定位类型、组合类型和类别名三个参数是可选项，主要针对不同程序模块。各个参数之间用空格分隔，参数之间的顺序不能改变。

下面是一个分段结构的源程序框架。

```
STACK1   SEGMENT  PARA  STACK  'STACK0'
          .....
STACK1   ENDS
DATA1    SEGMENT  PARA  'DATA'
         ......
DATA1    ENDS
STACK2   SEGMENT  PARA  'STACK0'
         ......
STACK2   ENDS
CODE     SEGMENT  PARA  MEMORY
       ASSUME  CS:CODE,DS:DATA1,SS:STACK1
MAIN:    ......
         ......
CODE     ENDS
DATA2    SEGMENT  BYTE  'DATA'
         ......
DATA2    ENDS
              END  MAIN
```

上述源程序经汇编和LINK程序进行连接处理后，程序被装入内存的情况如图所示。

如果在段定义中选用了PARA定位类型说明，则在一个段的结尾与另一个段的开始之间可能存在一些空白，图中以兰色框表示。CODE段的组合类型为MEMORY,因此被装入在其它段之后。



13.2.4 设定段寄存器伪指令

- ASSUME的作用是告诉汇编程序,在处理源程序时，定义的逻辑段与哪个寄存器关联。即对应的标号、变量等对应的段地址用哪个段寄存器来表示，可以在程序中多次设定。
- ASSUME并不设置各个段寄存器的具体内容，段寄存器的值是在程序运行时设定的。

一般格式：

ASSUME 段寄存器名：段名，段寄存器名：段名，…

其中段寄存器名可以为CS,DS,ES和SS，段名是用SEGMENT/ENDS伪指令定义的段名。

```
DATA1  SEGMENT
        VAR1   DB  12H
DATA1  ENDS
DATA2  SEGMENT
        VAR2   DB  34H
 DATA2  ENDS
 CODE   SEGMENT
        VAR3   DB  56H
     ASSUME  CS:CODE,DS:DATA1,ES:DATA2
START: .....
        .....
        INC  VAR1
        INC  VAR2
        INC  VAR3
        ......
CODE   ENDS
        END START
```

该指令汇编时，VAR1使用的是DS，默认

该指令被汇编时，VAR2使用的是ES,即指令编码中有段前缀ES

该指令汇编时，VAR3使用的是CS，即指令编码中有段前缀CS

段寄存器初始化方法：

1. DS和ES的装入

在程序中，使用数据传送语句来实现对DS和ES的装入。

```
DATA1    SEGMENT
         DBYTE1  DB  12H
DATA1    ENDS
DATA2    SEGMENT
         DBYTE2  DB 14H DUP(?)
DATA2    ENDS
CODE      SEGMENT
            ASSUME  CS:CODE,DS:DATA1,ES:DATA2
START: MOV  AX,DATA1
            MOV  DS,AX
            MOV  AX,DATA2
            MOV  ES,AX
            ......
CODE   ENDS
            END  START
```

2. SS的装入

SS的装入有两种方法：

（1）在段定义伪指令的组合类型项中，使用STACK参数，并在段寻址伪指令ASSUME语句中把该段与SS段寄存器关联。

例，
```
STACK1  SEGMENT  PARA  STACK
        DB  40H DUP(?)
STACK1  ENDS
    ......
CODE    SEGMENT
        ASSUME  CS:CODE,SS:STACK1
    ......
```

SS将被系统自动装入STACK1段的段地址，且（SP）=40H。

（2）如果在段定义伪指令的组合类型中，未使用STACK参数，或者是在程序中要调换到另一个堆栈，这时，可以使用类似于DS和ES的装入方法。

```
例，  DATA_STACK   SEGMENT
                    DB  40H DUP(?)
      DATA_STACK   ENDS
                ......
      CODE            SEGMENT
                ......
                    MOV  AX,   DATA_STACK
                    MOV   SS,   AX
                    MOV   SP,   OFFSET DATA_STACK
                ......
```

3、CS的装入

CPU在执行指令之前根据CS和IP的内容来从内存中提取指令,即必须在程序执行之前装入CS和IP的值。因此,CS和IP的初始值就不能用可执行语句来装入。

装入CS和IP一般有下面两种情况。

(1)汇编时系统软件按照结束伪指令指定的地址装入初始的CS和IP

任何一个源程序都必须以END伪指令来结束。

格式：　END 起始地址

起始地址：可以是一个标号或表达式，它与程序中第一条指令语句前所加的标号必须一致。

- END伪指令用来指示源程序结束和指定程序运行时的起始地址。
- 当程序被装入内存时，系统软件根据起始地址的段地址和偏移量分别被装入CS和IP中。

例，

```
      ......
CODE  SEGMENT
      ASSUME  CS:CODE,......
START: ......
      ......
CODE  ENDS
      END START
```

