
第3章 汇编语言程序设计



目录

▶ 第3章 汇编语言程序设计

▶ 3.1 机器语言、汇编语言与高级语言

▶ 3.2 汇编语言源程序的结构

▶ 3.3 数据定义

▶ 3.4 汇编语言的运算符

▶ 3.5 基本结构程序设计


▶ 3.6 操作系统资源的使用

① 操作数寻址方式

② 指令的寻址方式

CS+IP (IP会自动加1)

3.1 机器语言、汇编语言与高级语言

- ▶ **机器语言**：计算机能够直接理解和执行的二进制代码。

- ▶ **汇编语言**：采用助记符表示机器语言，便于理解和记忆。



源程序文件

```
assume cs:codesg
```

```
codesg segment
```

```
    mov ax,0123H
```

```
    mov bx,0456H
```

```
    add ax,bx
```

```
    add ax,ax
```

```
    mov ax,4c00h
```

```
    int 21h
```

```
codesg ends
```

```
end
```

编译连接

可执行文件

描述信息

B8 23 01

BB 56 04

03 C3

03 C0

B8 00 4C

CD 21

► **汇编程序的主要功能：**

- (1)、检查源程序，测出源程序的语法错误，并给出出错信息**
- (2)、产生源程序的目标文件（二进制），并给出列表文件**
- (3)、展开宏指令**

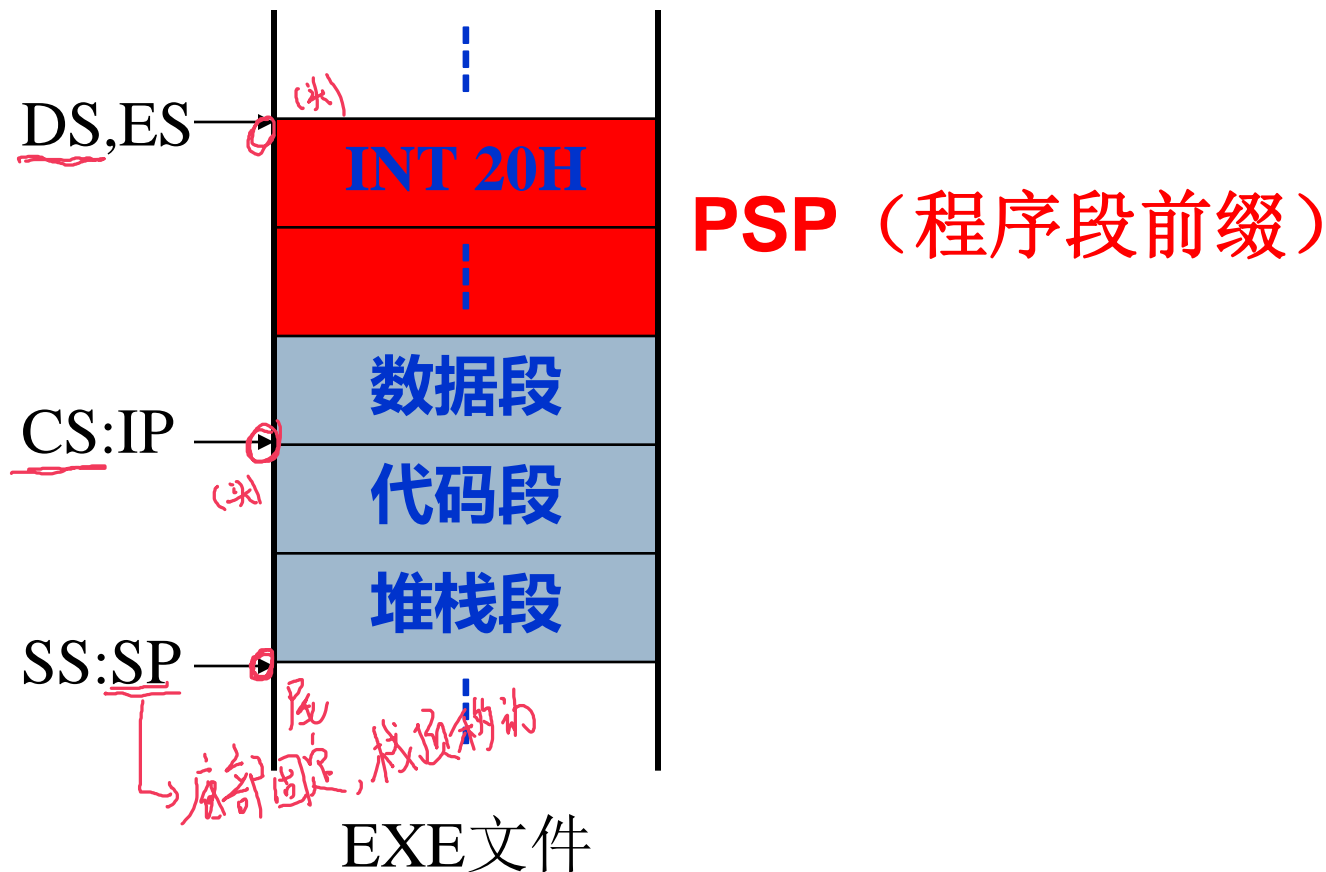
目前常用的汇编程序

Microsoft:	MASM
Borland:	TASM

编程调试过程

- 第一步：编辑 **EDIT** 文件名 (**.ASM**)
- 第二步：汇编 **MASM** 文件名 (**.ASM**)
--对应成二进制
- 第三步：连接 **LINK** 文件名 (**.OBJ**)
- 第四步：运行 文件名 (**.EXE**)
- 第五步：调试 **DEBUG** 文件名. EXE

DOS装入EXE文件后内存的分配状况



PSP的头两个字节是INT 20H, 用户程序可通过该指令返回操作系统DOS

如何使用户程序执行完后返回来执行这条指令？

方法一：

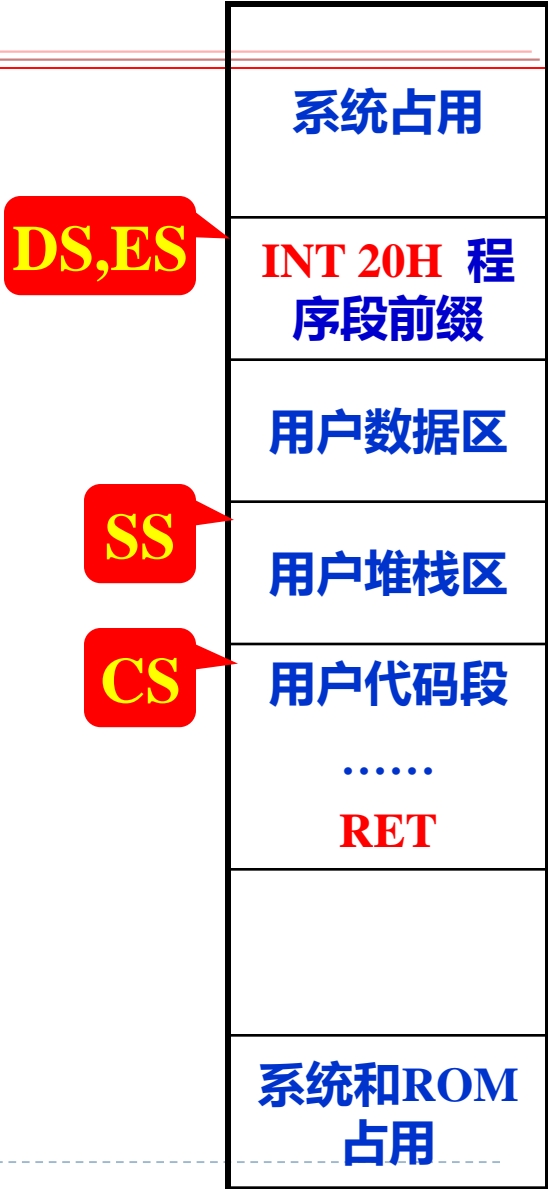
将用户程序定义为一个远过程，当可执行文件装入内存后，DS,ES两个段寄存器被CPU自动设置为指向PSP的首址，程序的开始指令为：

```
PUSH DS
XOR AX, AX
PUSH AX
```

堆栈情况



程序结束时的最后一条语句为RET，就把压入堆栈的PSP段的段基址和偏移量0000H弹出并送入CS和IP，转而执行返回DOS的指令INT 20H。



DATAS SEGMENT

MSG DB 'Hello world','\$'

DATAS ENDS

CODES SEGMENT

MAIN PROC FAR

ASSUME CS:CODES,DS:DATAS

START:

push ds

xor ax,ax ;mov ax,0

push ax

mov ax,DATAS

mov ds,ax

mov dx,OFFSET MSG

mov ah,09h

int 21h

ret ; 返回DOS

MAIN ENDP

CODES ENDS

END START



方法二:

在用户程序结束时，用下面两条指令：

MOV AH, 4CH

INT 21H



示例:

```
SSEG  SEGMENT          STACK                      ;设置堆栈段
SKTOP  DB              20      DUP(0)
SSEG  ENDS              ;堆栈段结束
;-----
DSEG  SEGMENT          ;设置数据段
STRING DB 'HELLO WORLD!$'
DSEG  ENDS              ;数据段结束
;-----
CSEG  SEGMENT          ;设置代码段
      ASSUME           CS:CSEG,DS:DSEG,SS:SSEG
START: MOV  AX, DSEG      ;将数据段起始地址装入DS
      MOV  DS, AX
      MOV  AX, SSEG      ;将堆栈段起始地址装入SS
      MOV  SS, AX
      MOV  SP, SIZE  SKTOP ;设置堆栈指针
      MOV  DX, OFFSET STRING
      MOV  AH, 9
      INT  21H
      MOV  AH, 4CH        ;返回DOS
      INT  21H
CSEG  ENDS              ;代码段结束
      END  START          ;汇编语言源程序结束
```

