第4章　汇编语言程序设计

4.1 汇编语言源程序

一、汇编语言基本概念

- 机器语言：能够直接控制计算机完成指定动作的。
- 汇编语言：一种用符号书写的、基本操作与机器指令相对应的、并遵循一定语法规则的计算机语言。
- 汇编源程序：用汇编语言提供的指令和伪指令编写的程序
- 汇编：汇编语言源程序要翻译成机器语言程序才可以由计算机执行。这个翻译的过程称为汇编。
- 汇编程序：把汇编源程序翻译成目标程序的语言加工程序
- 链接程序：将所需的库文件或其它目标文件链接到一起形成可执行文件

引入汇编语言的原因：克服机器语言的缺点。采用助记符表示机器指令的操作码，用变量代替操作数的存放地址等。

二、汇编语言源程序结构

- 完整的汇编语言源程序由段（代码段、数据段、附加段或堆栈段）组成；一个汇编语言源程序必须包含一个代码段，并指示程序执行的起始点（开始标号），一个程序只有一个起始点。数据段、堆栈段和附加段视情况而定。
- 每个段由若干语句行组成。语句行是汇编的编程基础。所有的指令必须位于代码段内，伪指令可根据需要位于任一段内。
- 段以“SEGMENT”开始，以“ENDS”结束，由用户定义。

汇编语言实例

```
DATA    SEGMENT
STR     DB 0DH,0AH, Hello,World! !$'
DATA    ENDS
STACK   SEGMENT STACK
        DB  100 DUP(0)
STACK   ENDS
CODE    SEGMENT
        ASSUME DS:DATA,CS:CODE,SS:STACK
BEGIN:  MOV AX,DATA
        MOV DS,AX
        LEA DX,STR
        MOV AH,9
        INT 21H
        MOV AH,4CH
        INT 21H
CODE    ENDS
        END BEGIN
```

汇编语言源程序的基本格式

```
DATA    SEGMENT
        ┆                                   ; 存放数据的数据段
DATA    ENDS
EXTRA   SEGMENT
        ┆                                   ; 存放数据的附加段
EXTRA   ENDS
STACK   SEGMENT   STACK
        DW 100 DUP(?)      ; 定义了100个字单元的堆栈段
STACK   ENDS
CODE    SEGMENT
        ASSUME    CS:CODE,DS:DATA,SS:STACK,ES:EXTRA
```

```
START:MOV    AX,DATA
      MOV    DS,AX              ; 段地址装入DS
      MOV    AX,EXTRA
      MOV    ES,AX              ; 段地址装入ES
        ¦                       ; 核心程序段
      MOV    AH, 4CH            ; 系统功能调用
      INT    21H                ; 返回操作系统
CODE  ENDS
      END    START
```

三、汇编语言语句类型及格式

1、汇编语言的语句类型

◆指令性语句（指令指令）

使CPU产生动作、并在程序执行时才处理的语句，就是第3章学习的处理器指令。

◆指示性语句（伪指令）

不使CPU产生动作、在程序执行前由汇编程序处理的说明性语句，例如，数据说明、变量定义等等。

◆宏指令

四、数据项及表达式

1、常量、变量与标号

（1）常量

在程序执行过程中值不发生变化的量，是一个立即数。

数值常量、字符常量、符号常量

数值常量

常量形式	格式	X 的取值	常例	说明
二进制常量	XX…XB	0 或 1	01110011B	数据类型后缀为 B
八进制常量	XX…XO	0~7	12537O	数据类型后缀为 O
十进制常量	XX…X XX…XD	0~9	1234D 1234	数据类型后缀为 D 后缀可省略
十六进制常量	XX…XH	0~9 A~F	0AB12H	如果第一位数是 A~F， 则必须在数的前面加 0

字符型常量

字符型常量：用引号引起来的可显示的ASCII码字符（字符串）。如‘a’、‘12345’ 等

符号常量

为经常使用的数值常量定义的名字，然后在语句中用名字来表示该常量。需用EQU或“=”定义后使用。

（2）变量

变量:操作数中第一个数据的偏移地址，在程序中作为存储器操作数使用，变量名由用户定义。如前面例子中的BLOCK、BUF等，一般在数据段中定义。

变量具有三种属性：

①段　值—变量定义所在段的段地址。

②偏移量—变量所指的单元地址与段起始地址之间的位移量

③类型—有字节（byte）、字(word)和双字(double word)等。

提示：每一个变量定义后都具有此三种属性，设置变量名是为了方便存取它指示的存储单元。

（3）标号

标号：指令的偏移地址。在汇编源程序中，只有在需要转向一条指令时，才为该指令设置标号，以便在转移类指令（含子程序调用指令）中直接引用这个标号。因此，标号可作为转移类指令的操作数，即转移地址。

标号的三种属性：

①段值:所在段的段地址,总是在CS段寄存器中;
②偏移量:与变量相同,所在段的段内偏移地址;
③类型：分NEAR和FAR两种。

NEAR—标号所在的指令与转移指令（调用指令）在同一段内，只需改变IP。可省略
FAR—标号所在指令与转移指令（调用指令）不在同一段内。

2、表达式

表达式：由运算对象和运算符组成的合法式子。分数值表达式和地址表达式两种。汇编时按一定的优先规则对表达式进行计算。

数值表达式：数值表达式的运算结果是一个数据，其只有大小，没有属性。如：MOV DX,(6*A-B)/2 中，(6*A-B)/2是一个数值表达式

地址表达式：由运算符将常量、变量、标号或寄存器的内容连接而成的式子，其值表示存储单元的偏移地址。当这个地址中存放的是数据时，称为变量；当这个地址中存放的是指令时，则称为标号 。如“BUF+1”就是一个地址表达式，表示指向BUF字节单元的下一个单元地址

3、运算符

表 4-1 MASM 支持的运算符

类型	符号	名称	运算结果	示例
算术运算符	+	加法	和	3+2=5
	–	减法	差	8–4=4
	*	乘法	乘积	3*4=12
	/	除法	商	7/2=3
	MOD	模除	余数	7 MOD 2=1
	SHL	左移	左移后的二进制数值	1010B SHL 2=1000B
	SHR	右移	右移后的二进制数值	1010B SHR 2=0010B
逻辑运算符	AND	与运算	逻辑与	1011B AND 1100B=1000B
	OR	或运算	逻辑或	1011B OR 1100B=1111B
	XOR	异或运算	逻辑异或	1011B XOR 1100B=0111B
	NOT	非运算	逻辑非	NOT 1011B=0100B
关系运算符	EQ	相等	关系成立结果为全‘1’ 关系不成立结果为全‘0’	5 EQ 4 = 全‘0’
	NE	不相等		5 NE 4 = 全‘1’
	GT	大于		5 GT 4 = 全‘1’
	LE	不大于		5 LE 4 = 全‘0’
	LT	小于		5 LT 4 = 全‘0’
	GE	不小于		5 GE 4 = 全‘1’

