第六章　8086/8088汇编语言程序设计

6-0 汇编语言介绍

6-1 汇编语言程序格式

6-2 MASM中的表达式

6-3 伪指令语句

6-4 DOS和BIOS中断调用

6-5 程序设计方法

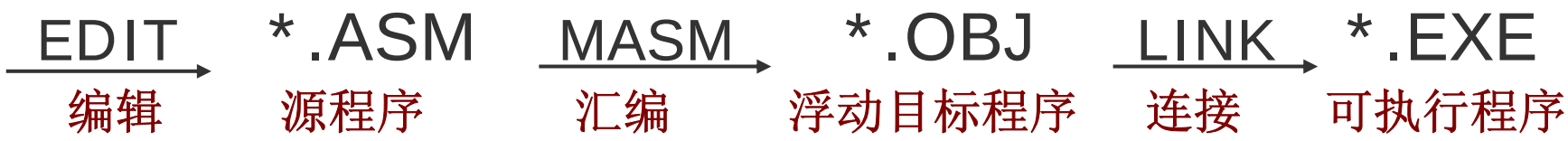
6-0 汇编语言介绍

汇编语言是一种面向机器的程序设计语言，不同类型的CPU，其汇编指令也不尽相同。它是对机器语言的符号化描述，是一门低级语言。

用汇编语言编写的程序叫“汇编语言程序”或“汇编语言源程序”.这种程序较机器语言直观、易懂、便于交流和维护。与其他高级语言一样，汇编语言源程序不能直接被计算机识别并运行，它必须通过汇编程序翻译成机器能够识别的机器语言程序(目标程序)才能运行。

汇编程序是系统程序，是用来将用户编写的汇编语言(源)程序转换为机器代码的系统工具程序。

利用汇编程序将汇编语言源程序翻译成机器代码的过程称为汇编。



编译程序和连接程序我们目前使用的通常有两种：

1、MicroSoft公司的 MASM.EXE 和 LINK.EXE。

2、Borland 公司的 TASM.EXE 和 TLINK.EXE。

汇编语言特点：

- 面向机器，与硬件紧密相关。
- 编制的程序简洁，高效，实时性好，节省内存，运行快。

汇编语言程序格式:

例 在屏幕上显示并打印字符串“**This Is a sample program.**”

```
DATA    SEGMENT                                  ; 数据段
        DA1   DB   'This Is a sample program.'
              DB   0DH,0AH, '$'
DATA    ENDS
STACK   SEGMENT
        ST1  DB   100 DUP(?)
STACK   ENDS
CODE    SEGMENT                                  ;代码段
        ASSUME CS:CODE,DS:DATA,SS:STACK
MAIN    PROC  FAR
START:  MOV AX,STACK                             ;送堆栈段段地址
        MOV SS,AX
```

```
        PUSH DS                          ；返回DOS用
        MOV AX,0
        PUSH AX
        MOV AX,DATA                      ;送数据段段地址
        MOV DS,AX
        MOV AH,9          ；DOS 9号功能调用，显示字符串
        MOV DX,OFFSET DA1
        INT 21H
        RET
MAIN    ENDP
CODE    ENDS
        END START
```

6-1　汇编语言程序格式

- 一个汇编语言程序可由若干个(此例有3个)段组成。每个段均以SEGMENT开始以ENDS结束。
- 每个段可以包含若干语句，而语句又可以是指令语句或伪指令语句两种。如程序中用到的ASSUME，DB，SEGMENT…ENDS等为伪指令语句。
- 每条语句可由标识符、保留字、表达式等元素组成。

一、 指令性语句

指令性语句与机器指令相对应，汇编程序可将它翻译成目标代码。语句格式为：

标号：指令助记符 操作数，操作数；注释

标　号：符号地址，后跟“：”，使用英文大小写字母阿拉伯字

母和特殊字符，第一个非数字，≦31个

助记符：与指令对应，不可省。

操作数：参加运算的数据，用常数、变量、标号、寄存器名或

表达式。

注　释：说明 指令或程序功能 可省略 用“；”隔离

二、伪指令语句

伪指令语句无对应的机器指令，汇编程序汇编源程序时对伪指令进行处理，它可完成数据定义，存储区分配，段定义，段分配，指示程序结束等功能。语句格式为：

名字 伪指令指示符 操作数，操作数；注释

名　字：赋予伪指令的名称，名字后不允许带冒号“:”,可省略。名字常为变量名、段名、过程名、符号名等。

指示符：汇编程序MASM规定的符号，各种定义语句后详述。

操作数：按伪指令要求可有可无，操作数可以是常数、变量、字符串、表达式等。

注　释：功能和使用与指令性语句相同。

三、数据项—常数、变量和标号是三种基本数据项。

1. 常数

二进制(B)、八进制(Q或O)、十进制(D)、十六进制(H)、ASCII码字符串(用单引号‘abcd’)。

2. 变量

常指存放在M中的数值，程序运行中可修改。

三个属性：

- 段值(SEGMENT):变量所在段的段基址。
- 段偏移地址(OFFSET):变量地址与所在段首地址之间的地址偏移字节数。
- 类型(TYPE):变量中每个元素包含的字节数,有字节变量(BYTE), 字变量(WORD)及双字变量(DWORD)等

3．标号

指令语句的地址的符号表示，可作为JMP指令和调用指令CALL的目标操作数，以确定程序转向的目标地址。

三个属性：

- 段值(SEGMENT)：标号所在段的段基址。
- 段内偏移地址(OFFSET)：标号地址与所在段的段首址之间的偏移地址字节数
- 类型(TYPE)：指转移指令中标号可转移的距离。近标号(NEAR)，远标号(FAR)。

  NEAR—指针长度2字节

  FAR —指针长度4字节

6-2 MASM中的表达式

表达式 〈 运算对象 / 运算符 ⟶ 汇编运算 ⟶ 结果作为语句中操作数

运算对象：常数、变量、标号

运算结果：常数或存储器地址（变量或标号）

8086汇编语言中的运算符

算术运算符	逻辑运算符	关系运算符	分析运算符	综合运算符
+(加法)	AND(与)	EQ(相等)	SEG(求段基址)	PTR
-(减法)	OR(或)	NE(不相等)	OFFSET(求偏移量)	THIS
*(乘法)	XOR(异或)	LT(小于)	TYPE(求变量的类型)	SHORT
/(除法)	NOT (非)	GT(大于)	SIZE(求字节数)	HIGH
MOD(取余)		LE(小于或等于)	LENGTH(求变量长)	LOW
SHL(左移)		GE(大于或等于)		
SHR(右移)				

一 算术运算符

所有的算术运算符均可以对数据进行运算，运算对象与运算结果都是整数。若对地址运算，通常是在标号上加 / 减某一个数字量，对地址乘是没有意义的。

例 源程序指令格式如下:

```
DA  EQU  300
    MOV  AX,DA-80
    MOV  BX,DA MOD 100
    MOV  CX,DA/100
    MOV  DH,01100100B SHR 2
```

汇编时计算表达式形成指令为

```
DA  EQU  300
    MOV  AX,220
    MOV  BX,0
    MOV  CX,3
    MOV  DH,19H
```

二 逻辑运算符

逻辑运算符是按位运算的．只能对常数进行运算，得到结果也是常数。

逻辑运算符与8086指令系统中的指令助记符AND、OR、NOT、XOR符号完全相同， 但二者是不会混淆的。作为MASM的运算符是在汇编过程中进行计算的，而指令助记符是在程序执行时进行运算的。

例：AND DX，PORT AND 0FEH

指令助记符程序执行时运算

逻辑运算符汇编时计算产生一立即数

三 关系运算符

- 所连接两操作数必为两数据或同一段内的两个存储单元的地址
- 运算结果：真—0FFFFH，假—0
- 常与逻辑运算符组合使用

例

```
MOV AX,10H GT 16
ADD BL,6 EQ 0110B
MOV CX,((PORT LT 5) AND 100) OR ((PORT GE 5) AND 200)
```

汇编时形成指令为:

```
MOV AX,0
MOV BL,0FFH
MOV CX,100   ;PORT<5时
```

四 数值返回运算符（分析运算符）

操作符号	表达式	表达式含义
SEG	SEG 变量或标号	取出变量或标号的段基址
OFFSET	OFFSET 变量或标号	取出变量或标号的偏移地址
TYPE	TYPE 变量或标号	取出变量或标号的类型
LENGTH	LENGTH 变量	变量中使用DUP时返回变量包含的单元数， 其它变量返回1
SIZE	SIZE 变量	返回变量包含的总字节数 SIZE=LENGTH*TYPE

表 TYPE运算符返回值

	类型	返回值
变量	DB	1
	DW	2
	DD	4
	DQ	8
标号	NEAR	-1 [0FFH]
	FAR	-2 [0FEH]

例：

```
M1    DW 100 DUP(?)
M2    DW 1,2,3
M3    DB ‘ABCD’
L1:   MOV AH,TYPE M1
      MOV AL,TYPE M3
      MOV BH,TYPE L1
      MOV CX,LENGTH M1
      MOV BL,LENGTH M2
      MOV CX,SIZE M1
      MOV BL,SIZE M2
      MOV AL,SIZE M3
```