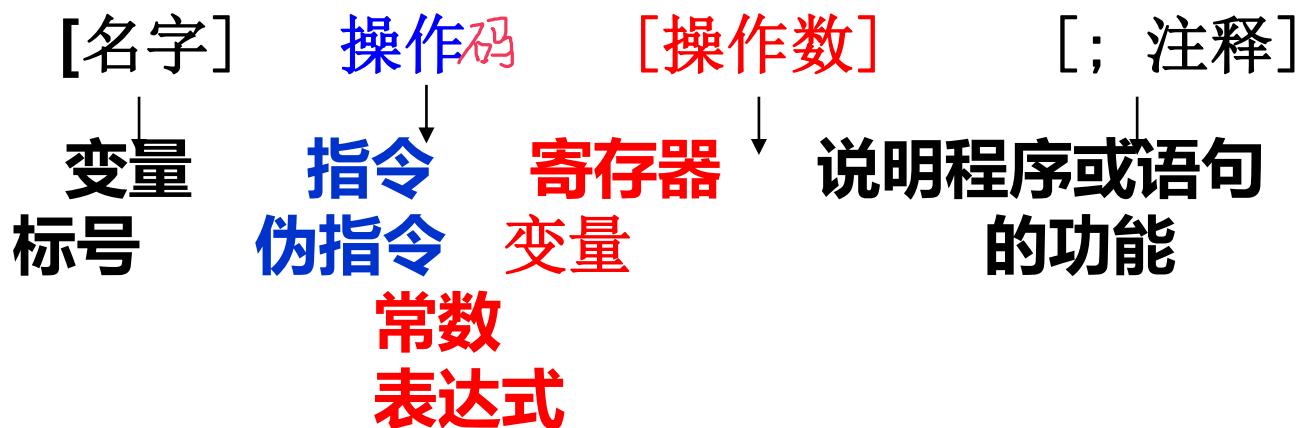
3.2 汇编语言源程序的结构

伪指令：伪指令不是处理器运行的指令，而是程序员给汇编编译程序下达的命令。是在编译源程序期间由汇编编译程序执行的命令。

伪指令语句：在汇编时不产生二进制代码，仅为编译程序提供汇编时所需要的信息的语句。



汇编语言的指令及伪指令语句格式:



例:

```
data SEGMENT      ;数据段  
var DB ?          } 可有可无  
data ENDS  
code SEGMENT      ;代码段  
    ASSUME CS:code,DS:data  
start: MOV AX, data  
       MOV DS, AX  
       MOV var, CL  
       MOV AH, 4CH  
       INT 21H      ;返回DOS  
code ENDS  
      END start
```

汇编语言源程序的段定义

段定义伪操作

格式1: 段名 **SEGMENT**

.....

段名 **ENDS**

格式2: 段名 **SEGMENT** [定位类型, 连接方式, ‘类别’]

.....

段名 **ENDS**



类型及属性的说明

定位类型：说明段的起始地址应有怎样的边界值

PARA: 指定段的地址从小段边界开始. -----默认值

BYTE: 该段可从任何字节开始.

WORD: 该段必须从字边界开始.

PAGE: 该段必须从页的边界开始. (每256个字节为一页)

连接类型：说明程序连接时段的合并方法

NONE(PRIVATE): 该段为私有段，连接时不与其他同名的分段合并 ---默认值

PUBLIC: 该段连接时与其他同名的分段连在一起

COMMON: 该段在连接时与其它同名的段具有相同的起始地址.

AT: 使段的起始地址为后面表达式的16位值.

STACK: 指定该段为堆栈的一部分.

MEMORY: 链接程序把本段定位在其他段之上（高地址区域）

类别：给出连接时组成段组的类别名. **'STACK', 'CODE', 'DATA'**

注意：

▲ 段定义由伪操作**SEGMENT**开始、**ENDS**结束。

其中：SEGMENT 和ENDS 必须成对出现，
且语句前必须有段名，**段名必须相同**。

▲ 程序中可以定义多个段。

▲ 程序经汇编、连接及装入内存后，段名为一具体的段值。



示例:

```
SSEG  SEGMENT          STACK                      ;设置堆栈段
SKTOP DB                20          DUP(0)
SSEG  ENDS                      ;堆栈段结束
;-----
DSEG  SEGMENT          ;设置数据段
STRING DB 'HELLO WORLD!$'
DSEG  ENDS                  ;数据段结束
;-----
CSEG  SEGMENT          ;设置代码段
                ASSUME      CS:CSEG,DS:DSEG,SS:SSEG
START:  MOV     AX, DSEG      ;将数据段起始地址装入DS
        MOV     DS, AX
        MOV     AX, SSEG      ;将堆栈段起始地址装入SS
        MOV     SS, AX
        MOV     SP, SIZE     SKTOP      ;设置堆栈指针
        MOV     DX, OFFSET STRING
        MOV     AH, 9
        INT     21H
        MOV     AX, 4C00H      ;返回DOS
        INT     21H
CSEG  ENDS                  ;代码段结束
                END      START      ;汇编语言源程序结束
```

汇编语言的段寻址 (ASSUME伪操作)

格式 **ASSUME 段寄存器: 段名 [, 段寄存器:段名, ...]**

其中: **段寄存器**为CS、DS、ES、SS中的一个

段名为用伪操作SEGMENT定义过的段名

例 **ASSUME CS: cc , DS:aa**

▲ ASSUME 伪操作的作用

明确段和段寄存器的关系，指示汇编程序指令中用到的标号、过程及变量所在的段。

注意: ASSUME只起指示作用，并不能把段地址装入段寄存器，必须在代码段中把段地址装入相应的段寄存器（代码段除外）



段寄存器的装填:

MOV	AX, 数据段名	}	装填 数据段
MOV	DS, AX		
MOV	AX, 附加数据段名	}	装填 附加数据段
MOV	ES, AX		
MOV	AX, 堆栈段名	}	装填 堆栈段
MOV	SS, AX		

说明: DS, ES须在程序段中进行人工装填, ss段已有初始化, 但也可通过装填进行修改

CS由OS根据文件头中的信息自动装填。


例: *data_seg1 segment*
 ...
 data_seg1 ends ; 定义数据段
 data_seg2 segment
 ...
 data_seg2 ends ; 定义附加段

第1 → *code_seg segment*

第2 → *assume cs:code_seg, ds:data_seg1, es:data_seg2*
 start:

第3 → *mov ax, data_seg1*
 mov ds, ax
 mov ax, data_seg2
 mov es, ax ; 段地址→段寄存器

 ...
 code_seg ends
 end start



(汇编结束语句) END伪操作

格式 END [标号]  启动地址

- 多个程序模块相连，则只有主程序要使用标号，其他子程序则只用END而不用指定标号

- 作用是指示源程序到此结束。

汇编程序对 END 之后的语句不进行处理。

程序中所有有效语句应放在 END 语句之前。

- 源程序中必须有 END 结束语句。

汇编程序对无 END 语句的源程序不进行处理，只给出无 END 语句错误信息。

示例:

```
SSEG  SEGMENT          STACK                      ;设置堆栈段
SKTOP  DB              20      DUP(0)
SSEG  ENDS              ;堆栈段结束
;-----
DSEG  SEGMENT          ;设置数据段
STRING DB 'HELLO WORLD!$'
DSEG  ENDS              ;数据段结束
;-----
CSEG  SEGMENT          ;设置代码段
      ASSUME           CS:CSEG,DS:DSEG,SS:SSEG
START:  MOV    AX,      DSEG          ;将数据段起始地址装入DS
        MOV    DS,      AX
        MOV    AX,      SSEG         ;将堆栈段起始地址装入SS
        MOV    SS,      AX
        MOV    SP,      SIZE  SKTOP  ;设置堆栈指针
        MOV    DX,      OFFSET STRING
        MOV    AH,      9
        INT    21H
        MOV    AX,      4C00H        ;返回DOS
        INT    21H
CSEG  ENDS              ;代码段结束
      END    START              ;汇编语言源程序结束
```

标准程序返回方式

方法1：在汇编程序最后加

MOV AH,4CH ; 功能模块号

INT 21H ;中断功能调用

cpu



方法2： 使用中断调用20H（过程结束）

<程序名> PROC FAR

PUSH DS

SUB AX, AX (MOV AX, 0)

PUSH AX

.....

RET

<程序名> ENDP



汇编语言的过程定义

过程（子程序）定义伪操作：

过程名 **PROC** 属性

.....

RET

过程名 **ENDP**

NEAR
(FAR)

与标号类似，
是子程序入口的符号地址

过程的属性

有NEAR和FAR两种属性

过程属性的确定原则：

NEAR属性：调用程序和子程序在同一代码段中
(段内调用)

FAR 属性：调用程序和子程序不在同一代码段中
(段间调用)



例 调用程序和子程序在同一代码段中

NEAR

code segment

main proc far

.....

call subr1

.....

ret

main endp

subr1 proc near

.....

ret

subr1 endp

code ends

code segment

main proc far

.....

call subr1

.....

ret

subr1 proc near

.....

ret

subr1 endp

main endp

code ends

把过程写成嵌套的形式

例 调用程序和子程序不在同一代码段中

FAR

segx segment

```
.....  
subt proc far  
.....  
ret  
subt endp
```

.....
call subt

segx ends

segx segment
segment

.....
call subt

segx ends

FAR

3.3 数据定义

一、常量、变量及标号

常量：表示固定的数值。如常数、字符、字符串等

常数：二进制(B)，八进制(Q)，十六进制(H)，十进制(D)(默认)

字符或字符串用单引号或双引号括起来

变量：代表存放在某些存储单元的数据，这些数据在程序的运行期间随时可以修改。

在程序中以变量名的形式出现。



每个变量都有三个属性

- 1) 段属性(SEG) 变量所在的存储单元的段基址
- 2) 偏移量属性(OFFSET) 变量所在的存储单元距段起点的字节数
- 3) 类型属性(TYPE) ^{属性操作符} 一个单位变量占用存储单元的字节数，分为：
DB(1个字节) **DW**(2个字节) **DD**(4个字节)
_{8 16}

标号：是代码段中某一指令的地址。

标号由下列字符组成：

字母：**A~Z, a~z**；数字：**0~9**；特殊字符：**? . @ _ \$**

数字不能作标识符的第一个字符，标识符最长为**31**个字符。

标号后面跟冒号，代表该行指令的起始地址，标号可以被转移指令、被调用指令直接引用。



标号也有3个属性:

- 1) 段属性(SEG) 该条指令所在段的段基址
- 2) 段内偏移量属性(OFFSET) 该指令的偏移地址(距段起点的字节数)
- 3) 类型属性 表示该标号是作为段内还是段间被调用或转移的, 该属性有两个值:
 - a) NEAR(FFFFH, -1): 本标号只能被标号所在段的转移或调用指令所访问(段内转移), 标号后有冒号;
 - b) FAR (FFFEH, -2): 本标号可被其他段(不是标号所在段)的转移或调用指令访问(段间转移)。

指明标号属性:

标号名 LABEL FAR

二、数据定义和分配数据单元的伪操作

DB: 定义字节，其后的每个操作数占有一个字节单元，连续存放；

BUFFER DB 2, 3, 5

BUFFER

02H
03H
05H

DW: 定义字，其后的每个操作数占有两个字节；

BUF DW 2, 3, 5

BUF

02H
00H
03H
00H
05H
00H

DD: 定义双字，其后每个操作数占4个字节；



► **另一种格式:**

若**仅保留单元**，**不初始化**，用**?**代替初值；

若**数据重复**，用 **n DUP()**代替，**n**为重复次数。

ARRAY DB 100 DUP(?)