4.2 汇编语言伪指令

符号定义伪指令

LABEL类型定义伪指令

变量定义伪指令

段定义伪指令SEGMENT/ENDS

假定伪指令ASSUME

置汇编地址计数器伪指令ORG

源程序结束伪指令END

4.2 汇编语言伪指令

一、符号定义伪指令

1、等值伪指令EQU

格式： 符号名 EQU 表达式

功能：用EQU左边的符号名代表右边的表达式

例：
```
MOVE   EQU   MOV
CONST  EQU   60
STR    EQU   ″How are you!″
```

说明：

- 用EQU已定义的符号不能被重新定义，若需要，需用PURGE先解除。
- 用EQU给常量或表达式赋予一个符号名，方便在程序中的使

格式： PURGE 符号1，符号2，...，符号n

2、等号伪指令 =

功能与EQU类似,但允许重新定义

例：

```
        ⋮
EMP=7.1273941546        ；值为7.1273941546
        ⋮
EMP= 8 . 41546273       ；值为8.41546273
```

特别注意：

符号定义是定义在程序中要使用的常量。

用EQU、= 定义的符号不占用存储器单元。

即，汇编程序不为这样的符号分配存储空间。

二、变量定义伪指令

定义变量的伪指令也称为数据定义伪指令。

格式：变量名 伪指令助记符 操作数 ；注释

功能：定义变量的类型、名字，并为变量中的操作数项分配存储单元

解释：

(1)变量名由用户起名；

(2)伪指令助记符有如下几种：

DB（字节）、 DW（字）、DD（双字）、DF、DQ、DT

(3)操作数可以是常数或表达式；

(4)注释用来说明伪指令的功能，它亦可有可无。

表达式:

表达式项是给变量或指定存储单元赋予的初值，多个操作数之间须用逗号“,”分隔，具体有以下几种形式：

（1）数值表达式

（2）地址表达式（只适用DW和DD两个伪指令）

（3）字符串表达式

（4）？表达式

（5）带DUP的表达式

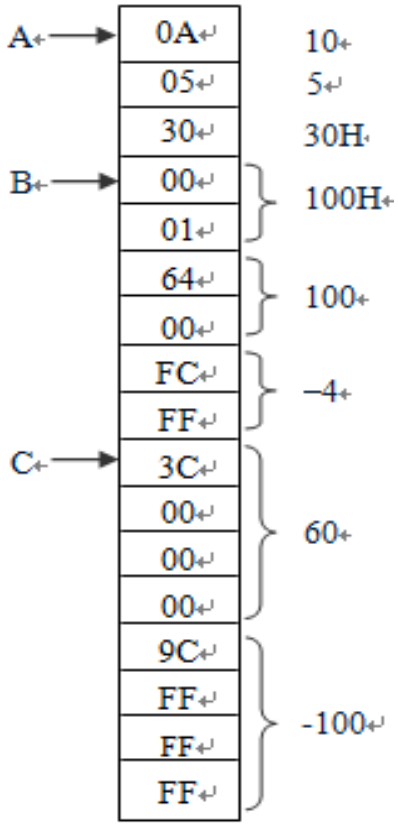
4.2 汇编语言伪指令

1、数值表达式

举例

```
Data segment
A   DB 10,5,30H
B   DW 100H,100,-4
C   DD 2*30,-100
Data ends
```

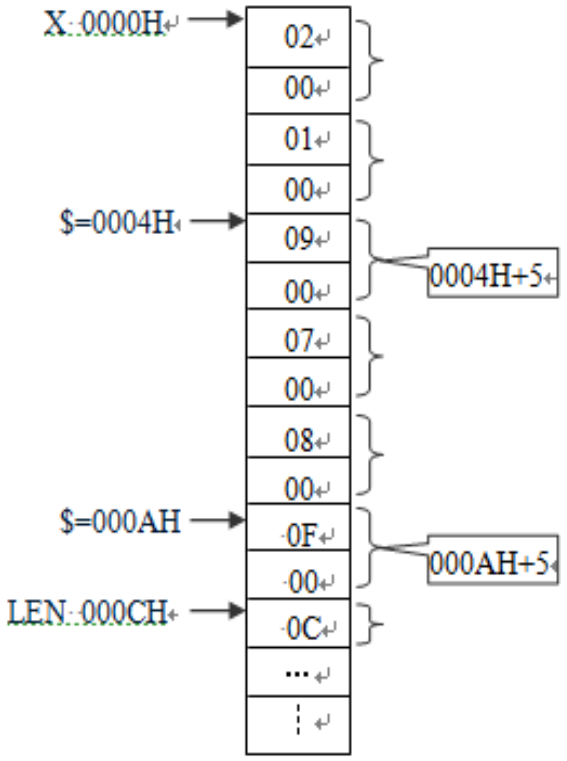
汇编后的内存分配如图所示：



2、地址表达式形式

```
DATA   SEGMENT
X DW 2,1,$+5,7,8,$+5
Len db $-x
DATA   ENDS
```

汇编后的内存分配如图所示：



解释：地址计数器$

在伪指令中，$表示地址计数器的当前值。

地址计数器：每进入一个新段，地址计数器清零；每分配一个单元，地址计数器自动加1，指向下一个待分配的单元；$代表当前值

3、字符串表达式形式

```
DATA   SEGMENT
STR1 DB  ‘12’
STR2 DB  ‘HELLO!’
STR3 DW  ‘CD’, ‘3’
DATA ENDS
```

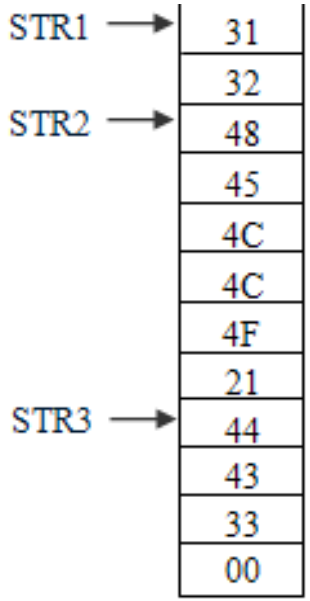


图4.3 STR1、STR2及STR3变量分配示意图

汇编后的内存分配如图所示：

解释： STR3 DW ‘CD’, ‘3’

注意：3个及以上的字符，只能用DB定义。

例： str1 DW ‘abcd’

