

第6章

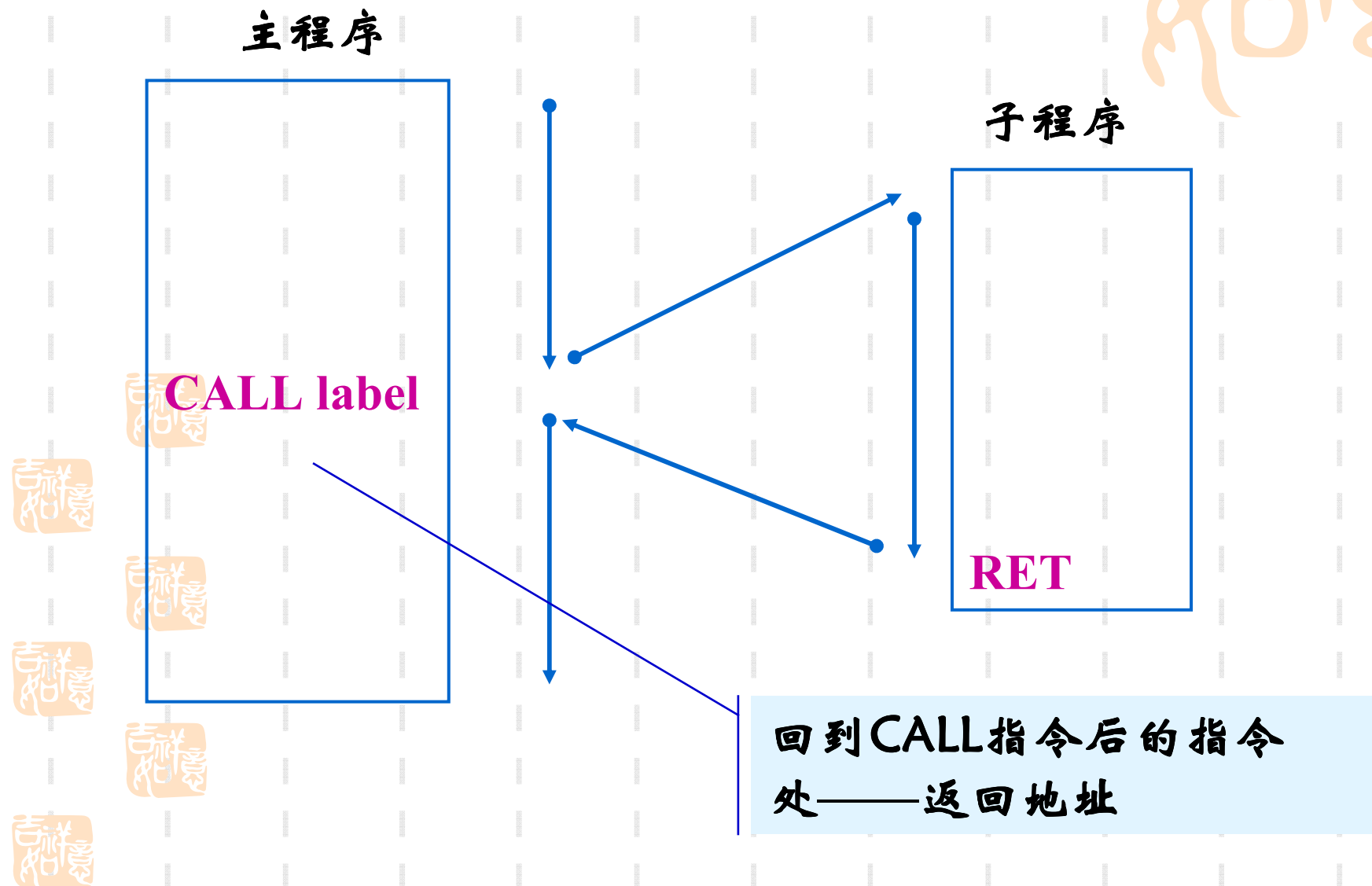
子程序结构

子程序设计

- 把功能相对独立的程序段单独编写和调试，作为一个相对独立的模块供程序使用，就形成子程序
- 子程序可以多次调用，每次调用参数不同
- 子程序可以实现源程序的模块化，可简化源程序结构，可以提高编程效率