保留100个字节，首地址为ARRAY，不初始化，即100个字节内均为随机值

DATA1 DB 100 DUP('AB')

初始化**200**个字节，内有100个41H, 42H



定义字符串变量(只能用DB定义)

```
data SEGMENT
```

```
str1 DB 'TsingHua '
```

```
str2 DB 'INPUT:', 0dH, 0aH, '$'
```

```
data ENDS
```

注意： 3个及其以上的字符，
只能用DB定义

```
str1 DW 'abcd'
```

```
str2 DD 'abcd'
```



str1→	54	'T'
	73	's'
	69	'i'
	6e	'n'
	67	'g'
	48	'H'
	75	'u'
	61	'a'
str2→	49	'I'
	4e	'N'
	50	'P'
	55	'U'
	54	'T'
	3a	':'
	0d	0dH
	0a	0aH
	24	'\$'

三、等值伪操作

名字 EQU 表达式

名字 = 表达式

用一个名字来代表一个常数或表达式，在汇编时，凡是出现该名字的地方就用定义的数据替代。

```
TIMES EQU 50
```

```
BUF DB TIMES DUP(?)
```

等效于 **BUF DB 50 DUP(?)**

```
ALPHA EQU 256
```

```
BETA = ALPHA-2
```

用EQU赋值的名字不能重新赋值

3.4 汇编语言的运算符

汇编语言运算符：是编译程序在编译时计算的，与运算指令不同，指令是在程序运行时计算的。

一、算术运算符

＋，－，＊，／，MOD，SHL，SHR

MOV AX, A+B

CC EQU DD SHL 2

二、逻辑运算符

AND, OR, XOR, NOT

C EQU B-A

► **AND AX, C AND 0FH**

三、关系运算符

EQ, NE, LT, GT, LE, GE **less than , great than**

结果产生一个逻辑值，**真为0FFFFH, 假为0000H**

$$AX = \begin{cases} 5 & choice < 20 \\ 6 & choice \geq 20 \end{cases}$$

MOV AX, ((choice LT 20) AND 5) OR
((choice GE 20) AND 6)



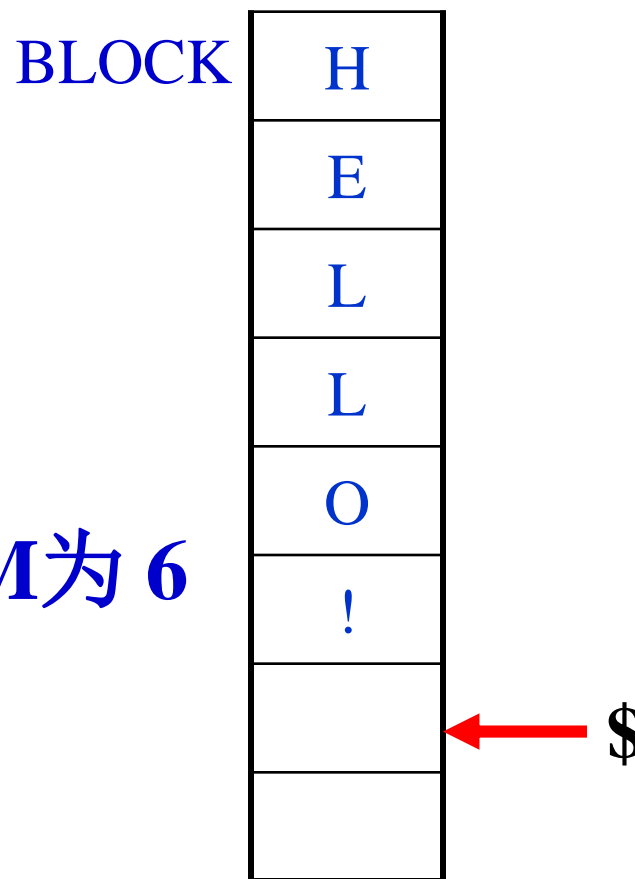
四、值返回符

1) \$运算符

\$: 当前地址偏移量的值

BLOCK DB 'HELLO!'

NUM EQU \$-BLOCK ;NUM为 6



2) SEG 和OFFSET

SEG: 求标号或变量的段基址

OFFSET: 求标号或变量的距段首址的偏移量

DATA SEGMENT

A DB 12

B DW 23, 25

DATA ENDS

.....

MOV BX, OFFSET B ; BX: 0001H

MOV AX, SEG B ; AX: DS值 *MOV Ax, DS*

▶ **LEA BX, B ; BX: 0001H 等价** **MOV BX, OFFSET B**

3) TYPE 标号或符号的类型值

对变量：表示变量的字节数 **DB 1, DW 2, DD 4**

对标号：表示过程或指令地址的调用类型 **(NEAR) -1** 或 **(FAR) -2**

DATA SEGMENT

A DB 12

B DW 23, 25

DATA ENDS

.....

MOV AX, TYPE A ;AX: 0001H

MOV BX, TYPE B ;BX: 0002H



4) LENGTH 和 SIZE

LENGTH: 对DUP情况下, 变量的项数或元素个数, 在其他情况下该项属性为1;

SIZE: 对操作数分配的字节数。

$$\text{SIZE} = \text{LENGTH} \times \text{TYPE}$$

A DB '1234'

B DW 5 DUP(2, 3 DUP(0))

C DW 'AB', 'C', 'D'

L1: MOV AL, TYPE B ; AL: 2

MOV BL, LENGTH B ; BL: 5

MOV AH, SIZE A ; AH: 1

MOV BH, SIZE C ; BH: 2

MOV CL, TYPE L1 ; CL: 0FFH

MOV CH, SIZE B ; CH: 0AH

5) HIGH 和LOW

HIGH: 对操作数取高字节;

LOW: 对操作数取低字节。

NUM EQU 0CDEFH

.....

MOV AH, HIGH NUM ; AH: 0CDH

MOV AL, LOW NUM ; AL: 0EFH



五、属性运算符

用来给指令中的操作数指定一个临时的属性，而暂时忽略操作数定义时的属性。

1) **PTR** 定义操作数为新的类型

新类型 PTR 操作数

操作数可以是存储器的地址或标号名

F1 DW 1234H

F2 DB 23H, 56H, 18H

.....

MOV AL, BYTE PTR F1 ; AL: 34H

▶ **MOV AX, WORD PTR F2 ; AX: 5623H**

F1	34H
	12H
F2	23H
	56H
	18H

DAT1 DB 12H, 34H

DAT2 DB 56H, 78H

.....

MOV AX, WORD PTR DAT1 ; AX: 3412H

CMP AX, WORD PTR DAT2 ; 7856H

JA L1

MOV BX, WORD PTR DAT2

MOV WORD PTR DAT2, AX

MOV WORD PTR DAT1, BX

L1: HLT

DAT1单元的值是: 56H

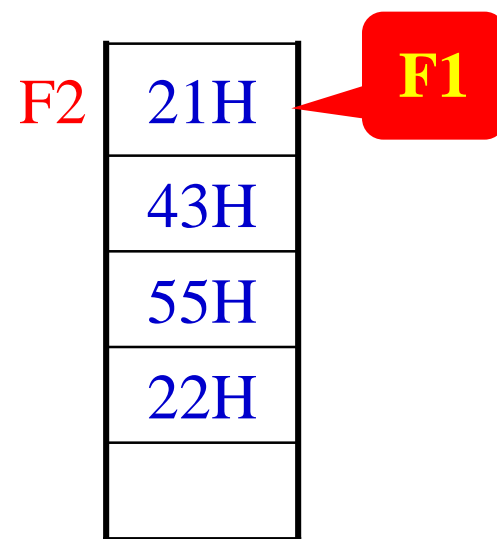


2) THIS 指定新类型

像PTR一样可用来建立一个某种类型的存储器地址操作数，而没有为它分配存储器。新的存储器操作数的段和偏移量部分就是下一个能分配的存储单元的段和偏移量。

THIS 类型名

```
F1 EQU THIS BYTE
F2 DW 4321H, 2255H
.....
MOV AL, F1      ; AL: 21H
MOV AX, F2      ; AX: 4321H
```



► F1和F2具有相同的段基地址和偏移地址，但类型不同。

3) 段超越

强迫当前指令的操作数按指定的段基地址寻址。

MOV AX, ES:[BX]