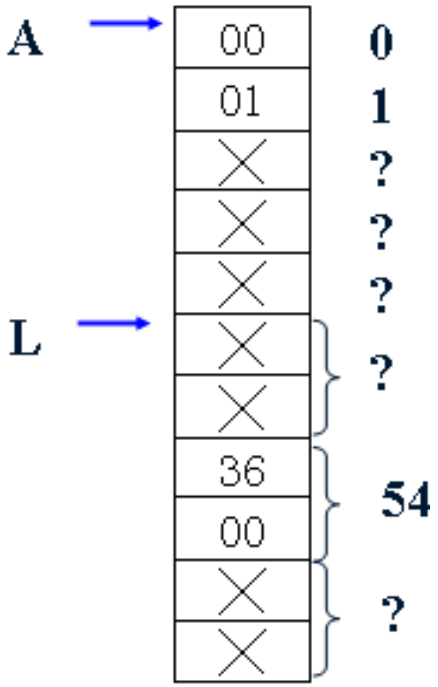
str2 DD ‘**abcd**’

4、？表达式形式

操作数是“?”，只用来保留内存空间，但不存入数据，即初始值未定义。

例：
```
A  DB   0,1,?,?,?
L  DW   ?,54,?
```

汇编后的内存分配如图示：



5、DUP表达式形式

格式：n DUP(表达式1,表达式2,…,表达式n)

功能：利用给出的一个或一组初值来重复地初始化存储单元。

n 为重复次数，括号中的内容为被重复的内容。注意： n与DUP之间需有空格

示例：ABC DB 0,1,?,?,?

亦可写成:

ABC DB 0,1,3 DUP(?)

DUP还可以嵌套，以下DAT变量的内存分配示意图如图所示

DAT DW 2 DUP(10H,2 DUP(1,2))

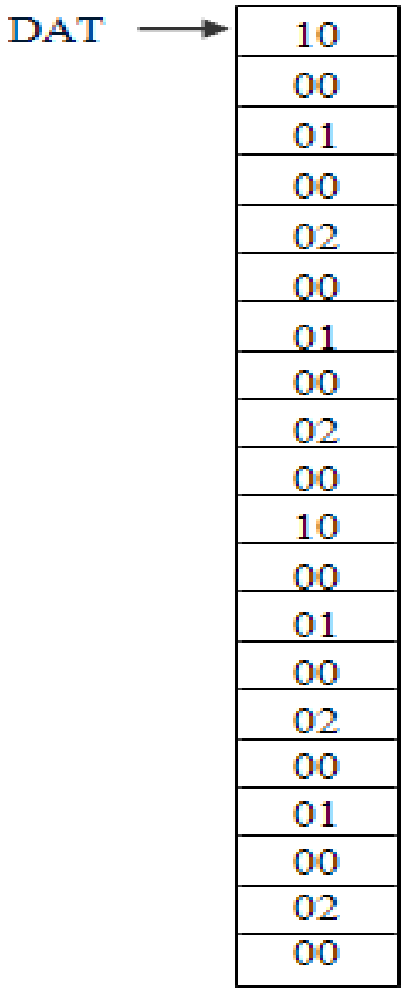


图4.4 DAT变量分配示意图

三、段定义伪指令

1、段定义伪指令SEGMENT/ENDS

格式：

段名 SEGMENT［定位类型］［组合类型］［’类别’］

⋮

段名 ENDS

解释：

SEGMENT和ENDS两个伪指令总是成对出现，二者前面的段名一致,由用户起名。SEGMENT和ENDS语句之间的内容可以是指令和其他伪指令，表示存放在该段内存中的变量、指令或其他伪指令对该段内存的处理。代码段中主要是指令。为了阅读方便，习惯上总是根据段体的性质起段名，譬如通常用DATA作为数据段段名，用CODE作为代码段段名

2.指定段寄存器伪指令ASSUME

格式：ASSUME 段寄存器名:段名[，段寄存器名:段名……]

功能：建立段寄存器与段之间关系。即指出某个段的段地址在哪个段寄存器中。如：

```
ASSUME CS：CODE，DS：DATA
```

说明：ASSUME伪指令只是告知汇编程序有关段寄存器与段的关系，并没有给段寄存器赋予实际初值。

CS、IP的初始值由伪指令“END 起始标号”装入；

堆栈段定义时若用了STACK，系统会自动初始化SS、SP

通常在程序中仅需对DS、ES段寄存器作初始化，通过以下指令：

```
MOVAX,DATA
MOVDS,AX          ；实现将段地址装入数据段
MOVES,AX          ；寄存器和附加段寄存器
```

3、置汇编地址计数器伪指令ORG

格式：ORG　数值表达式

功能：将数值表达式的值赋给汇编地址计数器。

ORG规定了段内的起始偏移地址，表达式的值即为段内的起始偏移地址，从此地址起连续存放程序或数据。数值表达式的值须为0--65535之间的非负整数。

例，给汇编地址计数器赋值。

```
DATA    SEGMENT
ORG     10                  ；置$值为10
VAR1    DW   100H，200H
ORG     $+5                 ；置$的值为14+5，即为19
VAR2    DB   ‘ABC’
DATA    ENDS
```

四、其它伪指令

1、LABEL类型定义伪指令

格式：名字　LABEL　类型

功能：为要使用的变量或标号定义一种新类型或修改原类型属性 。通常与数据定义伪指令连用，其功能类似语句“变量名或标号 EQU THIS 类型”。

解释：“名字”可以是变量或标号，当为变量时，类型是BYTE、WORD、DWORD等，当为标号时，类型是NEAR或FAR。

举例：
```
data segment
ba label byte
wa dw  100 dup(?)
data ends
```

定义了地址相同、类型不同的两个变量：字节类型变量ba与字类型变量wa。

```
Mov wa,AX
Mov ba,AL
```

2、源程序结束伪指令END

格式：END ［启动地址］

功能：标志整个程序的结束，是源程序的最后一条语句。

注意：

源程序中必须有 **END** 结束语句。

启动地址可是一个标号或过程名，指示程序的入口。

程序装入内存后，系统跳转到程序的入口处，开始执行。

4.4 DOS系统功能调用

DOS准备了许多程序（称为系统功能程序），涉及设备驱动和文件管理等方面的操作。系统功能调用是DOS为系统程序员及用户提供的一组常用子程序，对这些子程序的直接调用可以减少程序员对系统硬件环境的依赖，从而可以大大精简应用程序的编写；另一方面也可以使程序具有较好的通用性