演示

主程序与子程序



子程序设计



子程序设计要利用过程定义伪指令



参数传递是子程序设计的重点和难点



子程序可以嵌套
一定条件下，还可以递归



教学重点

子程序设计方法

过程定义伪操作

子程序的调用和返回

保护与恢复寄存器

子程序的参数传递



教学内容

一、子程序设计方法

1、过程定义伪操作

2、子程序的调用和返回

3、保护与恢复寄存器

4、子程序的参数传递

二、子程序的嵌套



1、子程序定义伪指令

过程名 **proc** [near|far]

...

过程名 **endp**

- **过程名**（子程序名）为符合语法的标识符，子程序入口的符号地址

➤ **NEAR**属性（段内调用）：调用程序和过程在同一代码段中。

➤ **FAR**属性（段间调用）：调用程序和过程不在同一代码段中。

例题 调用程序和子程序在同一代码段内

MAIN proc FAR
 :
 CALL SUBR1
 :
 ret
 endp
MAIN

SUBR1 proc NEAR
 :
 :
 RET
SUBR1 endp

可以嵌套
定义，写
为：

MAIN proc FAR
 :
 CALL SUBR1
 :
 ret

SUBR1 proc NEAR
 :
 :
 RET
SUBR1 endp
MAIN endp

例题 调用程序和子程序在不同代码段内

```
SEGX  SEGMENT
      :
SUBT   proc FAR
      :
      ret
SUBT   endp
      :
      CALL SUBT
      :
SEGX  ENDS

SEGY  SEGMENT
      :
      CALL SUBT
      :
SEGY  ENDS
```

定义为FAR属性的子
程序可以被相同或不
同代码段的程序调用

教学内容

一、子程序设计方法

1、过程定义伪操作

2、子程序的调用和返回

3、保护与恢复寄存器

4、子程序的参数传递

二、子程序的嵌套



2、主程序与子程序之间的转返

主程序转子程序，用CALL指令实现。对主程序，在什么时刻应从什么位置进入哪一个子程序，事先是清楚的，因此主程序调用子程序可以预先安排；

子程序返主程序，用RET指令实现。对子程序，每次执行完应该返回到哪个调用程序以及调用程序的什么位置，子程序是无法预先安排的。子程序的返回位置与主程序的调用位置有关。

CALL 指令

(1) .直接调用指令 CALL target

功能：将返回地址进栈后将程序控制转移到子程序。

target属于NEAR型，段内调用 CS不变，IP改变

target属于FAR型，段间调用 CS、IP都改变

段内调用的具体操作：

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (IP)$

$(IP) \leftarrow \text{OFFSET target}$

段间调用的具体操作：

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (CS)$

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (IP)$

$(IP) \leftarrow \text{OFFSET target}$

$(CS) \leftarrow \text{SEG target}$

CALL指令

(2) .间接调用指令 CALL dest

功能：将返回地址进栈后将目的操作数的内容送IP或CS或IP。

段内调用 CS不变，IP改变

段间调用 CS、IP都改变

段内间接调用的具体操作：

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (IP)$

$(IP) \leftarrow (EA)$

由dest的寻址方式确定的有效地址

段间调用的具体操作：

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (CS)$

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (IP)$

$(IP) \leftarrow (EA)$

$(CS) \leftarrow (EA+2)$

RET指令

格式 RET [N] (N为正偶数, 可缺省)

功能: 将程序控制返回到主程序。

段内返回:

$(IP) \leftarrow (SP)$

$(SP) \leftarrow (SP) + 2$

段间返回:

$(IP) \leftarrow (SP)$

$(SP) \leftarrow (SP) + 2$

$(CS) \leftarrow (SP)$

$(SP) \leftarrow (SP) + 2$

带有正偶数N的返回指令的操作, SP还要多加N, 加N的目的是废除堆栈中N/2个无用字。

用堆栈法传递参数的子程序常用带有正偶数的返回指令返回主程序。

例题 子程序调用时堆栈变化情况1/2

P101 例 3.82 主程序NAIN在一个代码段,子程序PRO_A、PRO_B、 PRO_C在另一个代码段中。它们之间的调用关系为:

```
MAIN
-----
-----
CALL FAR PTR PRO_A
(IP) =1000
-----
-----
(CS) =0500
```

```
PRO_A
-----
-----
CALL NEAR PTR PRO_B
(IP) =2500
-----
-----
CALL NEAR PTR PRO_C
(IP) =3700
-----
RET
```