4) SHORT

用于无条件转移指令JMP，通知编译器，转移的目标地址在+127~-128之间。

JMP 标号；是三字节指令

JMP SHORT 标号；是两字节指令



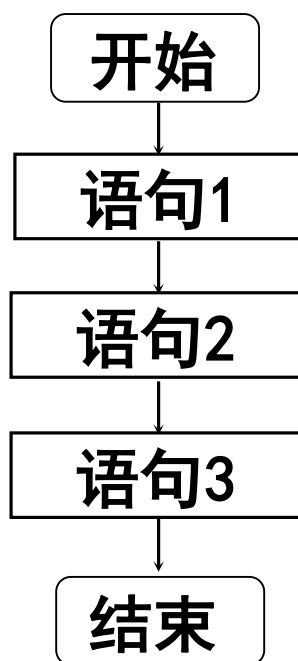
3.5 基本结构程序设计

主要程序结构

- 顺序结构
- 分支结构
- 循环结构
- 子程序结构



3.5.1 顺序结构程序设计



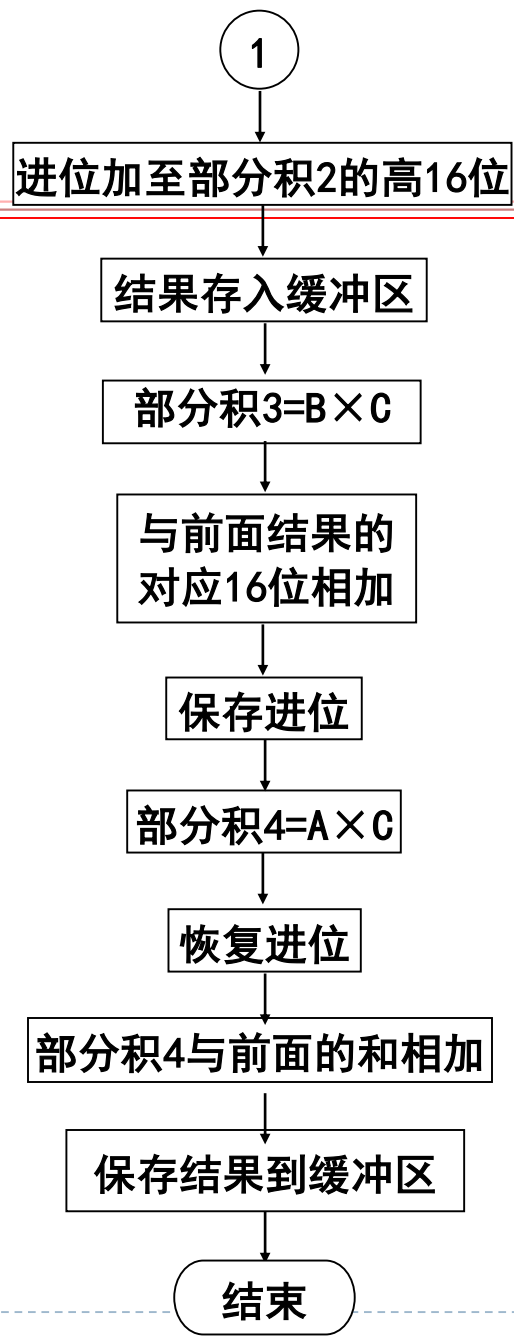
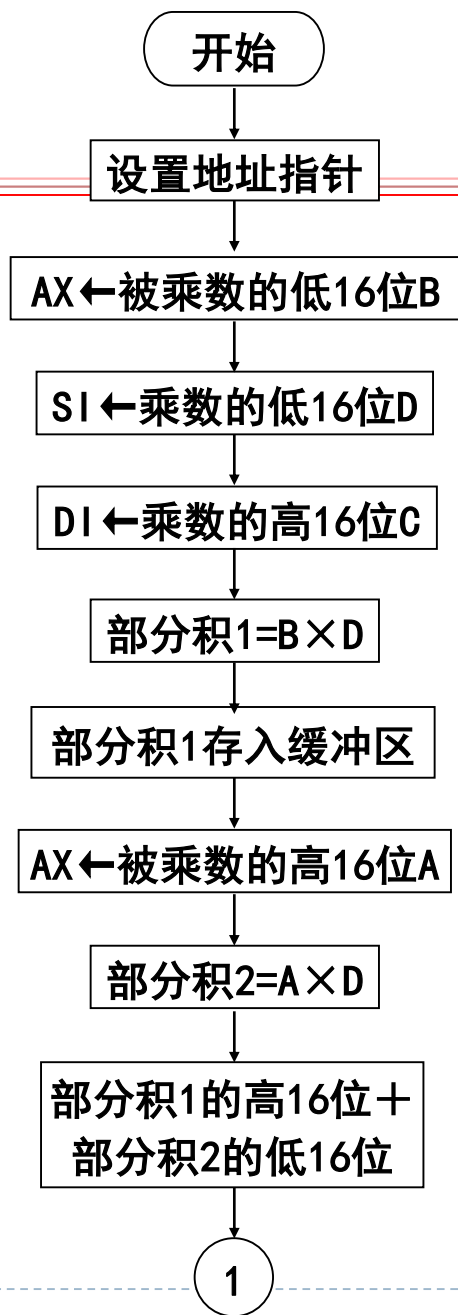
顺序结构流程



▶ 例 两个32位无符号数乘法程序。

▶ 1) 明确任务，确定算法。

▶ 2) 绘流程图



流程图

3) 根据流程图编写汇编语言程序

```
DATA            SEGMENT
MULNUN          DW      0000, 0FFFFH, 0000, 0FFFFH, 4 DUP (0)
DATA            ENDS
STACK           SEGMENT PARA STACK'STACK'
                DB  100  DUP (?)
STACK           ENDS
CODE            SEGMENT
    ASSRME      CS: CODE, DS: DATA, SS: STACK, ES: DATA
    MAIN        PROC          FAR
START:          PUSH      DS
                MOV       AX,  0
                PUSH      AX
                MOV       AX,  DATA
```



MOV DS, AX
MOV ES, AX
LEA BX, MULNUM ; 设置地址指针

MULU32: MOV AX, [BX]; B
MOV SI, [BX+4]; D
MOV DI, [BX+6]; C
MUL SI; B*D
MOV [BX+8], AX
MOV [BX+0AH], DX
MOV AX, [BX+2]; A*D
MUL SI;
ADD AX, [BX+0AH]
ADC DX, 0
MOV [BX+0AH], AX
MOV [BX+0CH], DX
MOV AX, [BX]; B*C
MUL DI;



ADD AX, [BX+0AH]

ADC DX, [BX+0CH]

MOV [BX+0AH], AX

MOV [BX+0CH], DX

PUSHF; A*C

MOV AX, [BX+2]

MUL DI

POPF

ADC AX, [BX+0CH]

ADC DX, 0

MOV [BX+0CH], AX

MOV [BX+0EH], DX

RET

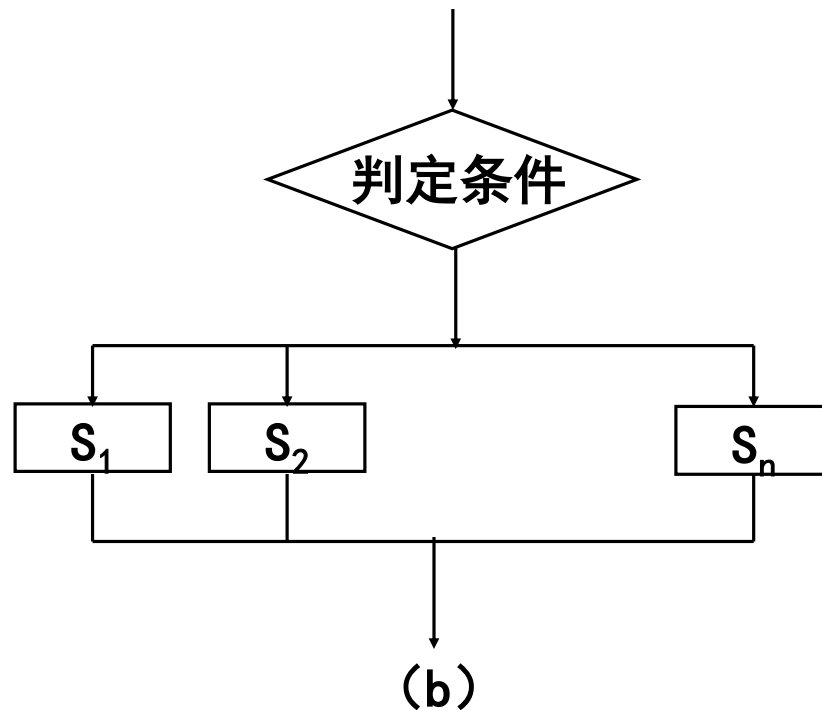
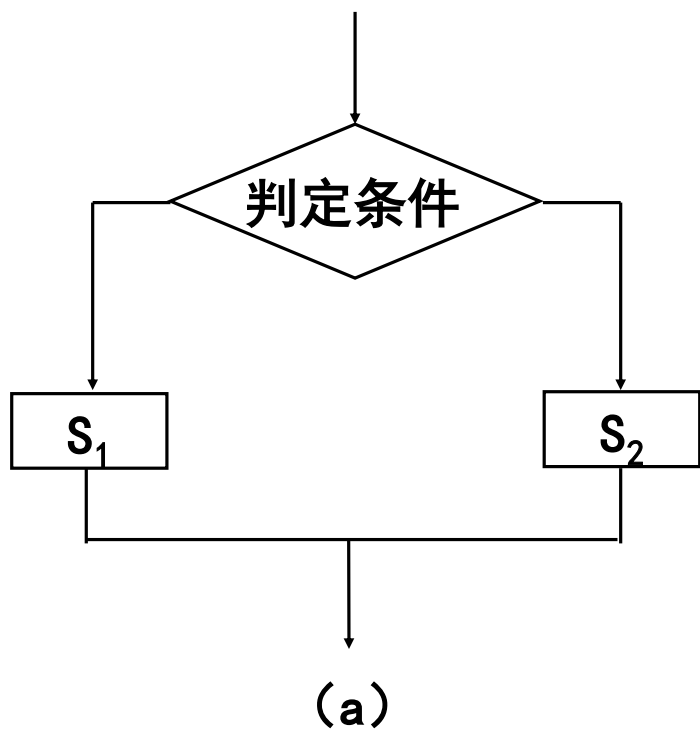
MAIN ENDP