汇编时形成指令：

```
      MOV AH,2
      MOV AL,1
      MOV BH,0FFH
      MOV CX,100
      MOV BL,1
      MOV CX,200
      MOV BL,2
      MOV AL,1
```

五 修改属性运算符（综合或合成运算符）

1.段操作符

段前缀：变量或地址表达式

表示某个变量或地址被修改到哪个段寄存器提供的段基址中

如：

```
MOV AX,ES:[BX]
```

2.PTR

类型/距离　PTR　变量或标号

功能：将PTR左边的类型属性赋给右边的变量或标号。PTR本身并不分配存储单元，仅给已分配的存储单元赋予新的属性，这样可以保证运算时操作数类型的匹配，常与类型BYTE、WORD、NEAR、FAR等连用。

例：

```
DATA1   DB   10H,20H,30H          ; 数据定义
DATA2   DW   4023H,0A845H
……
MOV   BX, WORD  PTR DATA1    ; 2010H 传送到 BX
MOV   AL, BYTE  PTR DATA2    ; 23H 传送到 AL
MOV   WORD PTR[BX],10H       ;[BX],[BX+1]←0010H
```

3.THIS

变量/标号　EQU THIS 类型/距离

功能:将EQU THIS右边的类型/距离属性，赋给左边的变量/标号，该变量或标号的段地址和偏移地址与下一个存储单元的地址相同

例:

```
MY_BYTE   EQU THIS BYTE
MY_WORD   DW 1122H
    ......
MOV AL,MY_BYTE      ; 将22H传送给寄存器AL
MOV BX,MY_WORD      ; 将1122H传送给寄存器BX
```

例：THIS 操作符对于建立FAR属性是方便的

```
START EQU THIS FAR
      MOV CX,100
```

赋予传送指令（MOV）有一个FAR属性的地址START，允许其他段的转移指令（JMP）直接转移到START.

4.SHORT

格式：JMP　SHORT 标号

功能：指定转移的距离属性为短，实际转移范围为±127字节。

例：

```
                JMP  SHORT  NEAR_LABEL
                      ……
NEAR_LABEL:           ……
```

5.HIGH和LOW

分离字节运算符

HIGH——分离高字节

LOW ——分离低字节

例

```
K1    EQU  0ABCDH
K2    EQU  1234H
      MOV  AH,HIGH K1     ;AH←0ABH
      MOV  BL,LOW  K2     ;BL←34H
```

六 其它运算符

1.圆括号( )

圆括号用来改变运算符的优先级别，( )中的运算符具有最高优先权。

2.方括号[ ]

方括号主要用来表示地址表达式或多重变量的下标值。

```
M1    DB 10H,20H,30H,40H
M2    DW 1234H,5678H,9ABCH
M3    DW 5 DUP(?)
      MOV BX,OFFSET M1
      MOV CL,[BX]              ;CL←10H
      MOV BX,OFFSET M2
      MOV DX,[BX+2]            ;DX←5678H
      MOV AL,M1[3]             ;AL←40H
```

6-3 伪指令语句

一 数据定义语句

格式1：变量名 助记符 操作数，操作数，.. ;注释

格式2：变量名 助记符 n DUP (操作数，操作数，..);注释

助记符：DB 定义字节 DW 定义字

DD 定义双字 DQ 定义四字

DT 定义十字节

变量名—符号表示，可省略。作其后第一字节符号地址。

操作数—常数，字符串，变量，标号，表达式

n DUP() —n为整数，表示括号中操作数重复次数。

注意：

① 定义多字节字符串用**DB**，**DW**只允许包含两个字符。

② 操作数用**?**定义不确定值变量，以保留存储空间存放运算结果。

③ 用**DW**和**DD**可以将变量或标号逻辑地址存入存储器。当用**DD**来定义时，原变量或标号的偏移地址存入低位字中，原变量或标号的段基址存入高位字中。

例：

```
DATA    SEGMENT
        DA1    DB 10H
        DA2    DW 1122H
        DA3    DD 0A0H
        ST1    DB ‘How ’
        ST2    DB ‘OK’
        ST3    DW ‘OK’
        M      DW  2  DUP(?)
        ADR1   DW  ST1
        ADR2   DD   ST2
DATA    ENDS
```

变量	MEMORY	EA
DA1→	10H	0000
DA2→	22H	0001
	11H	0002
DA3→	A0H	0003
	00H	0004
	00H	0005
	00H	0006
ST1→	48H	0007
	4FH	0008
	57H	0009
ST2→	4FH	000A
	4BH	000B
ST3→	4BH	000C
	4FH	000D
M→	?	000E
	?	000F
	?	0010
	?	0011
ADR1→	07H	0012
	00H	0013
ADR2→	0AH	0014
	00H	0015
	DATA	0016
		0017

二　表达式赋值语句

1 赋值语句EQU

格式：符号　**EQU**　表达式

功能：用来给变量，标号，常数，指令，表达式等定义一个符号名，在同一个程序模块中不能重新定义。

```
A  EQU  7          ；将7赋予符号名 A
B  EQU  A-2        ；将A-2的值5赋予符号名 B
COUT  EQU  CX      ；将COUT作为寄存器CX的同义名
```

PURGE语句可以解除对某一标号的赋值，使它在后面可以重新定义。

```
PURGE  COUNT        ;COUNT不再代替CX
```

2 等号语句 =

等号语句“=”与EQU语句具有相同功能，区别仅在于EQU中左边的标号不允许重新定义，而用“=”定义的语句允许重复定义。

```
A = 7          ; 正确
A = 19         ; 正确
A EQU 7        ; 正确
A EQU 19       ; 再次定义，错误
```

三　段定义、分配语句

1　段定义语句

格式：段名　SEGMENT　定位类型 组合类型　‘分类名’

....

段名　ENDS

段名：是逻辑段的标识符，不可省略，它确定了逻辑段在存储器中的地址，SEGMENT和ENDS前的段名必须相同。

SEGMENT...ENDS：是段定义的伪指令助记符，任何一个逻辑段必须以SEGMENT开始，ENDS结束，不可省略，并且必须成对出现，两者之间是本逻辑段的内容。

定位类型——对该段起始地址定位

- PAGE（页）——起始地址可以被256整除（XXXOOH）
- PARA（节）——起始地址可以被16整除（XXXXOH）
- DWORD（双字）——起始地址可以被4整除（XXXXNH）（N为4的倍数）
- WORD（字）——起始地址可以被2整除（XXXXNH）（N为偶数）
- BYTE（字节）——起始地址可以被1整除（XXXXXH）
- 缺省为PARA

组合类型——表示段与段之间的连接

- NONE：该段与其它同名段不进行连接，各段独立存在于存储器中，NONE可作为缺省参数。
- PUBLIC：该段与其它模块中的同名段连接时，由低地址到高地址连接起来，组成一个逻辑段，连接次序由连接命令指定，连接时满足定位类型要求。
- COMMON：该段在连接时与其它模块中的同名段有相同的起始地址,采用覆盖的方式在存储器中存放，连接长度为各分段中最大长度。
- AT表达式：定位该段的起始地址在表达式所指定的节(16的整数倍)边界上。一般情况下各个逻辑段在存储器中的位置由系统自动分配，当用户要求某个逻辑段在指定节 的边界上时，就要用AT参数来实现。

- STACK：指定该段为堆栈段，此参数在堆栈段中不可省略，多个模块只需设置一个堆栈段，各个模块中的堆栈段采用覆盖方式组合．容量为各个模块中所设置的最大堆栈段容量。

- MEMORY：定位该段与其它模块中的同名段有相同的首地址，采用覆盖方式在存储器中组合连接，其功能与COMMON参数类似，区别是第一个带MEMORY参数的逻辑段复盖在其它同名段的最上层，其它带此参数的同名段按照COMMON方式处理。

分类名

必须用单引号‘ ’括起来，分类名可选择不超过40个字符的名称，主要作用是汇编程序连接时将所有分类名相同的逻辑段组成一个段组。

2 段分配语句

格式：ASSUME　CS：段名，DS：段名，SS：段名，ES：段名

功能：--定义4个逻辑段，指明段与段寄存器的关系。

段名：--必须是**SEGMENT...ENDS**定义过的。

ASSUME...NOTHING取消前面**ASSUME**指定的段寄存器。

四个段不一定全部定义,代码段和数据段必须定义。

ASSUME只是指定某段分配给何寄存器，并不能将段地址装入段寄存器.仅CS在分配时自动装入。

四　过程定义语句

格式：过程名　PROC　属性

....