例题 子程序调用时堆栈变化情况2/2

PRO_B

CALL NEAR PTR PRO_C

(IP) =4000

RET

PRO_C

RET

教学内容

一、子程序设计方法

- 1、过程定义伪操作
- 2、子程序的调用和返回
- 3、保护与恢复寄存器
- 4、子程序的参数传递

二、子程序的嵌套



3、保存与恢复寄存器

保护现场——在使用某些不能被破坏的寄存器之前，将其内容保存起来。

恢复现场——使用之后将其还原。

方法：在转子程序之前，或在子程序的开始，将子程序要用到的寄存器内容用PUSH指令压栈保存；在返回主程序之前，或返回主程序之后立即用POP指令恢复现场。

子程序的常见格式

subname proc ;具有缺省属性的subname过程

push ax ;保护寄存器: 顺序压入堆栈

push bx ;ax/bx/cx仅是示例

push cx

... ;过程体

pop cx ;恢复寄存器: 逆序弹出堆栈

pop bx

pop ax

ret ;过程返回

subname endp ;过程结束

例题 实现光标回车换行子程序

```
dpcrlf    proc                ;过程开始
          push ax             ;保护寄存器AX和DX
          push dx
          mov dl, 0dh         ;显示回车
          mov ah, 2
          int 21h
          mov dl, 0ah         ;显示换行
          mov ah, 2
          int 21h
          pop dx              ;恢复寄存器DX和AX
          pop ax
          ret                  ;子程序返回
dpcrlf    endp                ;过程结束
```