(2)在程序运行期间，当执行某些指令时，CPU自动修改CS和IP，使它们指向新的代码段。

例如：

执行段间过程调用CALL和段间返回指令RET;

执行段间无条件转移指令JMP;

响应中断及中断返回指令;

执行硬件复位操作。

13.2.5 过程定义伪指令

在程序设计过程中，常常将具有一定功能的程序段设计成一个子程序。在MASM宏汇编程序中，用过程来构造子程序。

过程定义伪指令格式如下：

```
过程名   PROC  [NEAR/FAR]
          ……
          ……
         RET
过程名   ENDP
```

过程名是子程序的名称，它被用作过程调用指令CALL的目的操作数。它类同一个标号的作用。具有段、偏移量和类型三个属性。而类型属性使用NEAR和FAR来指定，若没有指定，则隐含为NEAR。

NEAR：过程只能被本段指令调用

FAR：过程可以供其它段的指令调用

每一个过程中必须包含有返回指令RET，其作用是控制CPU从该过程中返回到调用过程的主程序。

13.2.6当前位置计数器$与定位伪指令ORG

```
DATA1   SEGMENT
        ORG  30H
  DB1   DB  12H,34H
        ORG  $+20H
  STRING  DB  'ABCDEFGHI'
  COUNT   EQU  $-STRING
  DB2  DW  $
  DB3  DB  $ ,$+20H
DATA1   ENDS
  CODE     SEGMENT
           ASSUME  CS:CODE.....
     ORG  10H
  START:  MOV  AX,DATA
           MOV  DS,AX
           .....
  CODE    ENDS
           END START
```

DB1在DATA1段内的偏移量为30H

保留20H个字节单元，其后再存放'ABCD....

计算STRING的长度

取$的偏移量,类似变量的用法

此语句错误!

13.3 DOS功能调用

DOS操作系统为程序设计人员提供了可以直接调用的功能子程序。调用这些子程序可以实现从键盘输入数据，将数据送显示器显示，以及磁盘操作等功能。

调用DOS功能时需要使用软中断指令 INT 21H，并在执行该指令之前，需要将要调用的功能号送入寄存器AH中，有关的参量送入指定的寄存器。

调用过程包括以下三个步骤：

* 送入口参量给指定寄存器
* AH←功能号
* INT 21H

1、带显示的键盘输入（1号功能）

调用该功能子程序将等待键盘输入，直到按下一个键。将按键字符的ASCII码送入AL寄存器，并在屏幕上显示该字符。如果是Ctrl-C组合键，则停止程序运行。该功能调用无入口参量。

例，　MOV AH，01H

　　INT 21H

2、不带显示的键盘输入（8号功能）

该功能调用与1号功能的作用相似，区别是8号功能将不显示输入的字符。调用方法为：

```
MOV  AH，8
INT    21H
```

3、字符串输入（0AH号功能）

该功能调用可实现从键盘输入一个字符串，其长度可达255个字符。调用该功能前，应在内存中建立一个输入缓冲区。

缓冲区第一个字节是可输入的最大字符数+1；第二个字节是系统在调用该功能时，自动填入的本次调用时实际输入的字符个数；从第三个字节开始存放输入字符的ASCII码。

当用户输入回车键时，结束输入，并将回车键的ASCII码（0DH）作为最后一个字符送入缓冲区。但它不计入实际输入字符个数。

调用入口参量：

DS和DX寄存器分别装入输入缓冲区的段地址和偏移量

```
CHAR_BUF DB 31H       ;缓冲区的最大长度
         DB 0         ;存实际输入字符数
         DB 31H DUP(0);输入缓冲区
         ......
MOV DX,SEG CHAR_BUF;如果DS已经指向CHAR_BUF所在
MOV DS,DX          ;数据段，则可以省去这两条指令
MOV DX,OFFSET CHAR_BUF
MOV AH,0AH
INT 21H
```

4、字符显示（2号功能）

该功能实现在屏幕上显示单个字符。

入口参数：DL←要显示字符的ASCII码。

例，　MOV　DL，'A'

　　MOV AH，2

　　INT　21H

5、字符串显示（9号功能）

该功能实现将一个字符串显示到屏幕上。

入口参数：

（1）将待显示的字符串存放在一个数据缓冲区，字符串以符号“$”作为结束标志。

（2）将字符串的首址的段地址和偏移量分别送入DS和DX中

例，

```
CHARS    DB   'This is a test.' , 0DH, 0AH, '$'
.....
MOV  DX,  OFFSET CHARS
MOV AH,   9
INT  21H
```

6、返回DOS系统

执行DOS功能调用4CH，可以控制用户程序结束，并返回DOS操作系统。

```
MOV  AH，4CH
INT    21H
```