RET　N

过程名　ENDP

功能：定义一个过程，主程序可以用CALL指令调用它。

过程名：给所定义的过程取的名字，不可缺省。它是主程序调用的目标操作数。过程名具有三种属性

- ✓ 段属性：为该过程所在段的段基址。
- ✓ 偏移地址属性：指该过程第一个字节与段首址之间距离字节。
- ✓ 距离属性：为NEAR或FAR。格式中的属性就指距离属性,NEAR为缺省使用。

PROC...ENDP: 过程定义伪指令助记符，成对出现，不可缺省。二者前面有相同的过程名，整个过程内容包括在**PROC...ENDP**之内。

RET N: 过程内部的返回指令。过程内部至少有一条RET指令，它可以在过程的任何位置上，使过程返回到主程序调用它的**CALL**指令之下一条指令。**RET**后面跟的N为弹出值，可以缺省，N表示从过程返回以后，堆栈中应有**N**个字节的值作废(从栈顶开始)，**N**必须为正偶数。过程内部可以有多个**RET**，表示此过程具有多个返回出口(在不同条件下，从不同出口返回)。

例　两个16位无符号二进制数相乘

```
DATA    SEGMENT                          ；数据段
        D1   DW 1234H
        D2   DW 5678H
        P1   DD ?
DATA    ENDS
STACK   SEGMENT STACK 'STACK'            ；堆栈段
        DW 100 DUP(?)
STACK   ENDS
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA,SS:STACK
MAIN    PROC FAR
START:  MOV AX,STACK                     ；初始化SS
        MOV SS,AX
```

```
        PUSH DS                 ；返回DOS用
        SUB AX,AX
        PUSH AX
        MOV AX,DATA             ；初始化DS
        MOV DS,AX
L1:     MOV AX,D1               ；D1×D2，积→DX AX
        MUL D2
        MOV BX,OFFSET P1        ；积保存到存储单元P1中
        MOV [BX],AX
        MOV [BX+2],DX
        RET
MAIN    ENDP
CODE    ENDS
        END START
```

例　用过程调用的方法，将内存中N个BCD码相加

```
DATA    SEGMENT                          ；数据段
        ONE   DB 22H,33H,44H,55H
        TWO   DB 55H,66H,77H,88H
        SUM   DB 20 DUP(?)
DATA    ENDS
STACK   SEGMENT STACK                    ；堆栈段
        STT    DB 100 DUP(?)
        TOP    EQU LENGTH STT
STACK   ENDS
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA,SS:STACK,ES:DATA
MAIN    PROC FAR
START:  MOV AX,STACK                     ；初始化SS,SP
        MOV SS,AX
        MOV SP,TOP
        ……
```

```
        MOV SI,OFFSET ONE       ;SI指向第一个加数
        MOV BX,OFFSET TWO       ;BX指向第二个加数
        MOV DI,OFFSET SUM       ;DI指向和
        CLD                     ;清方向标志
        MOV CX,4
LL:     CALL ABC
        LOOP LL
        RET
MAIN    ENDP
ABC     PROC NEAR               ；完成单字节数据加法运算
        LODSB                   ;[SI]→AL,SI+1→SI
L1:     ADD  AL,[BX]
        DAA
        STOSB                   ;AL→[DI],DI+1→DI
        INC BX
        RET
ABC     ENDP
CODE    ENDS
        END START
```

五　程序开始和结束语句

1 NAME　　程序名　　;为源程序目标模块赋名

功能—置于程序开始,则在输出源程序列表文件时,将在每页开头打出程序名.省略则打印源文件名。

2 TITLE　文本名　　;文本名赋予目标模块

功能—同**NAME**

3　ORG　表达式

功能—给程序设置地址指针,指定下个语句起始偏移地址。

表达式—给定偏移地址,结果为正整数。

- ✓ 一般情况下，段定义语句(SEGMENT)指出了段的起点，偏移地址为0，段内各个语句或数据的地址由段地址开始依次类推可确定。当用户要求指定某条指令或数据为某个指定地址时，可用ORG语句来改变，ORG语句可以放在程序的任何位置。

例　用ORG改变数据段地址

```
DATA   SEGMENT
       ORG 100H
       A1 DB 10H,20H,30H
       ORG 200H
       A2 DW 3031H,3233H
DATA ENDS
```

4　END　标号名　　;标记汇编程序结束

⊿ **END**在源程序最后一行

⊿ 每个源程序只有一个**END**

6-4 DOS和BIOS调用

- DOS (Disk Operating System) 是IBM PC机的磁盘操作系统。
- DOS是用户和PC机之间的接口。
  - ◆内部命令：

    如DIR、TYPE、CD等
  - ◆外部命令：

    如PRINT、XCOPY、FORMAT等

- DOS还具有对I/O设备管理及磁盘与文件管理的功能。
  - 一部分被固化在系统的ROM中，可作为ROM BIOS(Basic Input/Output System)模块。用户使用BIOS中断调用来使用它们。
  - 另一部分存放在系统磁盘上，在系统启动时被装入内存，用户的应用程序及MS-DOS的大部分命令都将通过软件中断来调用它们。
- 调用这些软中断时，只要给定入口参数，接着写一条中断指令INT n就可以了。

DOS常用的软中断命令

软中断指令	功能	入口参数	出口参数
INT 20H	程序正常退出	无	无
INT 21H	系统功能调用	AH=功能号，相应入口号	相应出口号
INT 22H	结束退出		
INT 23H	Ctrl-Break处理		
INT 24H	出错退出		
INT 25H	读磁盘	AL=驱动器号 CX=读入扇区数 DX=起始逻辑扇区号 DS:BX=内存缓冲区地址	CF=0成功 CF=1出错
INT 26H	写磁盘	AL=驱动器号 CX=写入扇区数 DX=起始逻辑扇区号 DS:BX=内存缓冲区地址	CF=0成功 CF=1出错
INT 27H	驻留退出	DS:DX=程序长度	

一、常用的软件中断

- 读写磁盘扇区的软件中断

  INT 25H

  INT 26H

- 退出程序的软件中断

  INT 20H

  INT 23H

  INT 24H

  INT 27H

二、DOS系统功能调用

DOS系统功能调用分别实现设备管理、文件读写、文件管理和目录管理等功能。每个子程序对应一个功能号，所有的系统功能调用的格式是一致的，按下面4步进行：

(1)系统功能号送到AH寄存器中

(2)入口参数送到指定寄存器中

(3)由INT 21H指令执行功能调用

(4)根据出口参数分析功能调用执行情况

1. DOS键盘功能调用

- 键盘提供了字符键(数字0~9，字母A~Z，a~z，%，$，#)，功能键(Home，End，Del，Ins，PgUp，PgDown...等)和控制键(Ctrl，Alt，Shift)。每个键都有对应的键值，即标准ASCII码值。
- 通过DOS功能调用可读入键值到AL寄存器或存储器中。

DOS键盘功能调用

AH	功　　能	入 口 参 数	出口参数
1	从键盘输入一个字符,并在屏幕上显示,检查Ctrl_Bteak键		AL=字符
8	键盘输入一个字符,无回显		AL=字符
6	直接键盘输入/输出字符,不检查Ctrl_Break键	DL=0FFH	AL=字符
7	直接键盘输入/出字符,无回显,不检查Ctrl_Break键		AL=字符
0AH	输入字符串到内存缓冲区	DS:DX=缓冲区首址	
0BH	检查键盘输入状态		AL=FFH有键入 AL=0无键入
0CH	清键盘缓冲区，调用键盘输入功能。	AL=键盘功能(1,6,7,8,A)	

从键盘输入字符或字符串

- 1，8，6，7号功能调用从键盘输入一个字符到AL寄存器
  - 1号和6号功能调用－输入同时在屏幕上显示字符
  - 8号和7号功能调用－不在屏幕上回显字符
- 0AH号功能调用从键盘接收字符串到内存缓冲区
- 要求预先定义一个输入缓冲区，缓冲区的第一个字节指出能容纳的最大字符个数，由用户给出；第二个字节存放实际输入的字符个数，由系统最后填入；从第三个字节开始存放从键盘接收的字符，直到ENTER键结束。

例：1号功能调用

交互式程序中用户按下数字键1，2，3，程序转入相应的服务子程序，若按下其它键就继续等待。

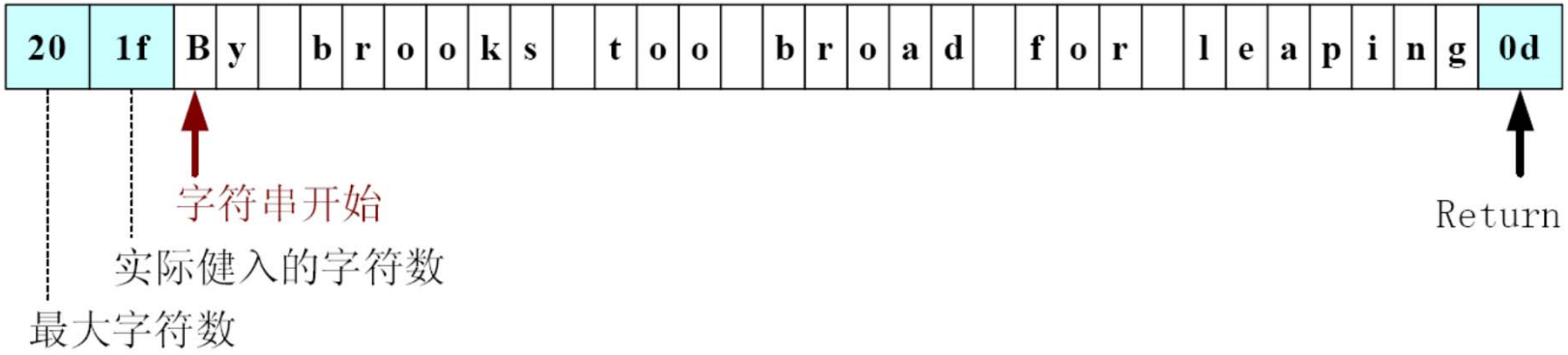
```
KEY: MOV AH, 1          ;读入键值→AL
     INT 21H
     CMP AL, '1'        ;键值为 '1'否?
     JE ONE
     CMP AL, '2'        ;键值为'2'否?
     JE TWO
     CMP AL, '3'        ;键值为'3'否?
     JE THREE
     JMP KEY            ;否则返回，继续等待键盘输入
ONE: .....              ;程序分支1
TWO: .....              ;程序分支2
THREE: ...              ;程序分支3
```