4.4 DOS系统功能调用

一、系统功能调用方法

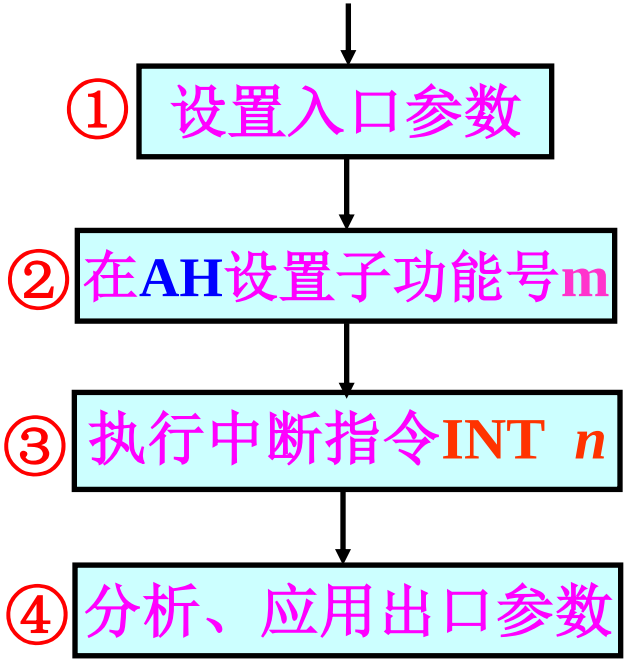
DOS系统功能调用的方法一般可分为以下几步：

① 设置所要调用功能的入口参数。

② 在AH寄存器中存入所要调用功能的功能号。

③ 执行INT 21H指令，自动转入中断子程序入口。

④ 相应中断子程序运行完毕，按规定获得出口参数。



二、常用的DOS系统功能调用

1. 带回显的键盘单字符输入（1号功能）
2. 不带回显的键盘单字符输入（8号功能）
3. 单字符输出（2号功能）
4. 单字符输入或显示（ 06H功能 ）
5. 字符串显示（9号功能）
6. 字符串输入（0AH号功能）
7. 程序正常返回系统（4CH号功能）

4.4 DOS系统功能调用

1、1号系统功能调用（从键盘读入一个字符）

功能：单字符输入。

将从键盘输入的字符的ASCII码值送AL寄存器，同时回显在屏幕上

入口参数：无

出口参数：AL寄存器存放输入字符的ASCII码值

格式：

```
MOV AH，1
INT 21H
```

说明：输入一个字符后，不需要输入回车键。

2、2号系统功能调用（显示一个字符）

功能：单字符输出。即将以DL寄存器的内容为ASCII码值的ASCII字符输出到屏幕上。

入口参数：DL＝要显示的字符（或其ASCII码值）

出口参数：无

格式：MOV DL，要显示的字符
MOV AH，2
INT 21H

例：MOV DL,41H ；41H=‘A’
MOV AH,2
INT 21H
屏幕上输出字符‘A’

说明：掌握回车符和换行符的使用。

2、2号系统功能调用（显示一个字符）

说明：掌握回车符和换行符的使用。

例：使光标回到下一行的行首。

```
MOV   DL , 0DH     ;显示回车符
MOV   AH , 02H
INT   21H
MOV   DL , 0AH     ;显示换行符
MOV   AH , 02H
INT   21H
```

3、单字符输入或显示 (06H功能)

06H：**入口参数** DL = 0FFH（输入）

或

DL≠0FFH(输出),其中是要显示字符的ASCII码

出口参数 若有输入（ZF = 0），AL = 输入的字符；

否则（ZF = 1），AL = 0。

实现功能 输入单个字符或显示指定的字符。

示例：

```
MOV   DL，0FFH  ;输入
MOV   AH，6
INT   21H
```

```
MOV   DL，24H   ;将 '$'输出
MOV   AH，6
INT   21H
```

01H与06H的区别：前者等待用户键入，后者不等待用户键入。

4.4 DOS系统功能调用

4、9号系统功能调用（显示一个字符串）

功能：字符串输出。即将DS:DX所指的以$字符结尾的一个字符串输出到屏幕上。

入口参数： DS:DX＝要显示的字符串在内存中的首地址

出口参数：无

格式：LEA DX，字符串变量名

```
MOV AH，9
INT 21H
……
```

说明：要显示的字符串必须以“$”作为结束标志，但‘$’不被显示

例：str DB ‘HELLO world!’，‘$’

```
……
LEA DX，str
MOV AH，9
INT 21H
```

5、10号系统功能调用（从键盘读入一个字符串）

功能：从键盘读入一个字符串，将其保存到DS:DX指定的内存缓冲区中。这个缓冲区由3部分组成：

- ✓第一字节定义缓冲区大小；
- ✓第二字节用于系统回填实际输入的字符个数；
- ✓从第三字节开始保存键盘输入字符的ASCII码值。

入口参数：DS：DX＝存放该字符串的输入缓冲区的首地址

出口参数：实际输入字符个数保存在缓冲区第二字节的位置（不包括回车符），实际输入字符的ASCII码值（包括回车0DH）顺序保存在缓冲区第三字节开始的位置。

说明：(1)可输入的字符个数最多为缓冲区第一字节内容减1，最少字符个数为0。(2)输入以按“回车键”结束，并将回车字符（0DH）保存在输入字符的最后一个位置上

4.4 DOS系统功能调用

格式：LEA DX，缓冲区首地址；设DS已指向用户定义的数据段

```
MOV AH，10   ;或0AH
INT 21H
```

调用方法示例：

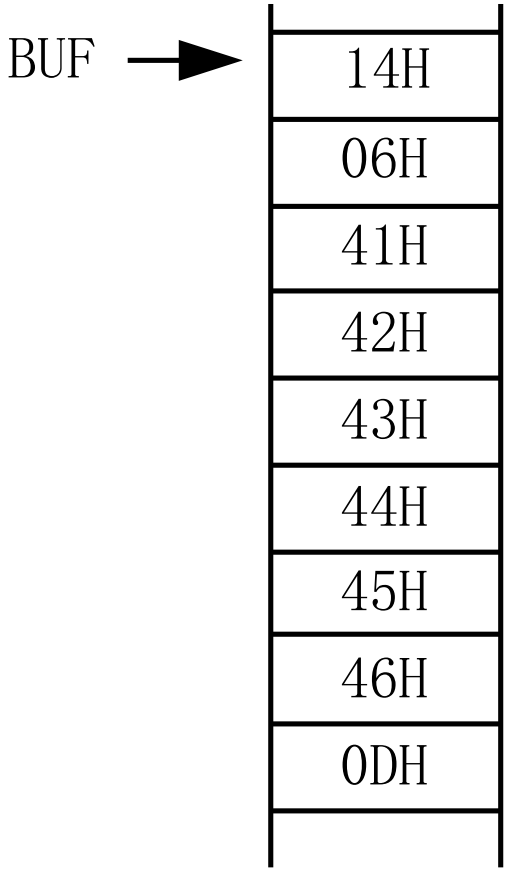
```
……
BUF DB 20   ;定义缓冲区大小，实际可输入字符最多20-1个
    DB ?      ;存放实际输入的字符个数，由系统自动设置
    DB 20 DUP（？）    ；存放实际输入的字符串
……
```

```
LEA DX，BUF
MOV AH，10
INT 21H
```