CODE ENDS

END START



3.5.2 分支结构

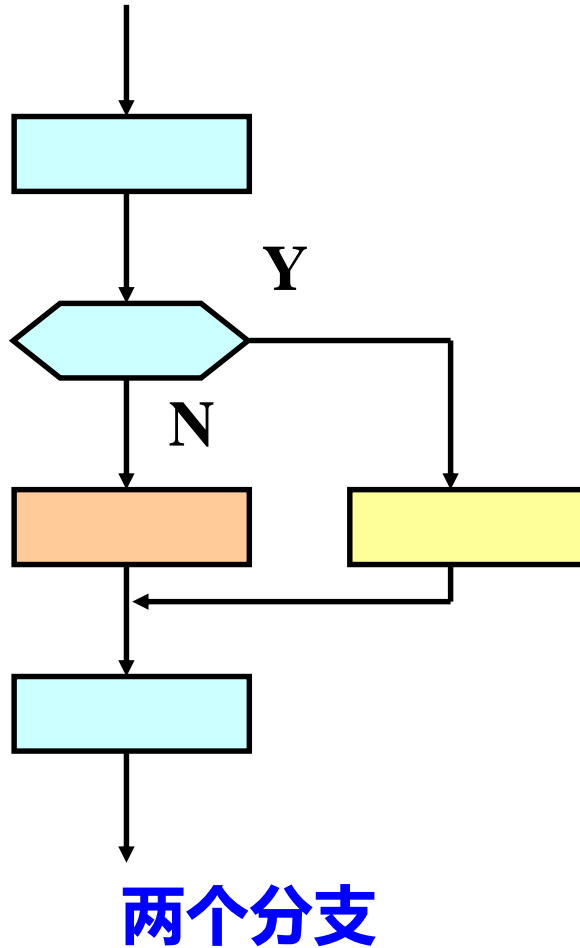


分支程序的结构形式

(a) 二分支结构

(b) 多分支结构

二分支结构形式



、 、 、
CMP AL, BL
JG great

AL≤BL处理

JMP exit

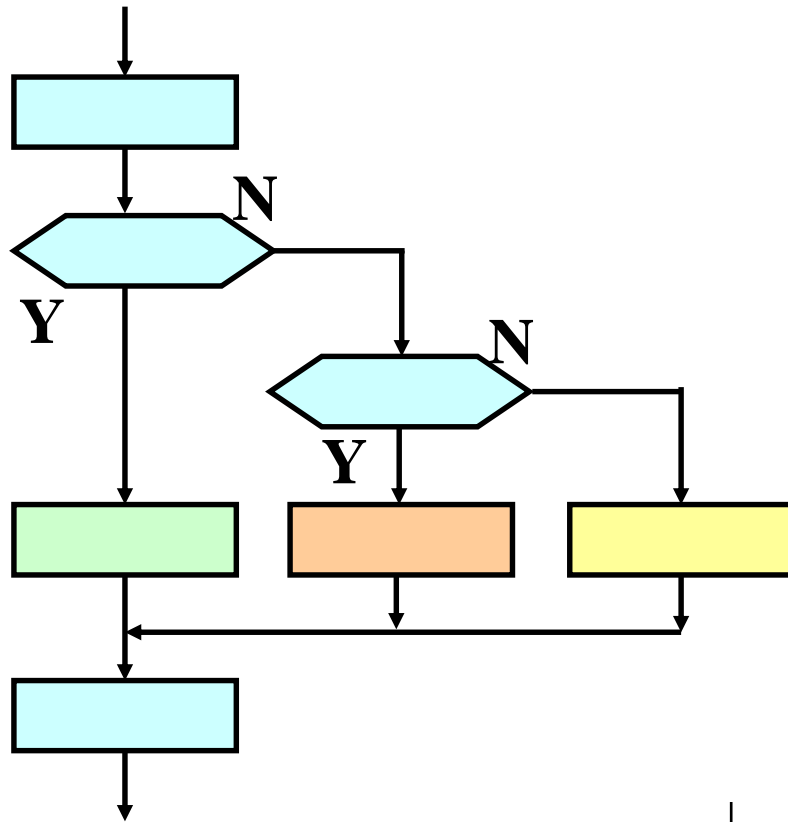
great:

AL>BL处理

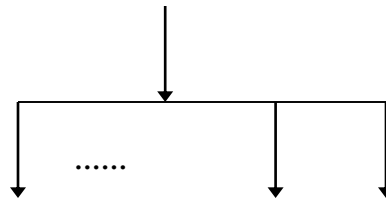
exit:

、 、 、
、 、 、

多分支结构形式



三个分支



、 、 、
CMP AL, 0
JG great
JL less

AL=0处理

JMP exit

less:

AL<0处理

JMP exit

great:

AL>0处理

exit:

、 、 、

例3-3

$$\text{实现函数 } Y = \begin{cases} 1 & ; X > 0 \\ 0 & ; X = 0 \\ -1 & ; X < 0 \end{cases}$$



```

DATA SEGMENT
    X DW 12
    Y DW ?
DATA ENDS

STACK SEGMENT STACK 'STACK'
    DB 100 DUP (?)
STACK ENDS

CODE SEGMENT PARA 'CODE'
    ASSUME CS: CODE, DS: DATA, SS: STACK

    SIGN PROC FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA    ; 装填DS
        MOV     DS, AX
        MOV     AX, X
        AND     AX, AX       ; 建立标志
        JZ      ZERO        ; X = 0转ZERO
        JNS     PLUS        ; X > 0转PLUS
        MOV     BX, 0FFFFH   ; X < 0令BX = - 1
        JMP     DONE
    ZERO: MOV    BX, 0
        JMP     DONE
    PLUS: MOV    BX, 1
    DONE: MOV    Y, BX       ; 存放结果
    RET
SIGN ENDP

CODE ENDS
END SIGN

```

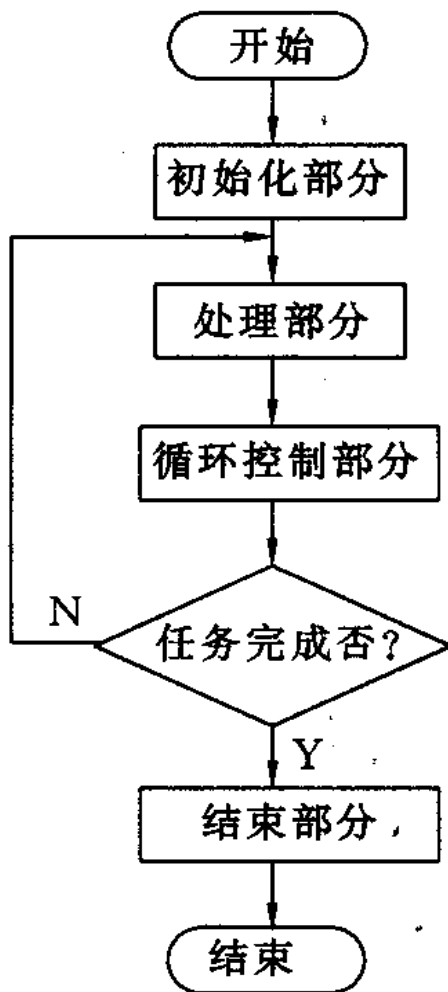
3.5.3 循环结构

循环结构是程序设计中最常用的结构。

凡需要重复做的工作，在计算机中都可以用循环结构程序来实现。

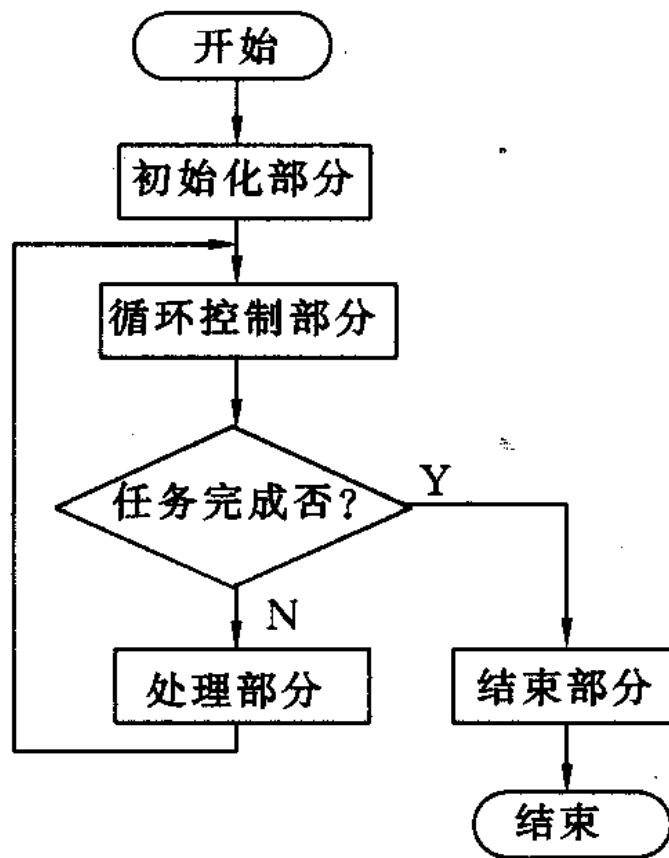


DO-UNTIL型循环 (直到条件成立退出循环)



不允许0次循环

WHILE型循环 (当条件成立进入循环)



允许0次循环

**例：100个字节数据从3000H:0100H送到
3000H:0200H单元.**



循环结构1:

```
    mov ax, 3000h
    mov ds, ax
    mov si, 0100h
    mov di, 0200h
    mov cx, 100
11:  mov al, [si]
    mov [di], al
    add si, 1
    add di, 1
    loop 11
```



循环结构2

mov ax, 3000h

mov ds, ax

mov si, 0100h

mov di, 0200h

mov cx, 100

inc cx

11: dec cx

jz 12

mov al, [si]