例题 具有多个出口的子程序

HTOASC proc

; 将AL低4位表达的一位16进制数转换为ASCII码

and al, 0fh

cmp al, 9

jbe htoasc1

add al, 37h

; 是0AH~0FH, 加37H

ret

; 子程序返回

htoasc1: add al, 30h

; 是0~9, 加30H

ret

; 子程序返回

HTOASC endp

教学内容

一、子程序设计方法

- 1、过程定义伪操作
- 2、子程序的调用和返回
- 3、保护与恢复寄存器
- 4、子程序的参数传递

二、子程序的嵌套



4、子程序的参数传递

- 主程序和子程序之间的信息传递
- **入口参数**（输入参数）：主程序提供给子程序
- **出口参数**（输出参数）：子程序返回给主程序
- 参数的形式：

① 数据本身（传值）

② 数据的地址（传址）

- 传递的方法：

(1) 寄存器

(2) 变量

(3) 堆栈

(4) 地址表

(5) 参数赋值

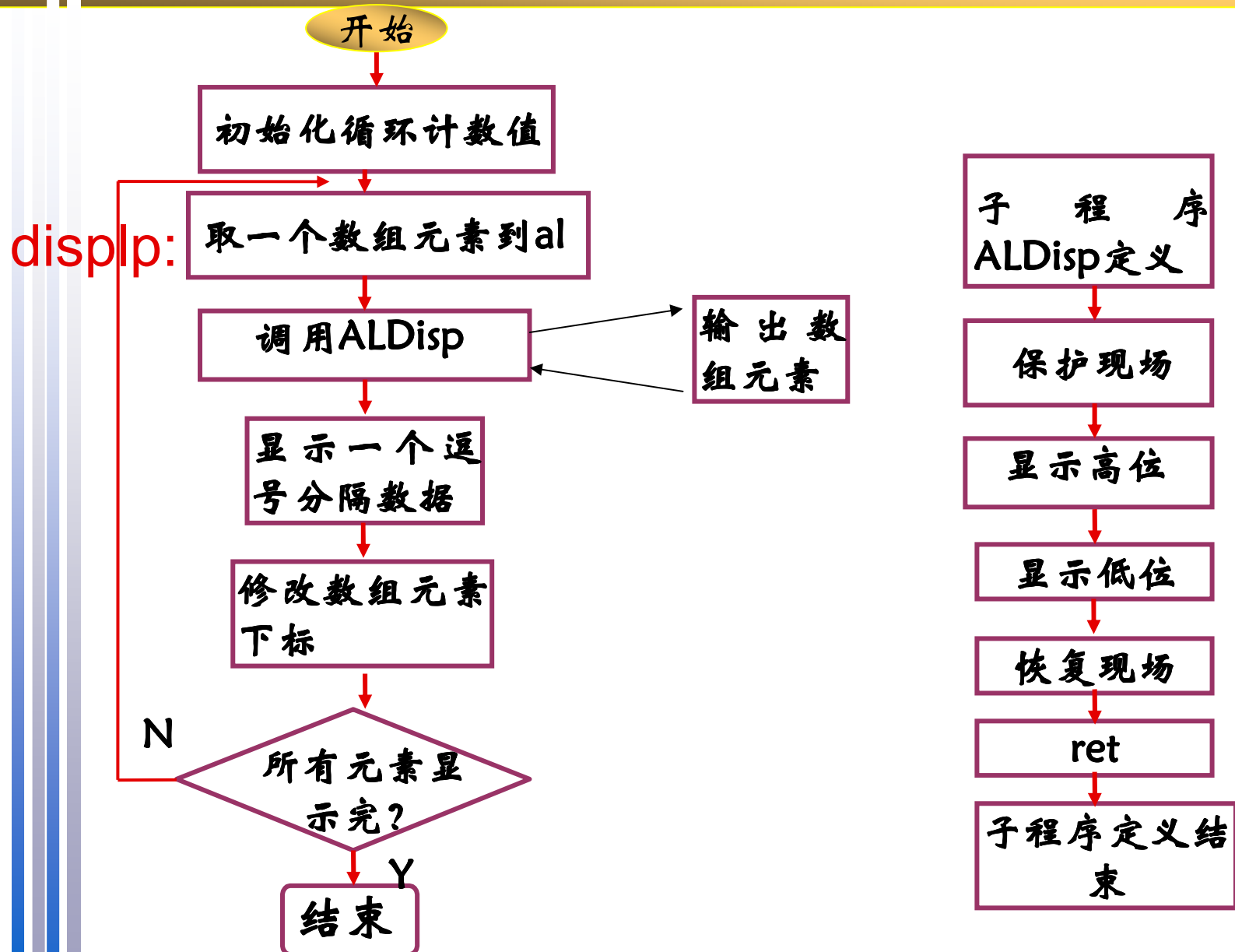
(6) 多个模块之间的参

数传递

(1) 寄存器传递

- 寄存器：主程序和子程序间传递的参数都在约定的寄存器中。
- 在调用子程序前，主程序将入口参数送到约定的寄存器中，子程序直接从这些寄存器中取得参数进行运算处理；
- 处理后得到的结果也放在约定的寄存器中，返回主程序后，主程序就从该寄存器中得到结果。

例 将字节数组ARRAY中的内容显示出来子程序



例 将字节数组ARRAY中的内容显示出来子程序1/3

ALdisp	proc	;实现al内容的显示
	push ax	;过程中使用了AX、CX和DX
	push cx	
	push dx	
	push ax	;暂存ax
	mov dl, al	;转换al的高4位
	mov cl, 4	
	shr dl, cl	
	or dl, 30h	;al高4位变成3
	cmp dl, 39h	
	jbe aldisp1	
	add dl, 7	;是0Ah~0Fh, 还要加上7
aldisp1:	mov ah, 2	;显示
	int 21h	



将字节数组ARRAY中的内容显示出来子程序2/3

```
        pop dx                ;恢复原ax值到dx
        and dl, 0fh           ;转换al的低4位
        or dl, 30h
        cmp dl, 39h
        jbe aldisp2
        add dl, 7
aldisp2: mov ah, 2             ;显示
        int 21h
        pop dx
        pop cx
        pop ax
        ret                   ;过程返回
ALdisp  endp
```