该指令序列将从键盘读入字符串(长度<20)，并存入输入缓冲区buf中

4.4 DOS系统功能调用

程序执行时等待键盘输入，一旦按“回车键”，输入字符及统计信息保存在缓冲区指定位置。假设输入：“ABCDEF回车”，内存分配如图所示：



6、程序正常返回系统的方法（4CH号系统功能调用）

功能：返回系统

入口参数：AL=终止代码或无

格式：

```
MOV AH，4CH
INT 21H
```

或

```
MOV AX，4C00H
INT 21H
```

4.5 汇编语言程序设计

一、顺序程序设计
二、分支程序设计
三、循环程序设计
四、中断功能调用的程序设计
五、子程序设计

汇编语言同高级语言一样，源程序的设计也有4大基本结构形式（顺序结构、分支结构、循环结构和子程序）。在实际的汇编程序设计中，单纯的一种结构程序并不多见，大多数都是多种结构的组合

一、顺序结构程序设计

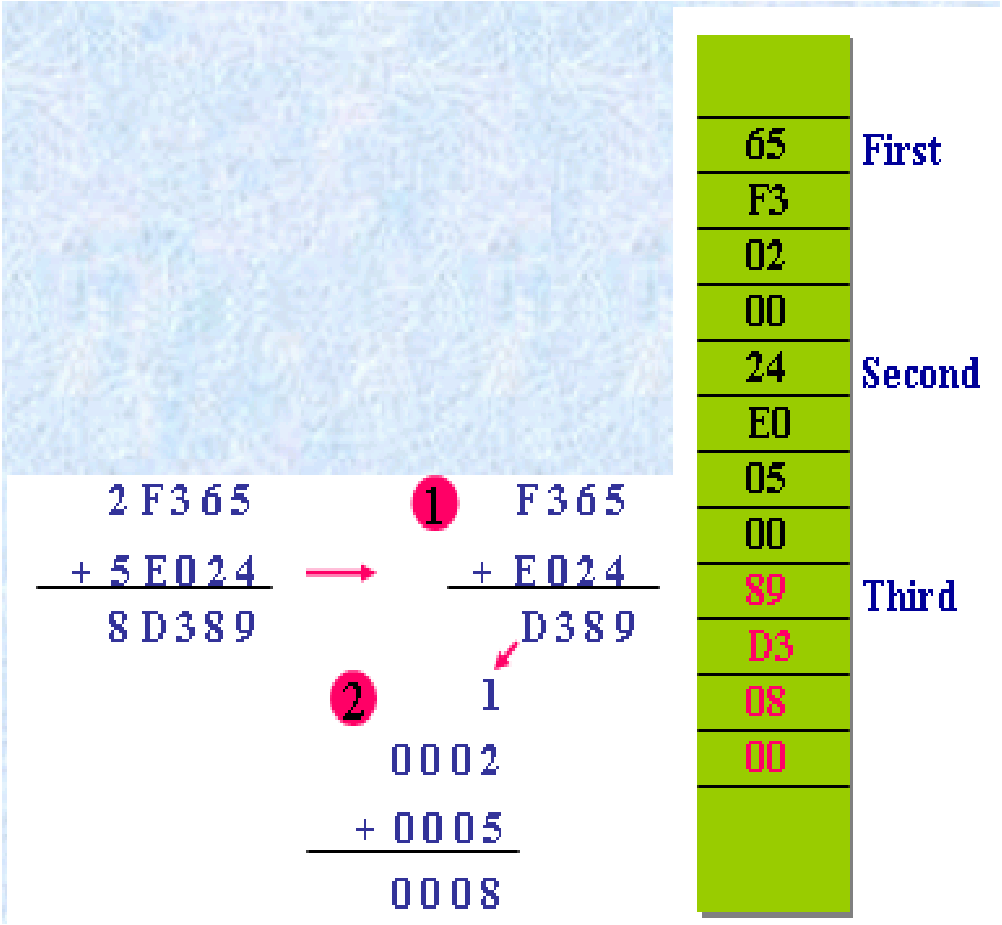
顺序程序完全按指令书写的前后顺序执行每一条指令，是最基本、最常见的程序结构

编程主要用到数据传送类指令、算术运算类指令、逻辑运算和移位类指令

4.5 汇编语言程序设计

示例：算术指令应用例子----两双字的二进制数加法。在内存的 First 和 Second 开始的区域中分别存放着 2F365H 和 5E024 H 两个数，要求求其和，并存入 Third 中。

```
MOV AX, First
ADD AX, Second
MOV Third, AX
MOV AX, First+2
ADC AX, Second+2
MOV Third+2, AX
```



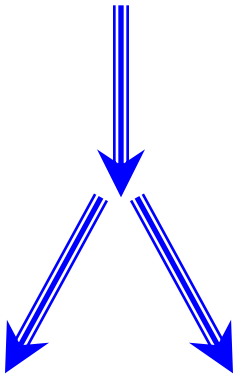
4.5 汇编语言程序设计

示例：算术运算指令例子

```
DATA   SEGMENT
First DW   0F365H，0002H
Second DW  0E024 H，0005 H
Third DW  2 DUP（?）
DATA   ENDS
CODE   SEGMENT
        ASSUME   CS：CODE，DS：DATA
START:
        MOV   AX，DATA
        MOV   DS，AX
        MOV AX, First
        ADD AX, Second
        MOV Third, AX
        MOV AX, First+2
        ADC AX, Second+2
        MOV Third+2, AX
        MOV   AH，4CH
        INT   21H
CODE   ENDS
        END   START
```