mov [di], al

inc si

inc di

jmp 11

12: mov AH,4CH

int 21h



多重循环基本方法与单重循环相同，
但要注意：

- 1、分别考虑各重循环的控制条件及其程序实现，相互之间不能混淆
- 2、每次从外层循环再次进入内层循环时，初始条件要重新设置



例：延时100ms程序

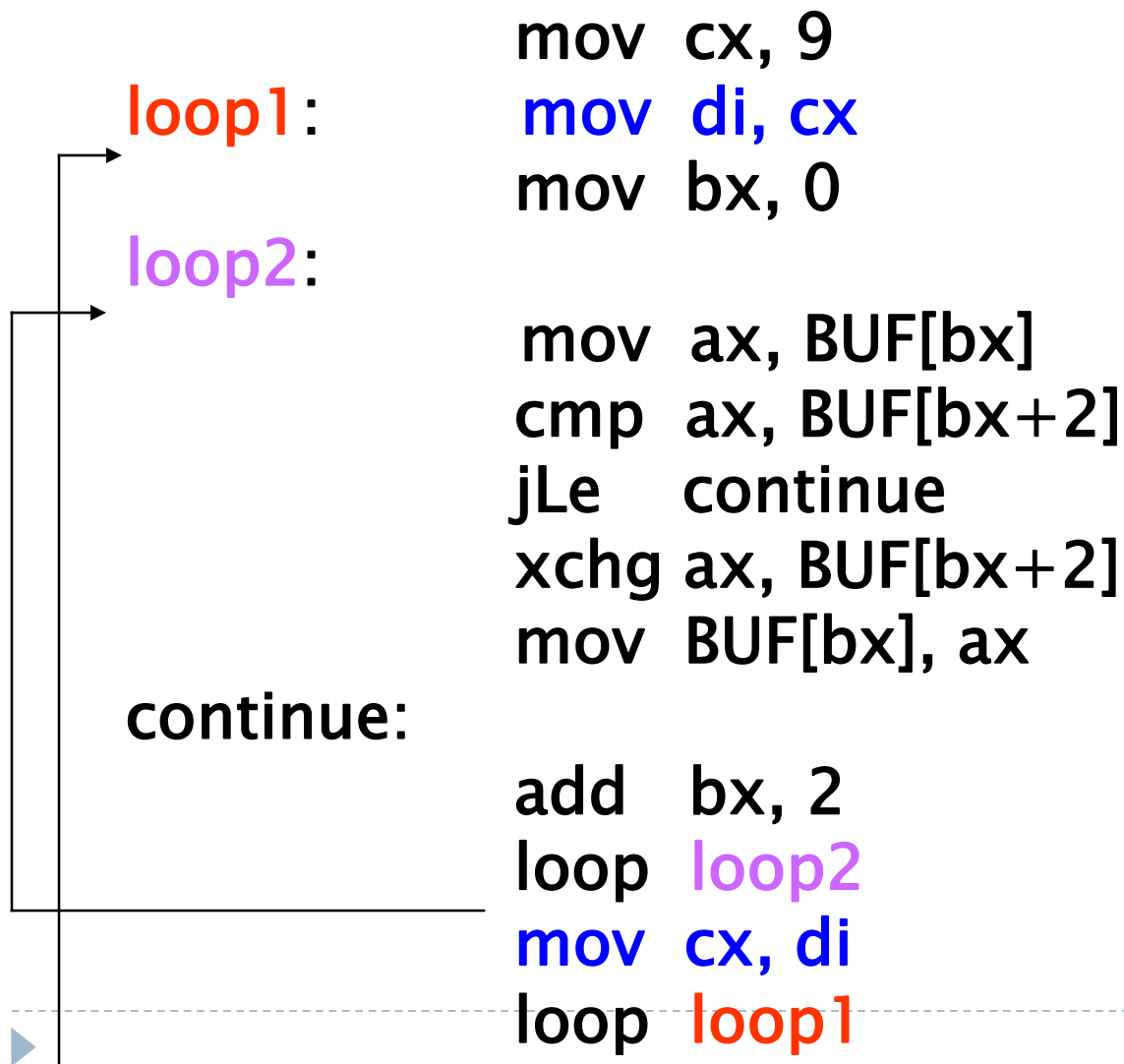
```
▶          MOV BL, 10
▶ DELAY:    MOV CX, 2801
▶          WAIT: LOOP WAIT    ; T0
▶          DEC BL
▶          JNZ      DELAY
```

} 10mS

▶ **延时时间=2801×T0×10**

例：将首地址为BUF的字数组从小到大排序（气泡排序法）

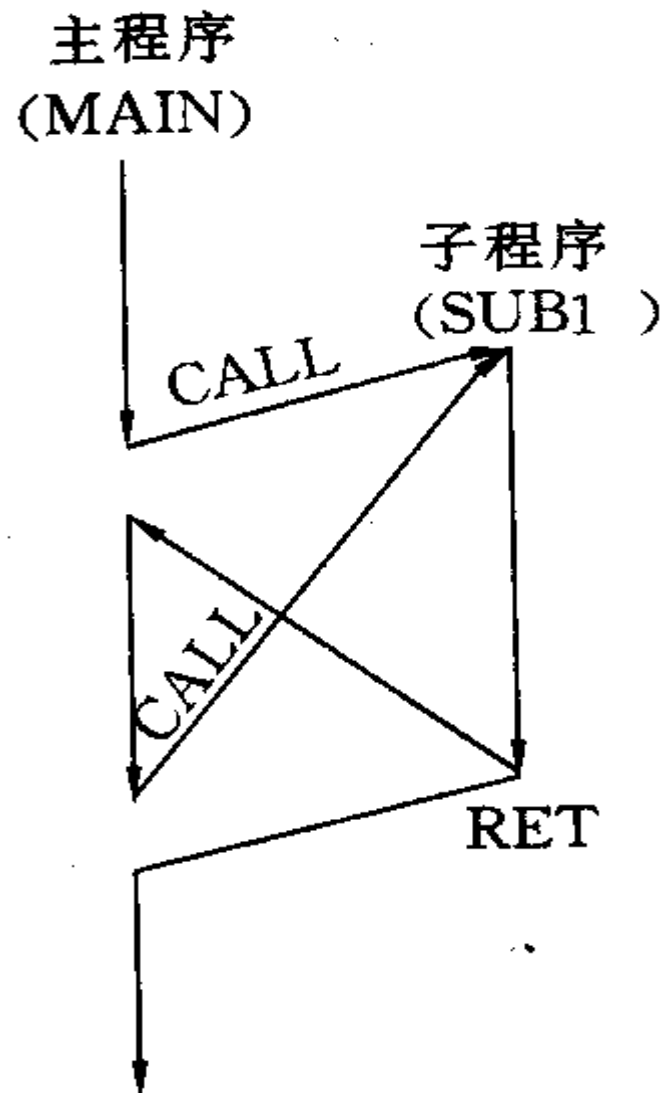
BUF dw 100,30,78,99,15,-1,66,54,189,256



3.5.4 子程序结构

- ▶ 若一段指令或在一个程序中多处使用，或在多个程序中用到，则通常在这段指令当做一个独立的模块出来，称为**子程序**（或**过程**）。
- ▶ 子程序是模块化设计的重要手段，具有以下优点：
 - (1) 简化程序设计过程，节省程序设计时间；
 - (2) 缩短了程序的长度，节省了存储空间；
 - (3) 增加了程序的可读性，便于对程序进行修改；
 - (4) 方便了程序的模块化、结构化和自顶向下的设计。

- ▶ 通用性：入口参数
- ▶ 出口参数
- ▶ 参数传递方法：
- ▶ 1.用REG,少量参数
- ▶ 2.用程序M中的参数表传递
- ▶ 3.用堆栈传递



(一) 用程序存储器中的参数表传递参数

在主程序中,将参数放在
CALL后面.例:
主程序

```
...  
CALL ADDSUB  
NUM1 DD 12345678H ;被加数  
NUM2 DD 5678H      ;加数  
BUFFER DD ?         ;和  
。 。 。 。 ; 返回地址
```

+11	
+10	
+9	
BUFFER	
	00
+7	00
+6	56
+5	78
NUM2	12
	34
+3	56
+2	78
+1	

NUM1

子程序:

PUSH BP

若EA(NUM1)=IP0

MOV BP,SP

...

MOV BX,[BP+2]

MOV CX,CS:[BX]

MOV SI,CS:[BX+2]

...

ADD BX,12 ;IP0+12

MOV [BP+2],BX;返回地址入栈 原SP

...

POP BP

RET

→ -4

-3

→ -2

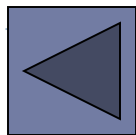
-1

BPL

BPH

IP0 L

IP0 H



若传送某变量的地址：

主程序：

CALL XXX

NUM1 DW OFFSET VAL1, SEG VAL1

子程序：

LDS SI, DWORD PTR CS: [BX]

; DS: SI=VAL1 的物理地址



(2) 用堆栈传递参数

堆栈传递适用于参数较多且子程序有嵌套、递归调用的情况。在主程序中把参数压入堆栈后调用子程序，子程序中从堆栈中取出。

例：数组求和子程序。 已知数组A由100个字数据组成，数组B由50个字数据组成，要求用堆栈传送参数的子程序结构编程，试分别求出这两个数组元素之和。



源程序如下:

```
DATA    SEGMENT

CNTA    DW    100

ARYA    DW    100  DUP ( ? )

SUMA    DD    ?

CNTB    DW    50

ARYB    DW    50  DUP ( ? )

SUMB    DD    ?

DATA    ENDS

CSEG    SEGMENT

        ASSUME  CS: CSEG, DS: DATA

START:  MOV     AX, DATA

        MOV     DS, AX
```