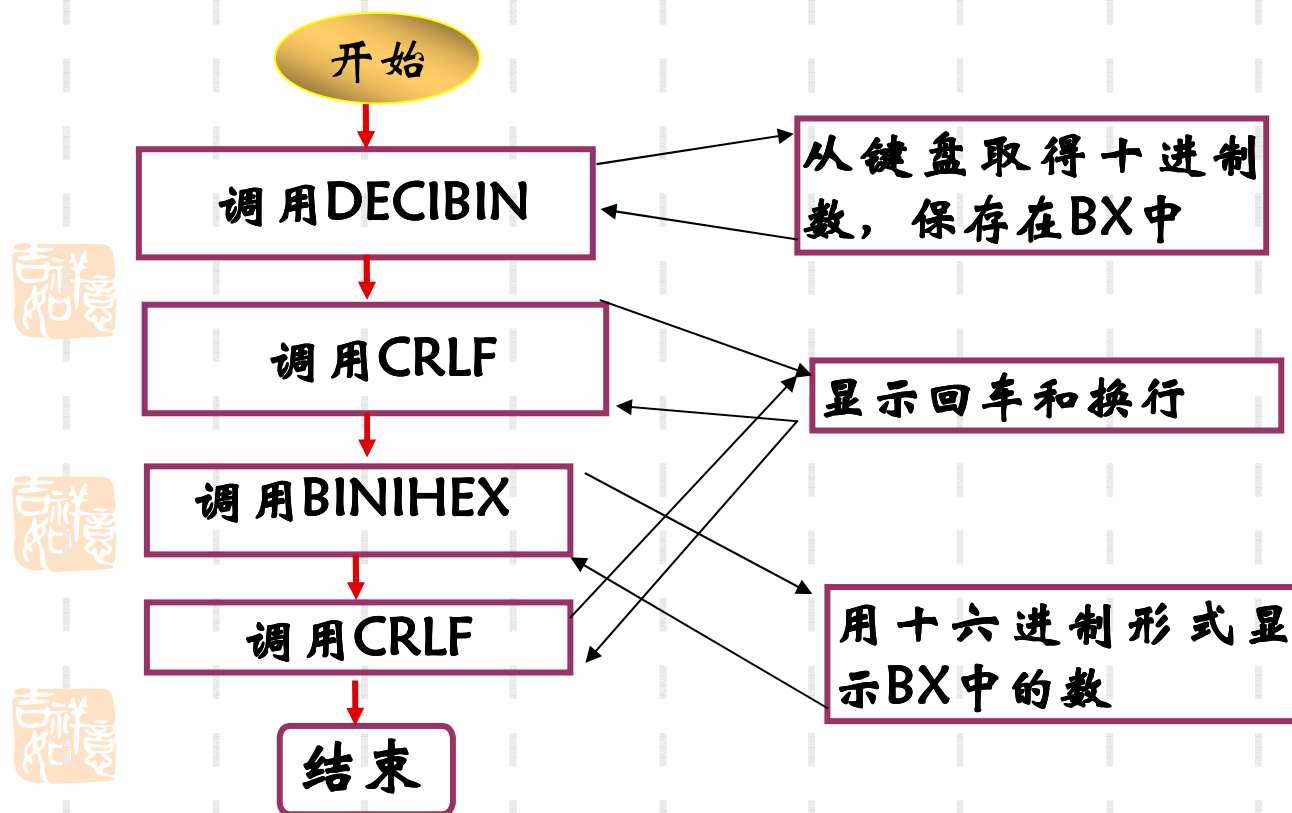
将字节数组ARRAY中的内容显示出来主程序 - 3/3

```
... ;  
mov bx, offset array;调用程序段开始  
mov cx, count  
dislp: mov al, [bx]  
       call ALdisp ;调用显示过程  
       mov dl, ',' ;显示一个逗号, 分隔数据  
       mov ah, 2  
       int 21h  
       inc bx  
       loop dislp ;调用程序段结束  
       .exit 0  
... ;过程定义  
end
```