例：0AH号功能调用

```
BUFF    DB 32
        DB ?
        DB 32 DUP(?)
        MOV AX,DATA
        MOV DS,AX
        MOV DX,OFFSET BUFF
        MOV AH,0AH
        INT 21H
```

输入字符串内(31bytes)：

By brooks too broad for leaping



2. DOS显示功能调用

DOS显示功能调用能够显示单字符或字符串，这些功能都自动向前移动光标。

AH	功　　能	入口参数	说　　明
2	显示一个字符,检验trl_Break键	DL=字符	光标跟随字符移动
6	显示一个字符,不检验Ctrl_Break键	DL=字符	光标跟随字符移动
9	显示字符串	DS:DX=串地址	串以'$'结束,光标随串移动

例：显示单个字符‘3’

```
CODE    SEGMENT
        ASSUME CS:CODE
START:  MOV DL,33H              ；将33H给DL
        MOV AH,2                ；调用显示功能
        INT 21H
        MOV AH,4CH              ；退回DOS
        INT 21H
CODE    ENDS                    ；程序段结束
        END     START           ；程序结束
```

例：输出一串字符

```
DATA    SEGMENT
        HM    DB 'HELLO,WORLD', 0DH, 0AH, '$'
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA
START   PROC FAR
        MOV AX, DATA        ;DS指向数据段
        MOV DS, AX
```

```
        MOV DX, OFFSET HM ;DX指向串首
        MOV AH, 9         ;dos功能调用9号子功能设置
        INT 21H           ;显示字符串
        MOV AH, 4CH       ; 返回DOS
        INT 21H
        RET               ;过程结束
START ENDP
CODE  ENDS ;程序段结束
        END START ;程序结束
```

3 DOS打印功能调用

INT 21H的5号功能调用完成将DL寄存器中的字符送到打印机，若需要回车换行，也同样将回车换行的字符码送到DL寄存器。

打印机标准控制字符	
字符码	功　　能
08H	空格
09H	水平TAB(横表)
0AH	换行
0BH	垂直TAB(纵表)
0CH	换页
0DH	回车

例：完成一串字符打印，遇到‘$’结束。打印开始换页，打印结束换行(2行)。

```
        TEXT DB 0CH, 'This is DOS function call!',0DH,0AH,0AH,'$'
        MOV BX, 0
        MOV AH, 5
NEXT:   MOV DL, TEXT[BX]
        CMP DL, '$'
        JE STOP
        INT 21H
        INC BX
        JMP NEXT
STOP:
```

特殊的打印命令

字符码	功　　能
0FH	设置紧缩方式
0EH	设置扩展方式
12H	取消紧缩方式
14H	取消扩展方式
1BH 30H	设置每英寸8行
1BH 32H	设置每英寸6行
1BH 45H	设置加重打印方式
1BH 46H	取消加重打印方式

例：打印一句标题，设置成紧缩方式，每英寸8行，打印并回车换行

```
TEXT   DB  0FH,1BH,30H,'TITLE...',0DH,0AH
       MOV  BX,0
       MOV  CX,20
       MOV  AH,5
NEXT:  MOV  DL,TEXT[BX]
       INT  21H
       INC  BX
       LOOP NEXT
```

三、BIOS功能调用

在存储系统中，从地址0FE000H开始的8K ROM（只读存储器）中装有BIOS (Basic Input/Output System)例行程序。驻留在ROM中的BIOS提供了系统加电自检，引导装入主要I/O设备的处理程序以及接口控制等功能模块来处理所有的系统中断。程序员可直接用指令设置参数，中断调用BIOS中的子程序。

BIOS功能调用的类型

10H 显示器I/O	16H 键盘
11H 设备检验	17H 打印机
12H 内存大小	18H 驻留BASIC
13H 磁盘	19H 引导
14H 通讯	1AH 时钟
15H I/O系统扩充	40H 软盘

DOS与BIOS的关系

- BIOS和DOS是两组系统服务程序的集合，它们使程序员能访问和使用组成IBM-PC机的硬件。
- 在一些情况下，选择DOS中断或选择BIOS中断可以执行同样的功能。
- BIOS程序提供了基本的低层服务，所以通过BIOS功能调用比通过相应的DOS功能调用更能提高程序的执行效率。
- 只有少数的BIOS功能调用没有相应的DOS中断。

如何进行BIOS调用

- 中断调用前需要把功能号装入AH中，设置相应的入口参数。中断调用后，或产生一个动作，或设置相应的出口参数。具体请查阅书上BIOS功能调用的相关表格。
- 举例来说，INT 10H是常用的BIOS功能调用，它是显示器I/O的BIOS功能调用。
- 置光标位置指令如下：

```
MOV  AH,02H        ;装入功能号
MOV  DH,6          ;定光标所在行
MOV  DL,6          ;定光标所在列
MOV  BH,  0        ;定光标所在页
INT  10H
```

BIOS键盘中断调用（INT 16H）

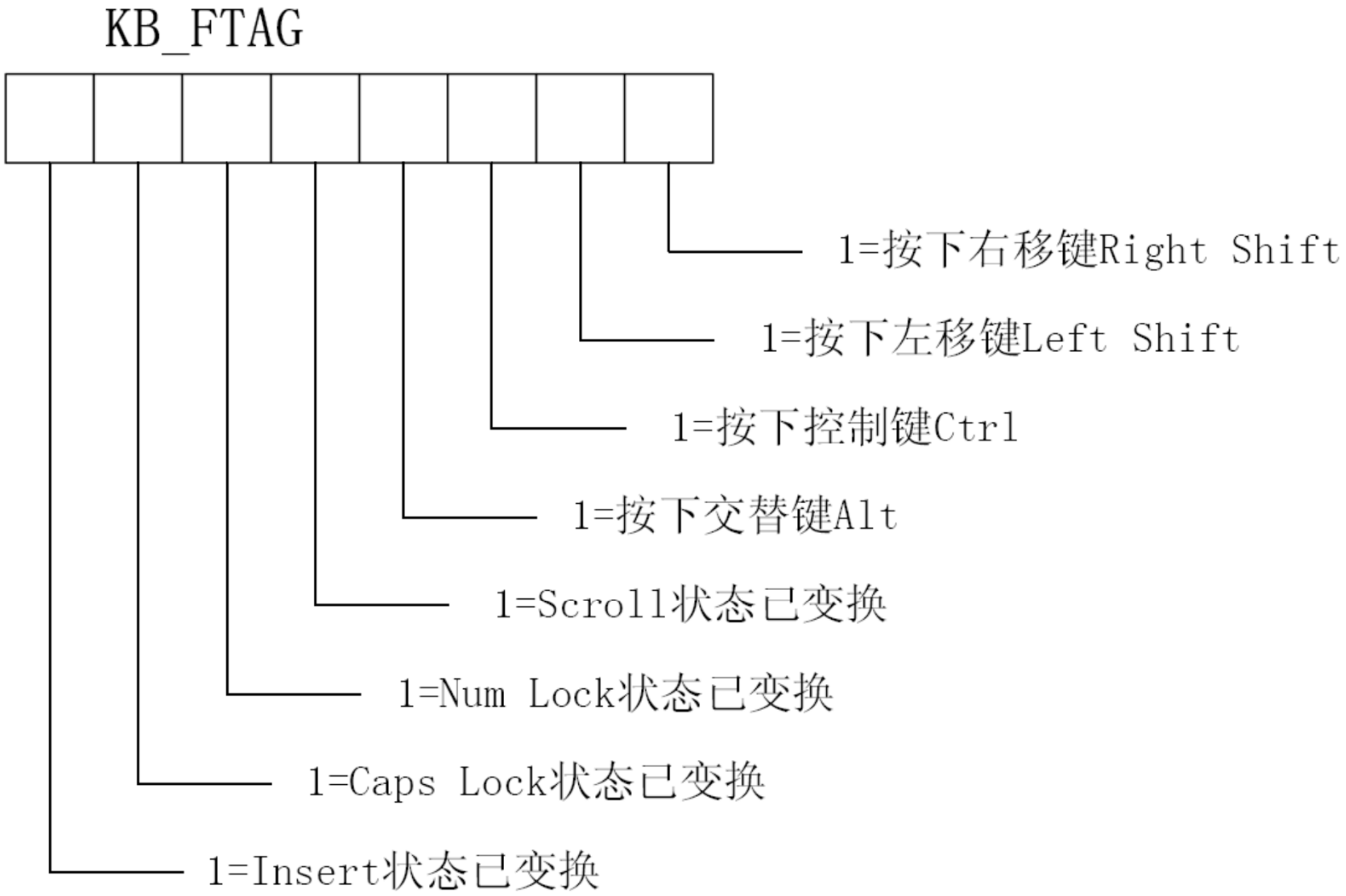
先送功能号到AH中，然后使用INT 16H指令，将键盘输入的ASCII码送到AL寄存器。