MOV AX, OFFSET ARYA

PUSH AX

MOV AX, OFFSET CNTA

PUSH AX

MOV AX, OFFSET SUMA

PUSH AX

CALL NEAR PTR RADD

MOV AX, OFFSET ARYB

PUSH AX

MOV AX, OFFSET CNTB

PUSH AX

MOV AX, OFFSET SUMB

PUSH AX

CALL NEAR PTR PADD

MOV AH, 4CH

INT 21H



RADD PROC NEAR

PUSH BP

MOV BP, SP

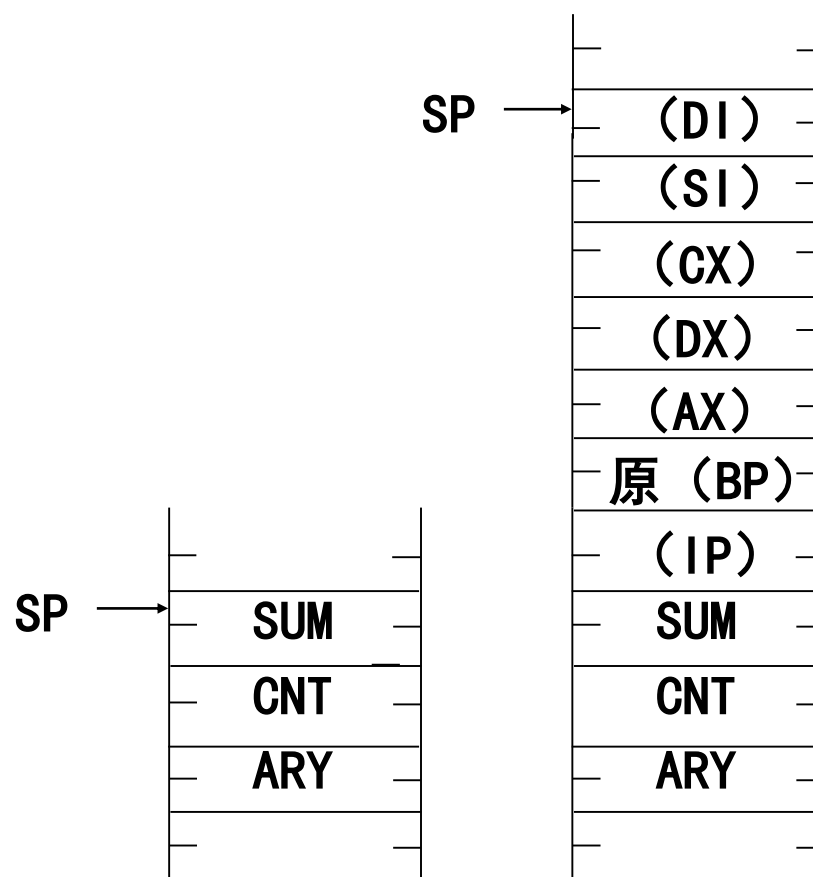
PUSH AX

PUSH DX

PUSH CX

PUSH SI

PUSH DI



调用前

当前

	MOV	SI, [BP + 8] ;数组首地址送SI
	MOV	DI, [BP + 6]
	MOV	CX, [DI] ; 数组长度送CX
	MOV	DI, [BP + 4] ;存放和的地址送DI
	XOR	AX, AX
	MOV	DX, AX
NEXT:	ADD	AX, [SI]
	JNC	NOCAY
	INC	DX
NOCAY:	ADD	SI, 2
	LOOP	NEXT
	MOV	[DI] , AX
	MOV	[DI + 2] , DX



```
                POP    DI
                POP    SI
                POP    CX
                POP    DX
                POP    AX
                POP    BP
                RET     6
RADD           ENDP
CSEG          ENDS
              END     START
```



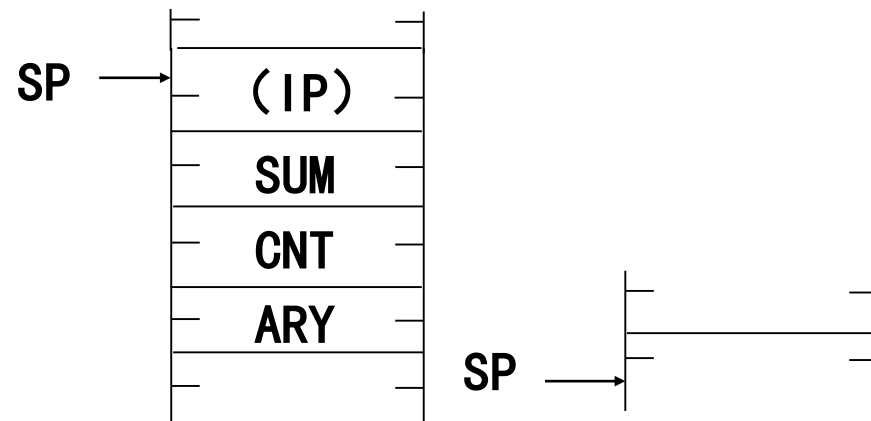


图 用堆栈传送参数时堆栈内容的变化情况

2 子程序嵌套与递归调用

子程序嵌套

子程序嵌套——是指一个子程序的内部再调用其他子程序。

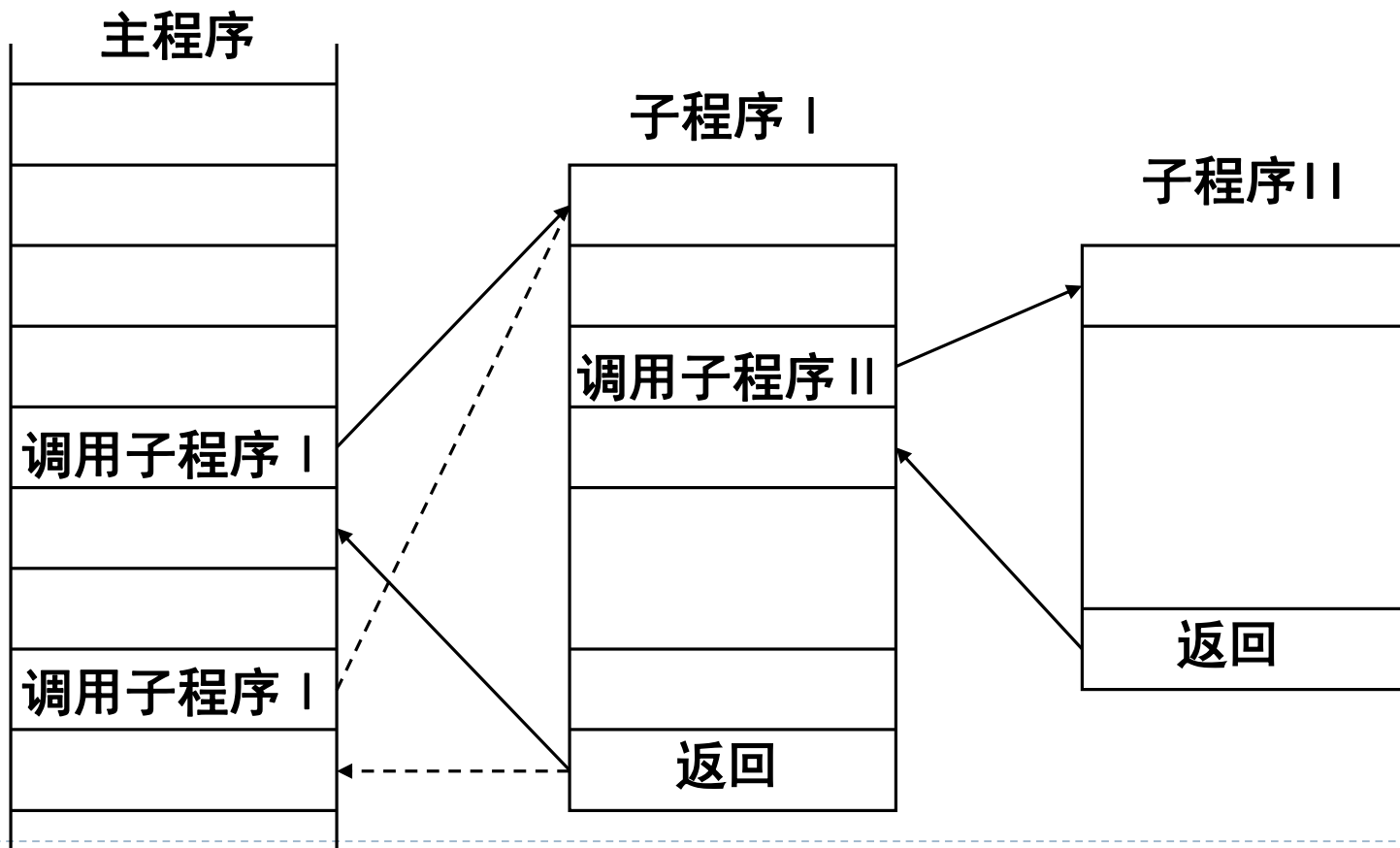


图 子程序嵌套

2、递归子程序

递归子程序——是这种具有递归调用性质的子程序。

递归子程序必须注意以下两点：

- 1) 注意现场的保护。
- 2) 注意递归结束条件。

例： 求变量NUMB的阶乘，把结果存入变量NJ。



‣ **计算 n!**

‣ **STA ENDPDATA SEGMENT**

‣ **NUM DB 3**

‣ **NJ DW ?**

‣ **DATA ENDS**

‣ **STACK SEGMENT STACK 'STACK'**

‣ **DB 200 DUP ('S')**

‣ **STACK ENDS**

‣ **CODE SEGMENT PARA 'CODE'**

‣ **ASSUME CS: CODE, DS: DATA, SS: STACK**

‣ **STA PROC FAR**

‣ **PUSH DS**

‣ **XOR AX, AX**

‣ **PUSH AX**

‣ **MOV AX, DATA**

‣ **MOV DS, AX**

‣ **MOV AH, 0**

‣ **MOV AL, NUM**

‣ **CALL FACTOR;** 入口和出口参数通过AX传递

‣ **X1: MOV NJ, AX**

‣ **RET**

; 求阶乘子程序

```
▶ FACTOR PROC NEAR  
▶ PUSH AX  
▶ SUB AX, 1  
▶ JNE FCON  
▶ POP AX  
▶ JMP RETUN  
▶ FCON: CALL FACTOR  
▶ X2: POP CX  
▶ MUL CL  
▶ RETUN: RET  
▶ FACTOR ENDP  
▶ STA ENDP  
▶ CODE ENDS  
▶ END STA
```