二、分支结构程序设计

分支程序根据条件是真或假决定执行与否，判断的条件是各种指令，如CMP、TEST等执行后形成的状态标志与条件转移指令Jcc联合可以实现分支控制

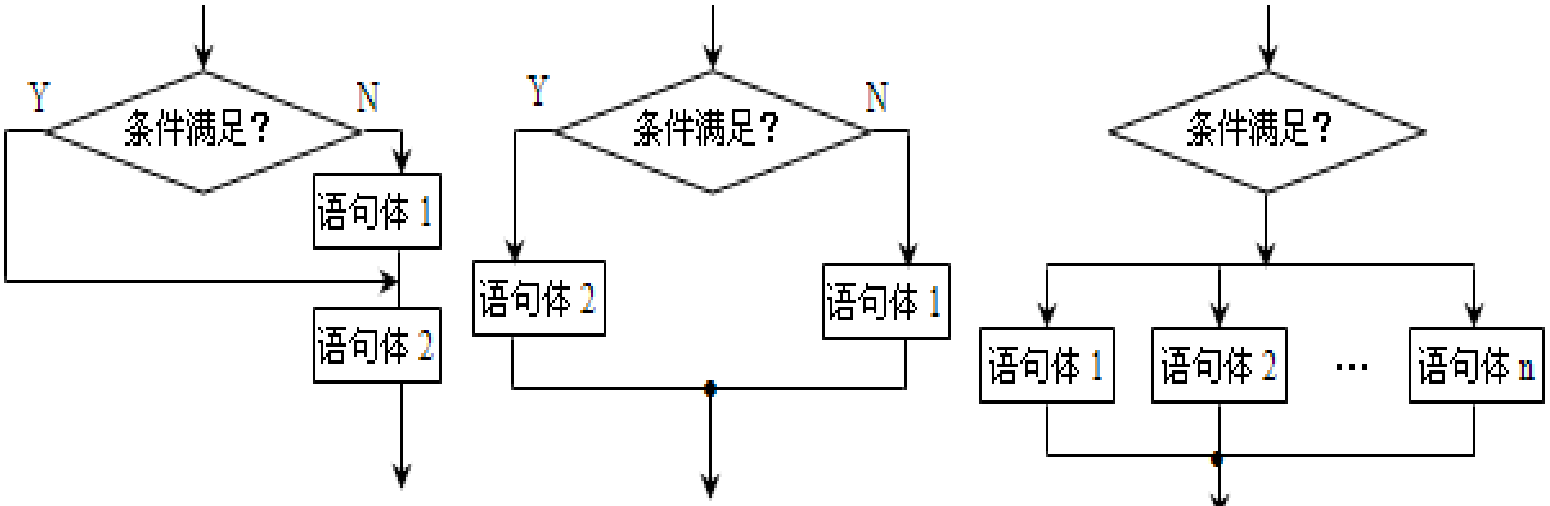


用无条件转移指令JMP也可以实现分支控制。

汇编语言中的分支通常有三种形式：

- ✓IF-THEN 型：也称单分支结构
- ✓IF-THEN-ELSE 型：也称双分支结构
- ✓DO-CASE 型：也称多分支结构

4.5 汇编语言程序设计



（a）IF-THEN 结构　（b）IF-THEN-ELSE 结构　（c）DO-CASE 结构

图 4.10 分支程序的结构形式

掌握：分支程序设计要领

示例：单分支程序设计例子。编程求｜X－Y｜，
结果存于RESULT单元中，设X和Y都为16位数据。

```
DATA  SEGMENT
X       DW  -10
Y       DW  20
RESULT DW ?
DATA  ENDS
STACK SEGMENT STACK
            DB 100 DUP(?)
STACK ENDS
CODE  SEGMENT
          ASSUME CS:CODE,DS:DATA,SS:STACK
START:
          MOV AX,DATA
          MOV DS,AX
          MOV AX,X
          SUB AX,Y
          JNS NONNEG
          NEG AX
NONNEG: MOV RESULT,AX
          MOV AH,4CH
          INT 21H
CODE    ENDS
          END START
```

4.5 汇编语言程序设计

示例：双分支程序设计。编写一程序，显示 BX寄存器中所存数据的最高位。即若BX的最高位为1，则在屏幕上显示“1”；否则显示“0”

```
CODE   SEGMENT
       ASSUME  CS:CODE
START:SHL  BX,1
       JC  ONE
        MOV  DL,'0'
       JMP  ZERO
ONE:   MOV  DL,'1'
ZERO:  MOV  AH,2
        INT  21H
       MOV  AH,4CH
       INT  21H
CODE   ENDS
       END  START
```

示例：多分支程序设计。编写计算以下函数值的程序，设X和Y都为有符号字变量（不考虑溢出情况）

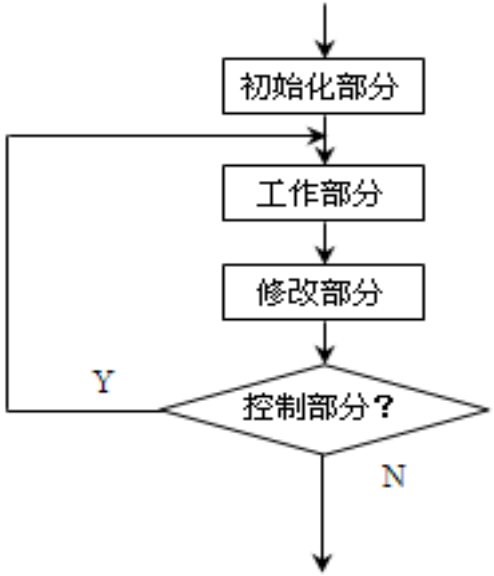
$$Y=\begin{cases} X+20 & X<0 \\ X*20 & 0\le X\le 10 \\ X-80 & X>10 \end{cases}$$

```
DATA   SEGMENT
X      DW  -10
Y      DW  ?,?
DATA   ENDS
STACK SEGMENT  STACK
DB  200  DUP(0)
STACK  ENDS
CODE   SEGMENT
       ASSUME  DS:DATA,SS:STACK,CS:CODE
START:MOV  AX,DATA
       MOV  DS,AX
       XOR  AX,AX
       XOR  DX,D
       MOV  AX,X
       CMP  AX,0
```

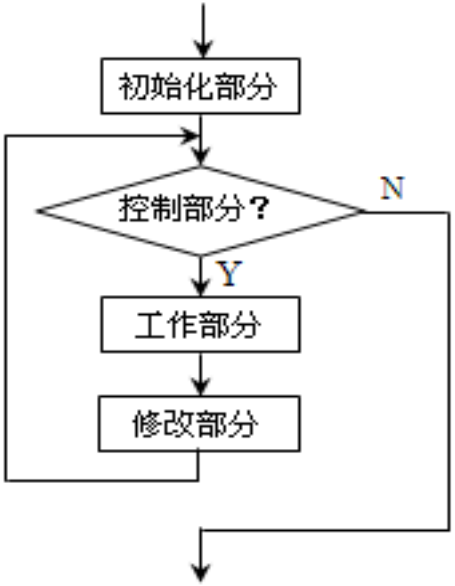
```
            JGE  CASE23
            ADD  AX,20
            JMP  RESULT
CASE23:CMP  AX,10
            JG   CASE3
            MOV  BX,20
            IMUL  BX
            JMP  RESULT
CASE3:  SUB  AX,80
RESULT: MOV  Y,AX
            MOV  Y+1,DX
            MOV  AH,4CH
            INT  21H
CODE    ENDS
            END  START
```