AH	功　能	返　回　参　数
0	从键盘读一个字	AL=字符码 AH=扫描码
1	读键盘缓冲区的字符	如ZF=0 AL=字符码 AH=扫描码
		如ZF=1，缓冲区空
2	取键盘状态字节	AL=键盘状态字节

特殊功能键按下的判断

- 特殊功能键指不具备ASCII码的键盘按键
- 特殊功能键包括：
  - Shift，Ctrl，Alt，Num，Lock，Scroll，Ins，Caps Lock
- 在INT 16H功能调用中，如果功能号AH＝2时，AL中返回键盘状态字节。
- 键盘状态字节表示特殊功能键的状态变换情况

键盘状态字节


KB_FTAG
1=按下右移键Right Shift
1=按下左移键Left Shift
1=按下控制键Ctrl
1=按下交替键Alt
1=Scroll状态已变换
1=Num Lock状态已变换
1=Caps Lock状态已变换
1=Insert状态已变换

6-5 程序设计方法

程序设计基本要求

—能正常运行,实现要求的功能,且具有以下特点：

① 程序结构模块化, 简明, 易读, 易调试, 易维护

② 执行速度快

③ 占用存储器空间少

汇编语言程序设计步骤

① 分析问题，抽象出描述问题的数学模型，确定合理算法

② 绘制程序流程图

③ 分配存储空间及工作单元（各段位置及段寄存器）

④ 根据流程图编写程序

⑤ 静态检查，上机调试

⑥ 程序运行，结果分析

程序结构

1 顺序结构

—程序顺序执行，无分支，无循环，无转移。

2 分支结构

—程序在顺序执行中，根据不同的计算结果由计算机自动判断，然后按不同条件选择下一步执行的程序段。

3 循环结构

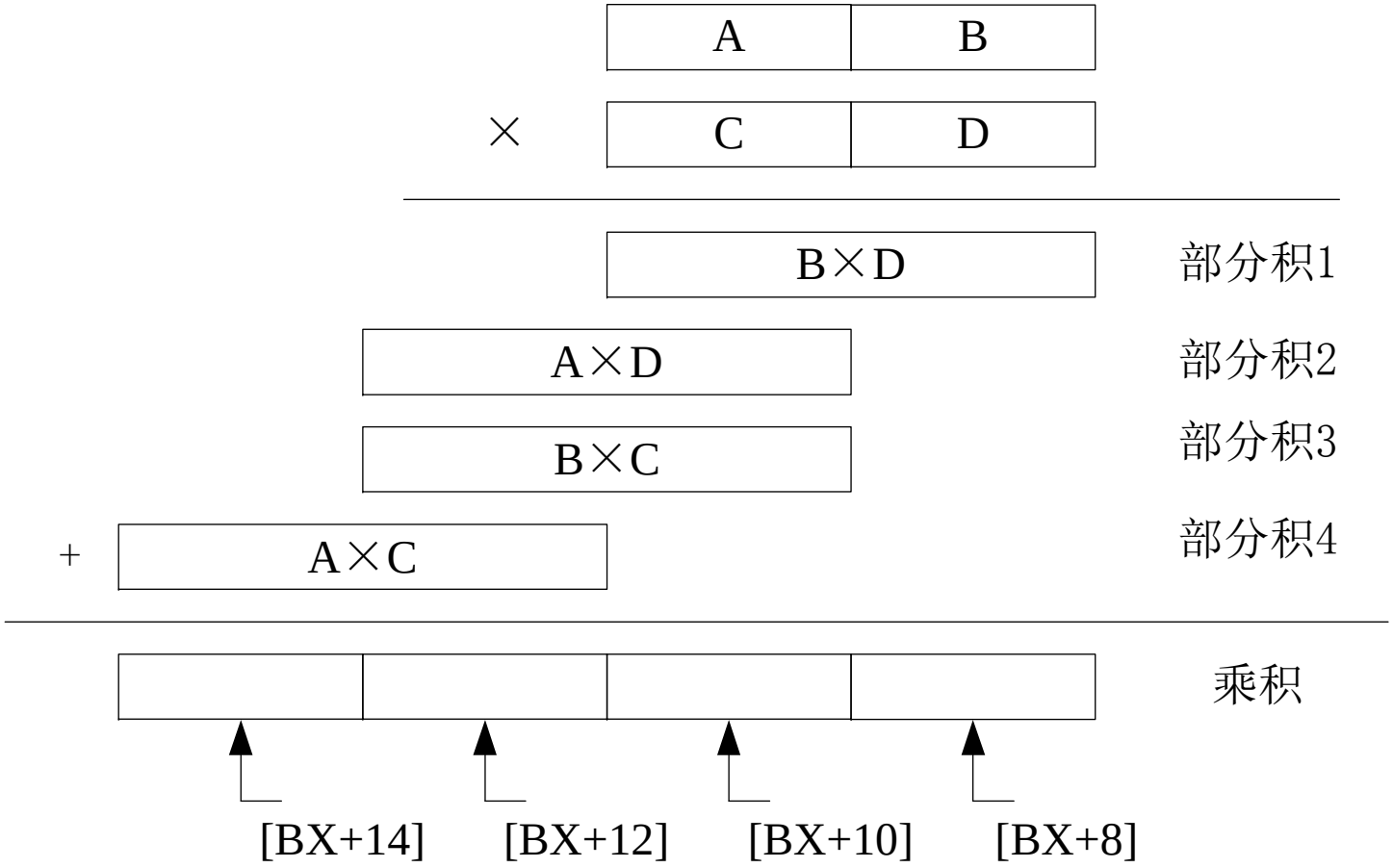
—在给定条件成立时，反复执行某程序段，直到条件不成立为止。 给定的条件称为循环条件，反复执行的程序段称为循环体。

4 子程序结构

—汇编语言中多次使用的程序段可写成一个相对独立的程序段，将它定义为“过程”或称子程序，需要执行这段程序时，就进行“过程”调用。

一、顺序结构

例：实现两个32位无符号数乘法。

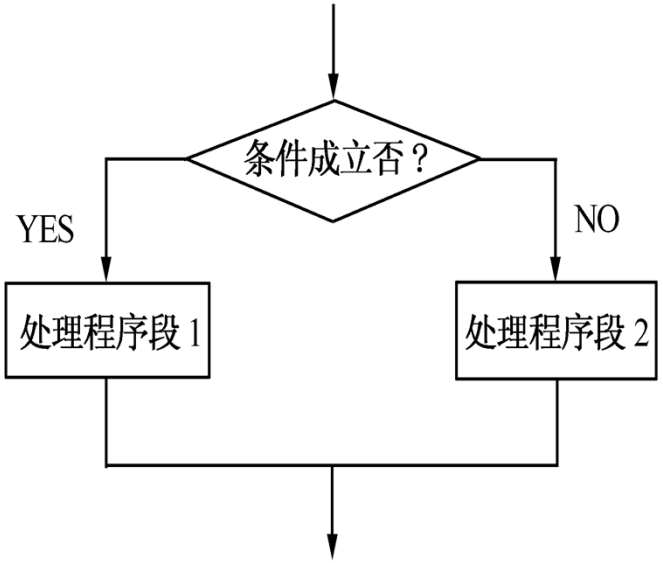


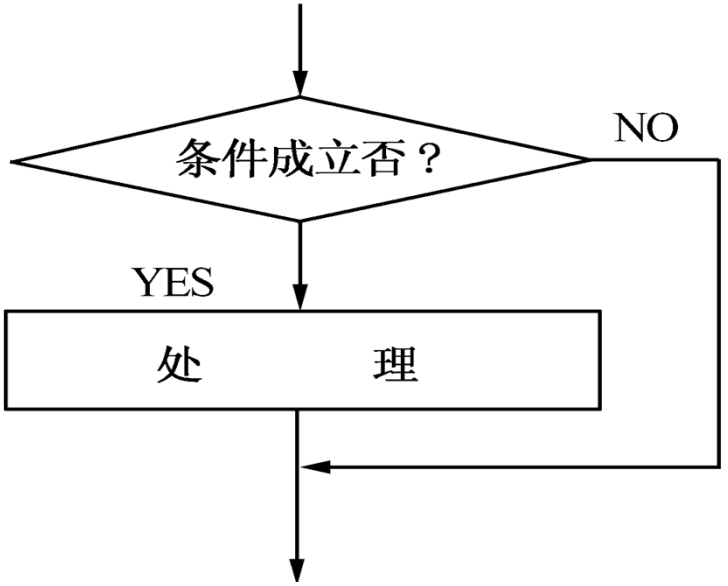
```
DATA    SEGMENT
        NUM1   DW     1200H，3400H
        NUM2   DW     5600H，7800H
        MUT    DW     4 DUP（？）
DATA    ENDS
STACK   SEGMENT       PARA  STACK ’STACK’
        DB     100 DUP（？）
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS:CODE，DS:DATA，SS:STACK
BEGIN:  PUSH          DS
        MOV           AX，0
        PUSH          AX
        MOV           AX，DATA
        MOV           DS，AX
```