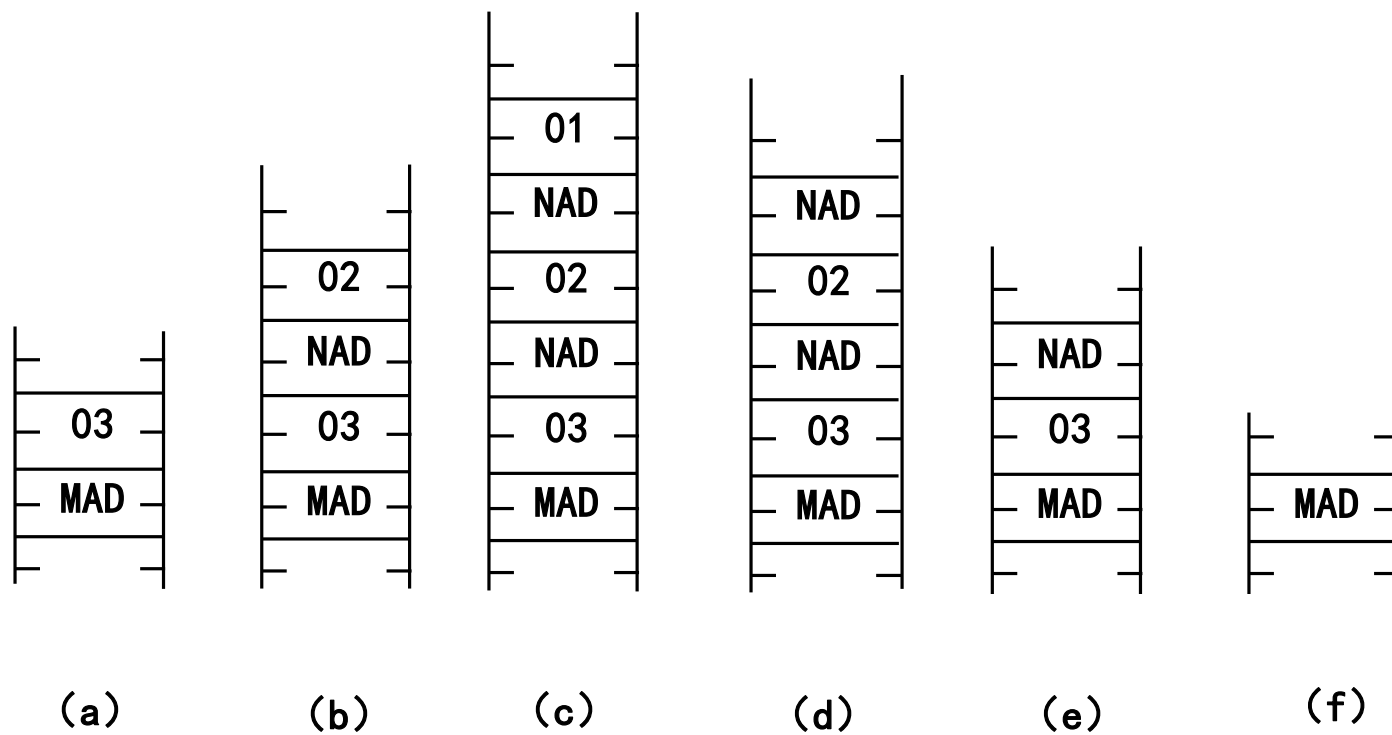
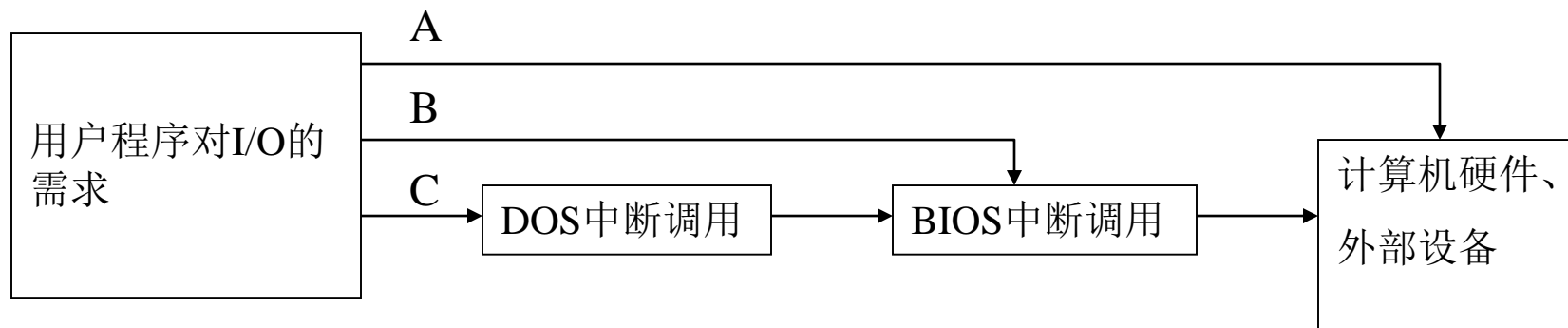


图 递归调用过程的堆栈变化情况

3.6 操作系统资源的使用



使用汇编语言编程对计算机硬件和外设进行控制和输入输出，有下面3种方法，

A——用户程序直接操作计算机硬件和外部设备。这样用户必须了解计算机硬件的细节，才能对硬件进行控制。

B——用户程序通过BIOS中断调用操作计算机硬件和外部设备，用户不必了解计算机硬件的细节，就可以对硬件进行操作，而且使程序更简洁，可读性和可移植性更好。

C——用户程序通过DOS中断调用来操作硬件，DOS中断调用为用户提供的功能更多。程序简洁，可读性和可移植性都很好。

3.6.1 DOS功能调用

在汇编程序中，如果用户程序要与输入输出设备打交道，就要调用现成的程序，**这些程序是DOS系统的一部分，随着DOS系统驻留内存**，用户需要按照这些程序要求的接口格式调用。

这些完成不同功能的子程序是以**中断服务程序**的方式提供的。

这些功能按不同的类别分成许多组，分别有不同的中断入口，在汇编程序中最常用的是INT 21H中断入口，也称为**DOS系统功能调用**。



调用方式:

这个中断入口中有许多小程序，每个小程序都被编上号，固定完成某一种功能，调用时有一定的格式，因为对某一个小程序而言，输入条件(入口参数)和输出结果(出口参数)的格式都是固定的。用户在使用时，既要**给出小程序的编号(AH)**，又要按照其提供的格式进行调用，才能正确完成操作。

只介绍与输入输出有关的一部分。

1) 在显示器上显示单个字符(2号功能)

功能： 在屏幕的光标处显示单个字符

入口参数： 要显示字符的ASCII码放在DL中

出口参数： 无

MOV DL, 'A' ; 在屏幕光标处显示字符A

MOV AH, 2 ; 提供调用功能号

INT 21H ; 系统功能调用

运行至此，屏幕上当前光标处显示字符A

2) 在屏幕上显示字符串(9号功能)

功能：在屏幕上当前光标处输出存储在内存数据段的一串字符串，该字符串以‘\$’结束。

入口参数：DS:DX指向欲显示字符串的首址

DATA SEGMENT

STRING DB ‘I am a student.\$’

DATA ENDS

.....

MOV DX, OFFSET STRING ; 指向字符串首址

MOV AH, 9 ; 提供调用功能号

INT 21H ; 系统功能调用

3) 带显示的键盘输入(1号功能)

功能：等待键盘输入，直到按下一个键。

入口参数：无

出口参数：键入键的ASCII码放在AL中，并在屏幕上显示该键。

MOV AH, 1 ; 提供调用功能号

INT 21H ; 系统功能调用

MOV [2000H], AL

程序运行到此
停下，等待用
户键盘输入

• 用户从键盘键入的键
的ASCII码进入AL中

4) 不带显示的键盘输入(7号功能)

功能：等待键盘输入，直到按下下一个键。

入口参数：无

出口参数：键入键的ASCII码放在AL中，但在屏幕上没有显示，常用于输入密码。

MOV AH, 7 ; 提供调用功能号

INT 21H ; 系统功能调用

MOV [2000H], AL

...

用户从键盘键入的键的ASCII码进入AL中

程序运行到此停下，等待用户键盘输入

该功能与1号功能类似，只是输入的字符不在屏幕上显示

5) 字符串输入(10号功能)

功能：等待从键盘输入一串字符到存储区的数据段，直到按下回车结束输入。

入口参数：DS:DX指向接收字符串的内存地址的首址，该地址的第一个字节是由用户设置的可输入字符串的最大字符数(含回车)

出口参数：存放输入字符串存储区的第二个字节是实际输入的字符数(不含回车)，实际输入的字符串从该存储区的第三个字节处开始存放。



DATA SEGMENT

```
BUF DB 20, 20 DUP(?)
```

```
DATA ENDS
```

```
.....
```

```
LEA DX, BUF
```

```
MOV AH, 0AH
```

```
INT 21H
```

程序运行到此停下，
等待用户键盘输入

BUF

DX

实际字符个数

开始存放

14H

04H

41H

42H

43H

44H

若输入ABCD<CR>后

6) 程序结束，返回DOS(4C号功能)

功能： 将控制权移交DOS。

入口参数： 无

出口参数： 无

MOV AH, 4CH

INT 21H



在屏幕上显示What's your name?, 用户输入自己的名字
###后显示: Welcome ###。

显示字符串后回车换行

DATA SEGMENT

MEG DB 'What's your name?', 10, 13, '\$'

MEG1 DB 'Welcome \$'

BUF DB 30, ?, 30 DUP(0)

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA
MOV DS, AX
LEA DX, MEG
MOV AH, 9
INT 21H ;输出字符串
LEA DX, BUF
MOV AH, 10
INT 21H;接受姓名

LEA DX, MEG1

MOV AH, 9

INT 21H ;输出Welcome

XOR BH, BH

MOV BL, BUF+1

MOV [BX+BUF+2], '\$'

LEA DX, BUF+2

MOV AH, 9

INT 21H;输出姓名

MOV AH, 4CH

INT 21H

CODE ENDS

END START

下面程序从键盘重复接收一字符送BUFF开始的单元，直到接收到回车符0DH为止。

DATA SEGMENT

BUFF DB 128 DUP(0)

DATA ENDS

CODE SEGMENT

**ASSUME CS: CODE,
DS: DATA**

**START: MOV AX, DATA
MOV DS, AX
LEA SI, BUFF**

**LOP: MOV AH, 1
INT 21H
MOV [SI], AL**

INC SI

CMP AL, 0DH

JNE LOP

MOV AH, 4CH

INT 21H

CODE ENDS

END START

3.6.2 BIOS中断调用

- ▶ 什么是BIOS?
- ▶ BIOS即Basic Input/Output System, 指的是基本输入/输出系统。这是一组底层的基础软件程序, BIOS驻留在**系统板的只读存储器ROM**中, 计算机加电后, 可以随时调用BIOS程序。BIOS程序**独立于任何操作系统**, 因此无论该计算机是运行DOS, CP/M, 还是UNIX等任何操作系统, 用户都可以调用这些服务程序。

BIOS 键盘中断 (INT 16H AH = 0, 1, 2)

从键盘读一字符 (0) /读键盘缓冲区字符 (1) /取键盘
状态字节 (2)

例：从键盘读一字符 (AH=0)

```
mov    ah, 0
```

```
int     16h                ; al = 字符码
```

```
                ; ah = 扫描码
```

```
mov     bx, ax
```

```
call    binihex
```



BIOS 显示中断 (INT 10H AH = 1, 2, 3, 6, 7, 8, 9, A)
(控制光标/读光标位置/清屏和卷屏/字符显示)

BIOS 打印中断 (INT 17H AH = 0, 1, 2)
(打印一个字符/初始化打印机/取打印机状态字节)

BIOS 串行通讯口功能 (INT 14H AH = 0, 1, 2, 3)
(初始化串口/向串口写字符/从串口读字符/取串口状态)