三、循环程序设计

1、循环程序的结构形式



（a）DO-UNTIL形式



（b）DO-WHILE形式

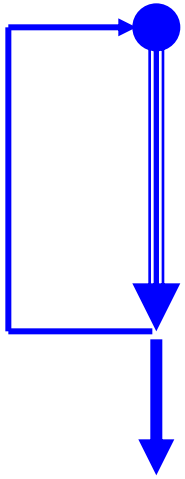


图 4.11 循环结构的流程

按层次分为单重循环和多重循环

按循环次数是否已知，通常分为计数控制与条件控制

2、循环结构程序设计举例

示例：计数控制。计算1＋2＋3＋…＋100的和,结果送 RSLT单元

```
DATA SEGMENT
      RSLT  DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START:
        MOV AX,DATA
        MOV DS,AX
        MOV AX,1
        MOV RSLT,0
        MOV CX,100
        CLC
NEXT:
        ADC RSLT,AX
        INC AX
        LOOP NEXT
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

4.5 汇编语言程序设计

示例：条件控制循环。统计BUF字单元中数据中含有1的个数，将结果存入COUNT字节单元中。

```
DATA    SEGMENT
BUF     DW 2345H
COUNT   DB 0
DATA    ENDS
STACK   SEGMENT STACK
        DB 100 DUP(?)
STACK   ENDS
CODE    SEGMENT
ASSUME CS:CODE,DS:DATA,SS:STACK
START:
        MOV AX,DATA
        MOV DS,AX
        MOV AX,BUF
        MOV CL,0
COPA:   AND AX,AX
        JZ EXIT
        SHL AX,1
        JNC COPA
        INC CL
        JMP COPA
EXIT:   MOV  COUNT,CL
        MOV AH,4CH
        INT 21H
CODE    ENDS
        END START
```

示例：多重循环程序设计。在以BUF为首址的字存储区中存放有N个带符号数据，编程实现将它们按由大到小的顺序排列后再存放回原存储区中

```
DATA    SEGMENT
BUF     DW 30，-44，62，57，19，23，0，-8，-9，-10，20
N=（$-BUF）/2
DATA    ENDS
STACK   SEGNMENT STACK
        DB 200 DUP（0）
STACK   ENDS
CODE    SEGMENT
        ASSUME CS：CODE，DS：DATA，SS：STACK
START：  MOV AX，DATA
        MOV DS，AX
        MOV CX，N
        DEC CX
```

```
NEXT1：MOV  DX，CX                ;趟数送dx中
       MOV  BX，0                 ;BX为基地址
NEXT2：MOV  AX，BUF[BX]           ;每一趟中比较的次数
       CMP  AX，BUF[BX+2]
       JGE  L
       XCHG AX，BUF[BX+2]
       MOV  BUF[BX]，AX
L：    ADD  BX，2
       DEC  CX
       JNE  NEXT2
       MOV  CX，DX
       LOOP NEXT1                 ;趟数的比较
       MOV  AH，4CH
       INT  21H
CODE   ENDS
       END  START
```

四、中断功能调用程序设计

1、1号系统功能调用应用

例：从键盘上读入一串指定长度的字符
(要求：用1号系统功能调用实现)

```
DATA SEGMENT
HL      DB 12 DUP(0)
DATA ENDS
CODE  SEGMENT
        ASSUME CS: CODE,DS: DATA
GO:    MOV AX, DATA
        MOV DS,A
        MOV CX,12
        MOV SI,OFFSET HL
L1:     MOV AH,1
        INT 21H
        MOV [SI],AL
        INC SI
        LOOP L1
        MOV AH,4CH
        INT 21H
CODE   ENDS
        END GO
```

1号与2号系统功能综合应用

例：试编写一个汇编语言程序，要求对键盘输入的小写字母用大写字母显示出来。（要求：可连续输入）

```
CODE    SEGMENT
        ASSUME  CS:CODE
BEGIN:  MOV AH, 1
        INT 21H
        CMP AL,' a'
        JB BEGIN
        CMP AL,' z'
        JA BEGIN
        SUB AL, 20H
        MOV DL, AL
        MOV AH, 2
        INT 21H
        JMP BEGIN
STOP:   MOV AH,4CH
        INT 21H
CODE    ENDS
        END  BEGIN
```

接受键盘输入的单个字符，并放入AL中

输出字符

示例：9号系统功能应用

```
DATA    SEGMENT
STR     DB 0DH,0AH,' How are you!$'
DATA    ENDS
STACK   SEGMENT STACK
        DB  100 DUP(0)
STACK   ENDS
CODE    SEGMENT
        ASSUME DS:DATA,CS:CODE,SS:STACK
BEGIN:  MOV AX,DATA
        MOV DS,AX
        LEA DX,STR
        MOV AH,9
        INT 21H
        MOV AH,4CH
        INT 21H
CODE    ENDS
        END BEGIN
```

（4） 10号系统功能调用应用。从键盘上读入一串指定长度的字符，然后利用9号系统功能调用显示输出该串字符。

```
 data   segment
message  db 'please input a string:$'
buf     db 50,?,50 dup(0)
data  ends
code segment
    assume cs:code,ds:data
go:    mov ax,data
        mov ds,ax
        mov dx,offset message
        mov ah,9
        int 21h
        mov dx,offset buf
        mov ah,10
        int 21h
        mov ah,2
        mov dl,0ah
        int 21h
        mov dl,0dh
        int 21h
        mov bl,buf+1
        mov bh,0
        mov byte ptr buf+2[bx],'$'
        mov dx,offset buf+2
        mov ah,9
        int 21h
        mov ah,4ch
        int 21h
code  ends
      end go
```

五、子程序设计

子程序是程序中功能相对独立的一段程序，子程序设计是程序设计中被广泛使用的一种方法。子程序设计的使用方式较多，运用十分灵活，涉及面较广，需掌握基本方法。在汇编语言中，子程序又称过程。

使用子程序的好处：
可供一个或多个程序使用；
可以简化源程序结构；
提高程序的可读性与可维护性；
有利于代码复用；
提高程序的设计效率。

1、子程序的定义

格式：