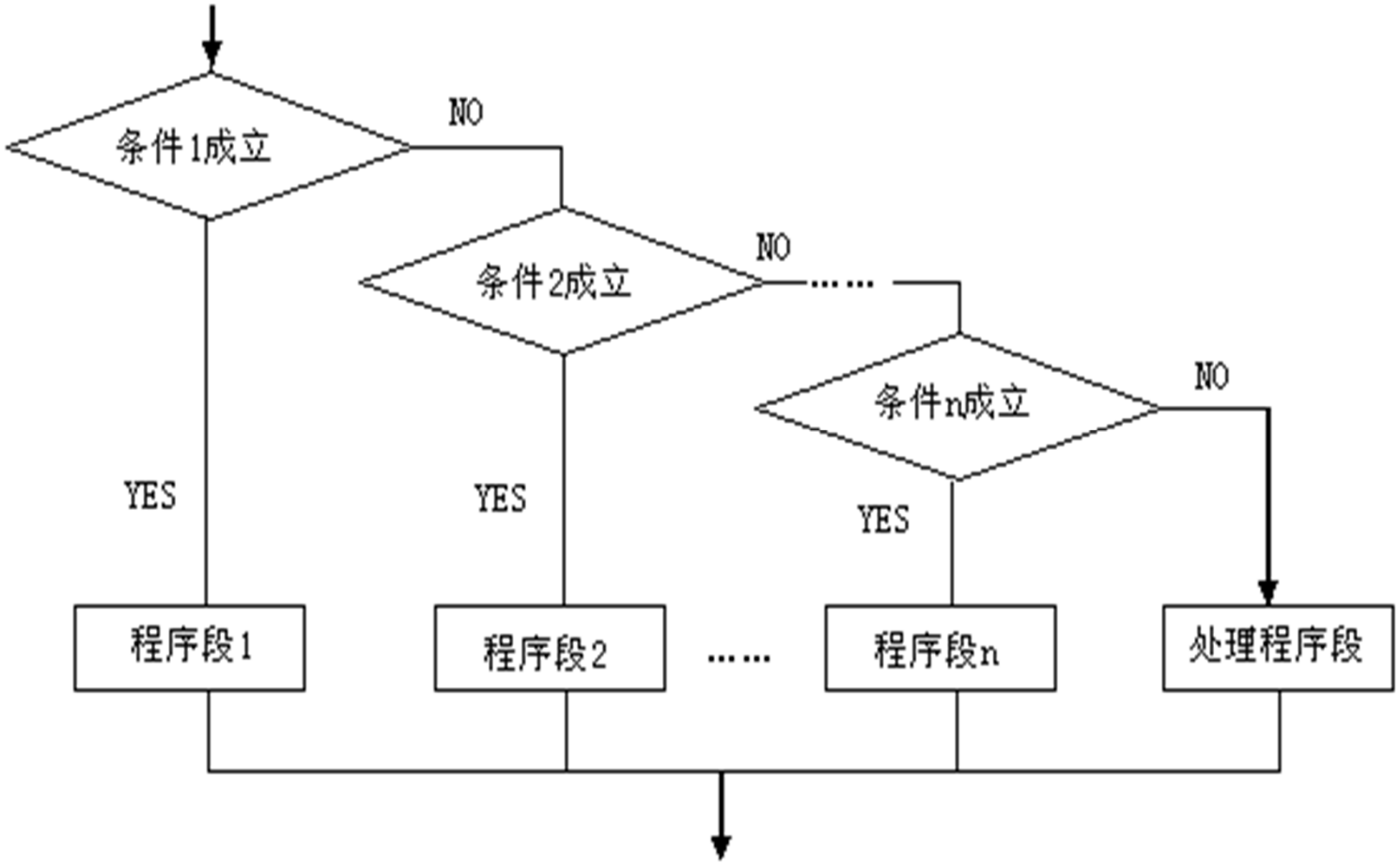
```
LEA   BX, NUM1
MOV   AX, [BX]     ; B→AX
MOV   SI, [BX+4]   ; D→SI
MOV   DI, [BX+6]   ; C→DI
MUL   SI           ; B*D
MOV   [BX+8], AX   ; 保存部分积1
MOV   [BX+0AH], DX
MOV   AX, [BX+2]   ; A→AX
MUL   SI           ; A*D
ADD   [BX+0AH], AX
ADC   [BX+0CH], DX; 带进位加入积2单元中
```

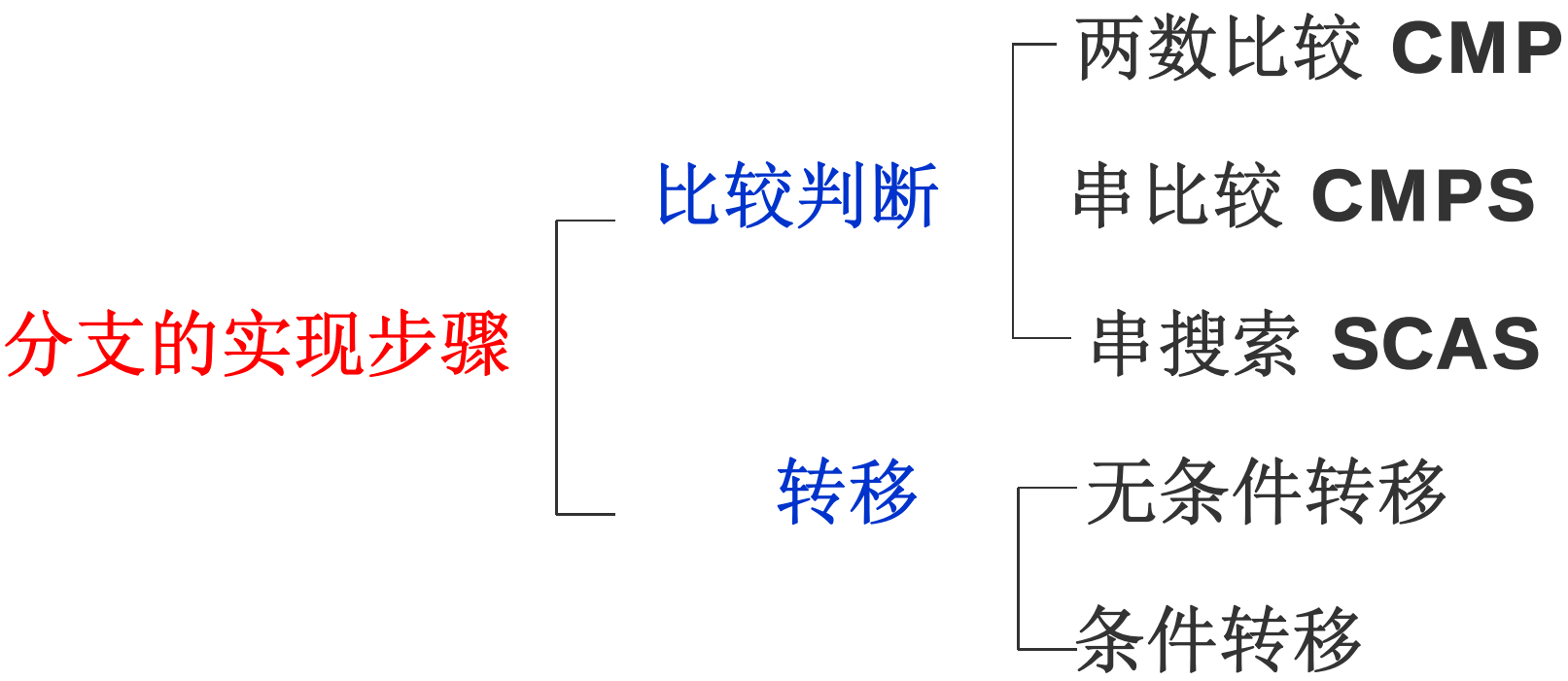
```
        MOV     AX, [BX]        ; B→AX
        MUL     DI              ; B*C
        ADD     [BX+0AH],AX     ; B*C加入积单元
        ADC     [BX+0CH],DX             ; 带进位加入积3单元中
        ADC     WORD  PTR[BX+0EH], 0    ; 带进位加入积4单元中
        MOV     AX, [BX+2]      ; A→AX
        MUL     DI              ; A*C
        ADD     [BX+0CH], AX
        ADC     [BX+0EH], DX
        MOV     AH, 4CH         ; 返回DOS
        INT     21H
CODE    ENDS
        END             BEGIN
```

二、分支结构





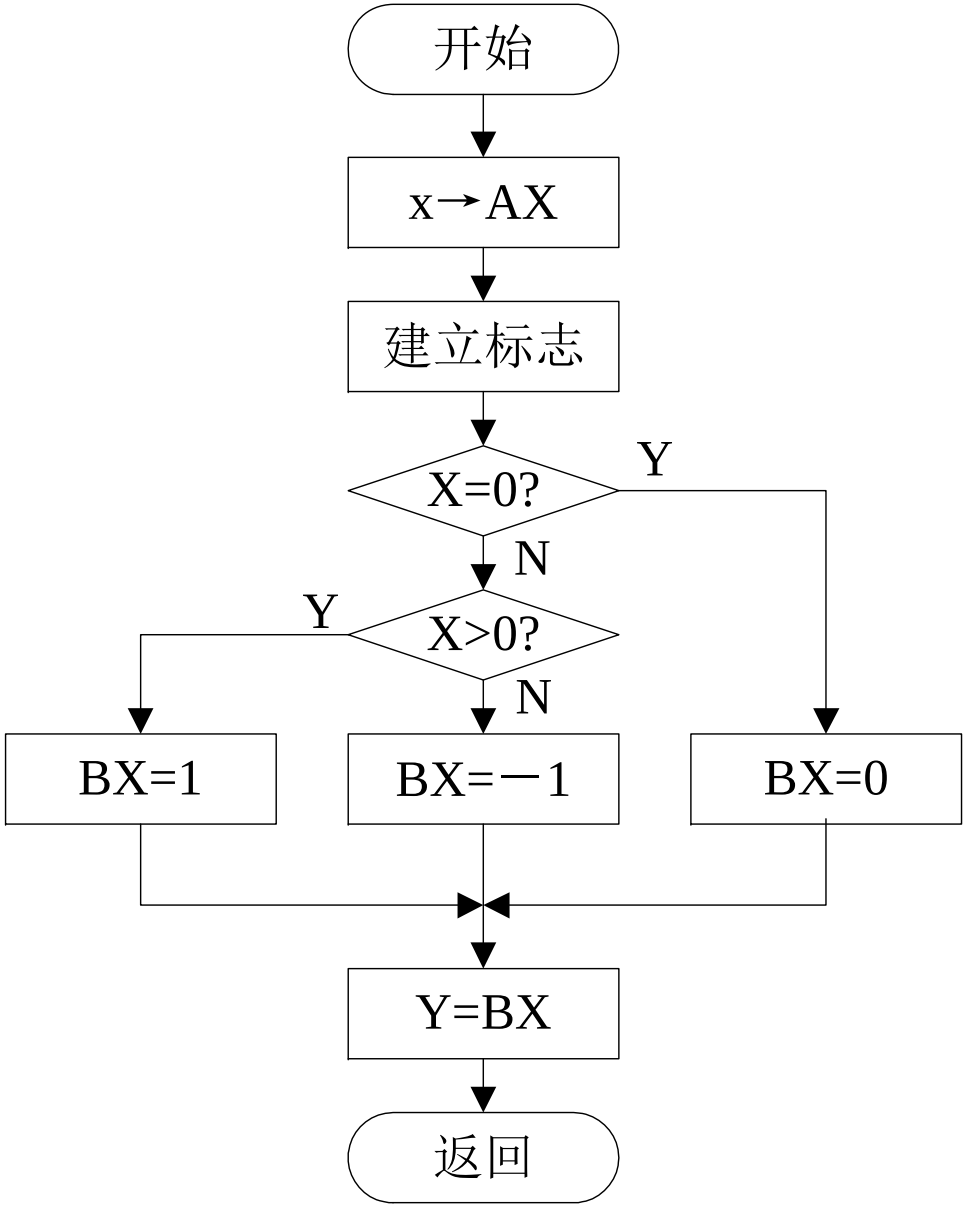

条件1成立
NO
条件2成立
NO
……
条件n成立
NO
YES
YES
YES
程序段1
程序段2
……
程序段n
处理程序段


分支的实现步骤
比较判断
两数比较 CMP
串比较 CMPS
串搜索 SCAS
转移
无条件转移
条件转移

例 实现符号函数

$$y=\begin{cases}1, x>0\\0, x=0\\-1, x<0\end{cases}$$

- 要实现符号函数，只要把*x*从内存中取出来，执行一次“与”或“或”操作，就可以把*x*的数值特征反映在标志上，再根据标志来转移。



```
DATA        SEGMENT
            X       DW      -5
            Y       DW      ?
DATA        ENDS
STACK       SEGMENT         PARA   STACK 'STACK'
            DB      100 DUP (? )
STACK       ENDS
CODE        SEGMENT
            ASSUME          CS:CODE, DS:DATA, SS:STACK
BEGIN:      PUSH   DS
            MOV    AX, 0
            PUSH   AX
            MOV    AX, DATA
            MOV    DS, AX
```

```
        MOV     AX，X
        AND     AX，AX          ；建立标志
        JZ      ZERO            ；X=0转ZERO执行
        JNS     PLUS            ；X>0转PLUS执行
        MOV     BX，0FFFFH      ；X<0，使BX =-1
        JMP     DONE
ZERO:   MOV     BX，0
        JMP     DONE
PLUS:   MOV     BX，1
DONE:   MOV     Y，BX           ；存放结果
        MOV     AH，4CH         ；返回DOS
        INT     21H
CODE    ENDS
        END             BEGIN
```

三、循环程序结构

循环结构包含4部分:

⑴初始化:设置循环记数值(次数)和变量初值;

⑵循环体：循环核心，包括循环的全部执行指令;

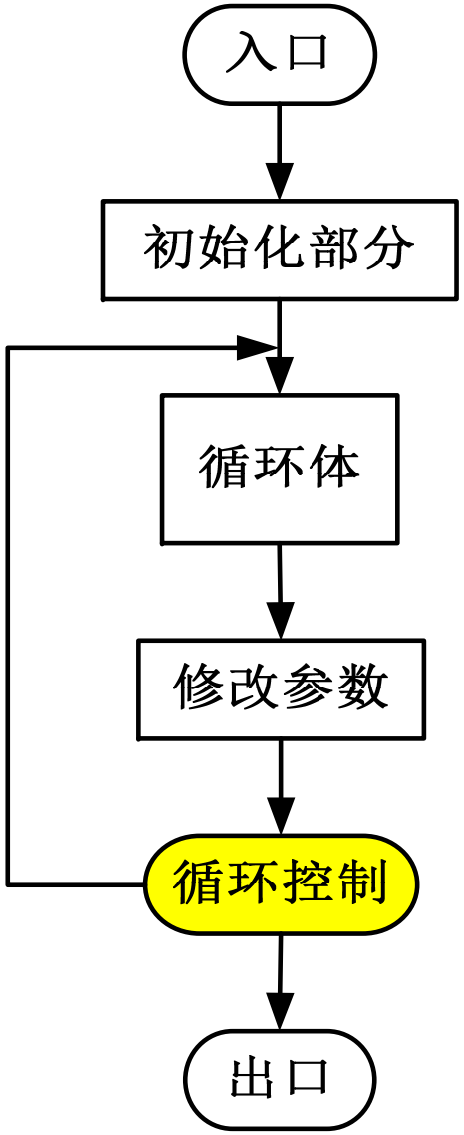
⑶修改参数：修改源和目的操作数地址;

⑷循环控制：修改记数器值，判断控制条件，决定是否跳转循环。

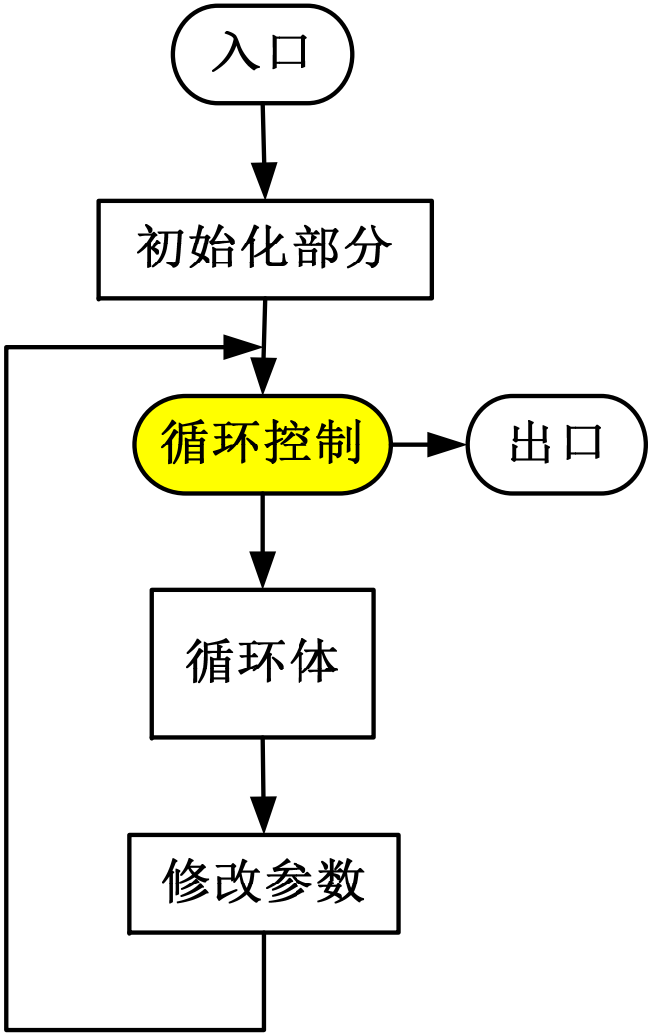
两种结构形式:

⑴先执行后判断：先执行循环体（至少一次），再判断是否结束，一般适用于次数固定的循环。

⑵先判断后执行：先判断结束条件，再决定是否执行循环体。适用于次数不固定的循环。

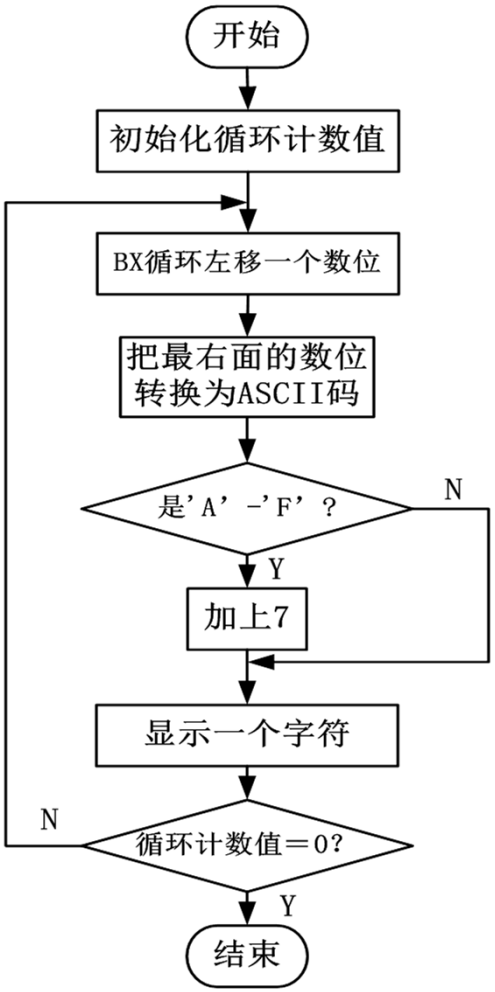


“先执行，后判断”结构



“先判断，后执行”结构

例 将BX中的16进制数转换为ASCII码，存放到BUF开始的内存单元中去，并在屏幕显示出数值。



先执行,后判断

注意:

① BX中有4位十六进制数

② HEX数‘0’~’9’和 ‘A’~'F’的ASCII码不连续

③ 显示1位字符用DOS功能2

```
        MOV SI,OFSET BUF      ；设置内存地址
        MOV CH,4              ；计数初值＝4
NEXT:   MOV CL,4
        ROL BX,CL             ；最高位移到右边
        MOV AL,BL             ；一位数转换成ASCII码
        AND AL,0FH
        ADD AL,30H
        CMP AL,3AH              ；字符为A～F吗？
        JL STORE                ；字符为0～9
        ADD AL,7                ；字符为A～F
STORE:  MOV [SI],AL             ；字符存入内存
        MOV AH,2                ；调用屏幕显示
        MOV DL,AL
        INT 21H
        INC SI                  ；修改计数并判断
        DEC CH
        JNZ NEXT
        HLT                     ；循环结束
```

例 AX寄存器中有一个16位二进制数，编程统计其中1的个数，结果放到CL寄存器。 （先判断,后执行 ）

```
        MOV CL,0          ；初始化
L1:     AND AX,AX         ；控制循环
        JZ STOP
        SAL AX,1          ；循环体
        JNC L2
        INC CL
L2:     JMP L1
STOP: HLT
```

四、子程序结构

(1)子程序必须用过程定义语句定义

一般采用FAR属性,同一代码段可用NEAR

调用时属性应与过程属性对应。

(2)使用CALL须注意

①保护断点

②保护寄存器内容

③主子程序之间参数传递

- a.寄存器传递:适用参数少,传递速度快
- b.存储器传递:适用参数多,须建参数表
- c.堆栈传递:参数较多时可用,适于子程序嵌套和子程序递归调用

(3)子程序说明

①功能描述：名称、功能及性能；

②子程序用到的寄存器及存储器单元；

③入口参数和出口参数；

④子程序调用其它子程序的名称。

例　数据段定义两个数组，编程实现数据段分别求和(不计溢出)　　存储器传递参数

```
DATA    SEGMENT
        ARY1     DW 100 DUP(?)    ;定义数组1
        SUM1     DW ?
        ARY2     DW 100 DUP(?)    ;定义数组2
        SUM2     DW ?
DATA    ENDS
STACK   SEGMENT STACK
        SA       DW 50 DUP(?)
        TOP      EQU LENGTH SA
STACK   ENDS
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA,SS:STACK
```

```
MAIN    PROC FAR
START:  MOV AX,DATA
        MOV DS,AX
        MOV AX,STACK
        MOV SS,AX
        MOV SP,TOP
        LEA SI,ARY1              ;数组1首地址，入口参数
        MOV CX,LENGTH ARY1       ;数组1长度，入口参数
        CALL SUM                 ;调用求和子程序
        LEA SI,ARY2              ;数组2首地址，入口参数
        MOV CX,LENGTH ARY2       ;数组2长度，入口参数
        CALL SUM                 ;调用求和子程序
        MOV AH,4CH               ;返回DOS
        INT 21H
        RET
MAIN    ENDP
```

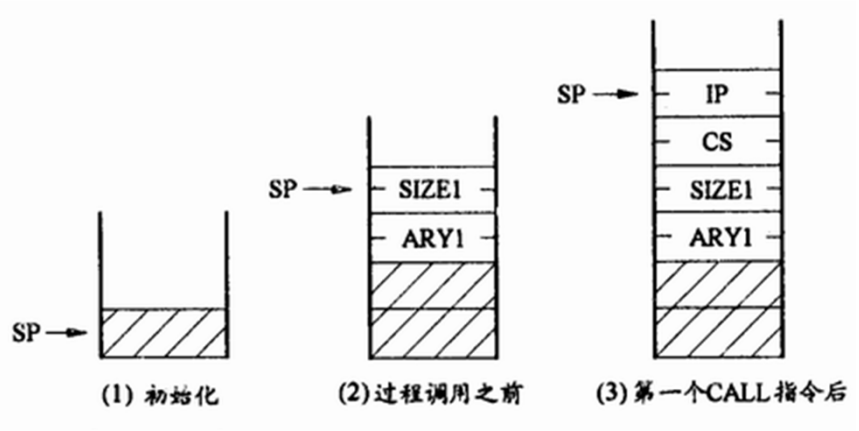
```
SUM     PROC NEAR                ;子程序
        XOR AX,AX                ;AX清0
L1:     ADD AX,WORD PTR[SI]      ;加数组元素
        INC SI
        INC SI
        LOOP L1
        MOV WORD PTR[SI],AX      ;数组和送SUM
        RET
SUM     ENDP
CODE    ENDS
        END START
```

例　通过堆栈传递参数，实现十进制数数组求和，要求主程序和过程不在同一个代码段中，要进行段间调用

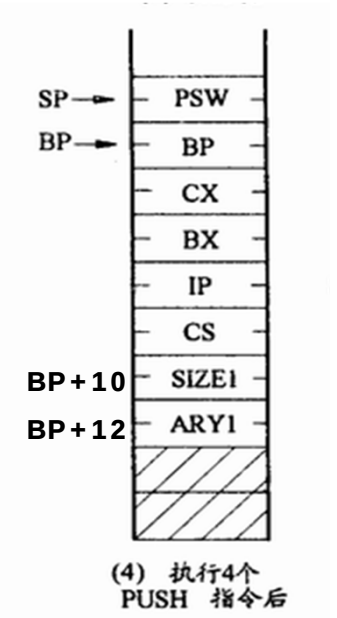
说明：

1. 使用堆栈传递参数，调用前要给过程传递两个参数，主程序中将数组的偏移地址值及数组长度压入堆栈，然后调用过程。
2. 段间过程调用，进入过程时要重新分配CS段，使之指向当前有效代码段PCODE；在过程运行中可直接调用堆栈中参数，完成累加运算，并将结果送回指定的存储单元。
3. 过程返回时用返回指令RET 4，要将堆栈中由 CALL指令之前传递来的 4个字节作废。

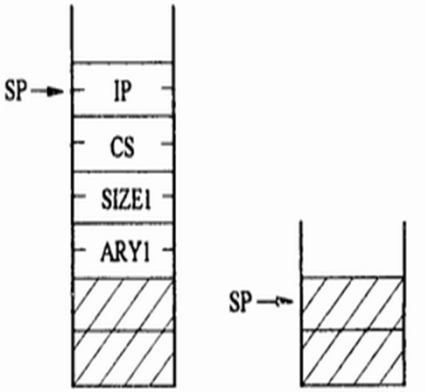
```
MDATA   SEGMENT
        ARY1  DB 20 DUP(?)      ;定义数组1
        SUM1  DW ?
        ARY2  DB 100 DUP(?)     ;定义数组2
        SUM2  DW ?
MDATA   ENDS
          ……   ……
MCODE   SEGMENT
          ……   ……
        MOV AX,OFFSET ARY1
        PUSH AX
        MOV AX,SIZE ARY1
        PUSH AX
        CALL FAR PTR PADD
          ……   ……
MCODE ENDS
```



```
PCODE    SEGMENT          ;子过程
          ……
PADD     PROC FAR
         PUSH BX
         PUSH CX
         PUSH BP
         MOV BP,SP
         PUSHF
         MOV CX,[BP+10]   ;取数组长度
         MOV BX,[BP+12]   ;取数组起始地址
           ……   ……       ;数组相加
         MOV [BX],AX      ;数组和送SUM
         POPF
         POP BP
         POP CX
         POP BX
         RET 4
           ……   ……
```



(4) 执行4个 PUSH 指令后



(5) 执行4个 POP 指令后　(6) RET 4 指令执行后

子程序嵌套和递归调用

- 子程序本身又可调用其它子程序，称为子程序嵌套，嵌套的层数不限，只要堆栈空间足够就可以。
- 注意寄存器的保护及恢复，避免各层子程序之间寄存器使用冲突，造成程序出错。
- 子程序调用子程序本身，称为子程序递归调用。