```
子程序名  PROC [NEAR/FAR]
    ┆              ；过程体
子程序名  ENDP
```

解释：

- ✓ 子程序名：子程序入口地址的符号表示。同标号一样，具有段属性、偏移地址属性及类型属性三种属性
- ✓ **PROC表示子程序定义开始，ENDP表示子程序定义结束**
- ✓ 类型NEAR/FAR

2、子程序的调用和返回

调用CALL指令

返回RET指令

```
SUBP  PROC
      PUSH  AX        ; 现场保护
      PUSH  BX
      PUSH  CX
        ┇             ; 子程序主体
      POP  CX         ; 恢复现场
      POP  BX
      POP  AX
      RET
SUBP  ENDP
```

强调：在使用时应注意的问题

3、子程序的书写形式和结构示例

```
code   segment                ;code段
                …
       call  subp             ;调用指令 ,xor指令的地址入栈
       xor ax,ax
                …
       mov ah,4ch
       int 21h
subp   proc  near             ;子程序定义
            …
       ret                    ;返回
subp   endp
code   ends
       end start
```

near 类型子程序结构示例

```
code    segment
start:      …
        call  sub
            …
        call  sub
            …
        mov  ah, 4ch
        int  21h
sub     proc
          、 、
          、 、
        ret
sub     endp
code    ends
        end start
```

多处调用完成同一功能的子程序结构示例

```
code    segment
begin:  call  sub1
        call  sub2
        call  sub3
        mov  ah, 4ch
        int    21h
sub1    proc
          、、
         ret
sub1    endp
sub2    proc
          、、
        ret
sub2    endp
sub3    proc
          、、
         ret
sub3    endp
code    ends
        end     begin
```

多个子程序的调用示例

子程序的位置通常在主程序的所有可执行指令之前或之后，不能放在主程序的可执行指令序列内部，否则会破坏主程序结构

3. 参数传递

（1）参数传递概念

一个程序调用子程序时需要传递参数给子程序，这些参数是子程序所需的原始数据；子程序处理完后向调用它的程序传递处理结果，所传递的原始数据和处理结果可以是数据、地址。调用程序和子程序之间所传递的信息统称为参数传递

（2）入口参数与出口参数

入口参数（入口条件）：主程序调用子程序前，为子程序内部数据处理准备所需的预置值；

出口参数（出口条件）：子程序返回主程序后，子程序处理的结果传递给主程序的数据。

（3）参数传递方法

✓寄存器法: 通过CPU寄存器传递。传递数据方便、快捷，但所能传递的数据长度和个数都有限。

✓变量法: 通过内存单元传递参数。传递数据的大小和个数不受限制，程序设计比较灵活。

✓堆栈法: 通过堆栈传递参数。用堆栈保存所要传递的数据或存储地址，是常用的参数传递方法。

4．子程序的嵌套与递归调用

子程序嵌套：子程序作为调用程序又去调用其它子程序。一般只要堆栈空间允许，子程序嵌套层数不限，但嵌套层数较多时应特别注意寄存器内容的保护与恢复，以避免各层子程序之间寄存器使用发生冲突，造成程序出错。

子程序递归调用：当子程序直接或间接地嵌套调用自身时。

5、子程序设计举例

（1）三种传递方法举例。分别用三种参数传递方法编写求1＋2的和的程序。要求将结果送到内存单元，并显示。

☞通过寄存器传送

```
data    segment
sum     db 0
data    ends
stack   segment
        db   100  dup(?)
stack   ends
code    segment
        assume  ds:data,ss:stack,cs:code
start:  mov ax,data
        mov ds,ax
        mov al, 1
        mov bl, 2
        call subprog
        mov ah,4ch
        int 21h
code    ends
        end     start
```

```
subprog  proc
         add al, bl
         or al, 30h
         mov sum, al
         mov dl,al
         mov ah,2
         int 21h
         ret
sub      endp
```

☞通过变量传送

```
data    segment
sum     db 0
d1      db ?
d2      db ?
data    ends
stack   segment
        db   100  dup(?)
stack   ends
code    segment
        assume  ds:data,ss:stack,cs:code
start:  mov ax,data
        mov ds,ax
        mov d1, 1
        mov d2, 2
        call sprog
        mov ah,4ch
        int 21h
code    ends
        end     start
```

```
sprog   proc
        mov al, d1
        add al, d2
        or al, 30h
        mov sum, al
        mov dl,al
        mov ah,2
        int 21h
        ret
sprog   endp
        end  start
```

☞通过堆栈——功能最强/最灵活/最复杂

```
DATA    SEGMENT
SUM      DB 0
DATA    ENDS
STACK   SEGMENT
         DB   100  DUP(?)
STACK   ENDS
CODE    SEGMENT
         ASSUME  DS:DATA,SS:STACK,CS:CODE
START:  MOV AX,DATA
        MOV DS,AX
        MOV AL, 1
        MOV BL, 2
         MOV AH,0
        MOV BH,0
        PUSH AX
        PUSH BX
        CALL SPR
        POP BX
        POP AX
         MOV AH,4CH
         INT 21H
CODE    ENDS
END     START
```

```
Spr     PROC
        PUSH BP
        MOV BP, SP
        MOV AX, [BP+6]
        MOV BX, [BP+4]
        ADD AL, BL
        OR AL, 30H
        MOV DL,AL
        MOV AH,2
        INT 21H
        MOV SUM, AL
        POP BP
        RET
Spr     ENDP
```

（2）递归调用举例。编程计算N!（N≥0）

$$N!=\begin{cases} N(N-1)! & N>0 \\ 0 & N=0 \end{cases}$$

```
data   segment
n      dw  3
result  dw  2  dup(?)
data   ends
stack segment stack
       dw 100 dup(?)
stack ends
code  segment
       assume  cs:code,ds:data,ss:stack
start:mov  ax,data
       mov  ds,ax
       mov  bx,n
       push  bx                    ; 入口参数n入栈
       call  fact                  ; 调用递归子程序
       pop  result                 ; 出口参数：n!
       mov  ah,4ch
       int  21h
```

```
FACT    PROC
        PUSH  AX
        PUSH  BP
        MOV  BP,SP
        MOV  AX,[BP+6]          ; 取入口参数 N
        CMP  AX,0
        JNE  FACT1              ; N＞0,N!＝N×(N-1)!
        INC  AX                 ; N＝0,N!＝1
        JMP  FACT2
FACT1:DEC  AX                       ; N-1
        PUSH  AX
        CALL  FACT              ; 调用递归子程序求(N-1)!
        POP  AX
        MUL  WORD  PTR [BP+6]   ; 求 N×(N-1)!
FACT2:MOV  [BP+6],AX            ; 存入出口参数 N!
        POP  BP
        POP  AX
        RET
FACT    ENDP
CODE   ENDS
       END  START
```