P199 例6.3 十进制到十六进制数转换程序

程序要求从键盘取得一个十进制数，然后把该数以十六进制形式在屏幕上显示出来。



例 求校验和

- 子程序计算数组元素的“校验和”
- 校验和是指不记进位的累加

P202 例6.4主程序MAIN和过程PROADD在同一源文件中，要求用过程PROADD累加数组中的所有元素，并把校验和送到指定的存储单元中去。

入口参数:

数组的逻辑地址（传址）

元素个数（传值）

出口参数:

求和结果（传值）

data segment

array dw 100dup(?)

count dw 100

result dw ?

data ends





例6.4a

入口参数: CX = 元素个数,

DS:BX = 数组的段地址: 偏移地址

出口参数: AX = 校验和

用寄存器传递参数



例6.4a 主程序

.startup

;设置入口参数 (含有DS←数组的段地址)

mov bx, offset array; BX←数组的偏移地址

mov cx, count; CX←数组的元素个数

call checksuma; 调用求和过程

mov result, ax; 处理出口参数

.exit 0



例6.4a 子程序

```
checksuma    proc
               xor ax, ax           ;累加器清0
suma:         add ax, [bx]          ;求和
               inc bx
               inc bx               ;指向下一个字
               loop suma
               ret
checksuma     endp
end
```





用寄存器传递参数特点

- 把参数存于约定的寄存器中，可以传值，也可以传地址。



- 子程序对带有出口参数的寄存器不能保护和恢复（主程序视具体情况情况进行保护）



- 子程序对带有入口参数的寄存器可以保护，也可以不保护



4、子程序的参数传递



■ 传递的方法:

(1) 寄存器

(2) 变量

(3) 堆栈

(4) 地址表

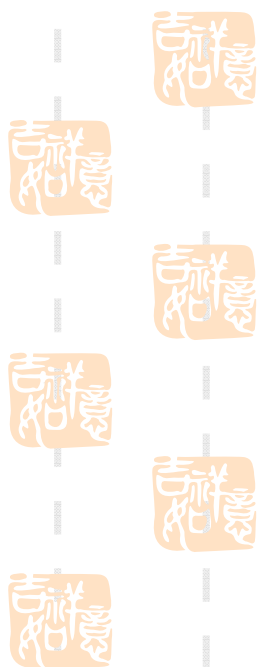
(5) 参数赋值

(6) 多个模块之间的参数传递



(2) 变量传递

- **变量**：在调用程序中定义变量，如果子程序和调用程序在同一程序模块中，则子程序可直接按名访问模块中的变量。





例6.4b

入口参数:

count = 元素个数,

array = 数组名 (含段地址: 偏移地址)

出口参数: result = 校验和

用变量传递参数

```
data    segment
        array    dw 100dup(?)
        count    dw 100
        result    dw ?
data    ends
```

例6. 4b 主程序

```
code segment
main proc far
    assume cs:code,ds:data

start: push ds
        sub ax,ax
        mov ax,data
        mov ds,ax
        .....
        call checksumb
        .....
        ret
main endp
```

例 6. 4b - 1/2

checksumb proc

push ax

push bx

push cx

xor ax, ax ;累加器清0

mov bx, offset array

;BX←数组的偏移地址

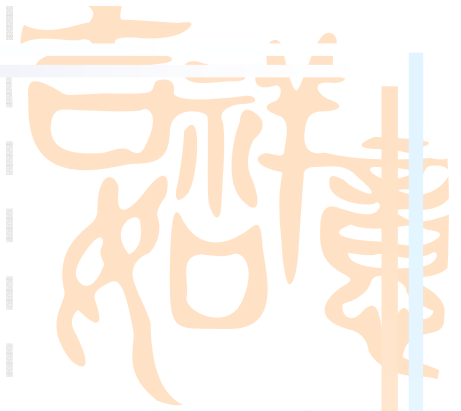
mov cx, count

;CX←数组的元素个数



例 6. 4b - 2/2

```
sumb:      add ax, [bx]      ; 求和
           inc bx
           inc bx
           loop sumb
           mov result, ax    ; 保存校验和
           pop cx
           pop bx
           pop ax
           ret
checksumb   endp
Code       ends
end start
```





用变量传递参数特点

- 主程序和子程序直接采用同一个变量名共享同一个变量，实现参数的传递
- 方便，通用性差
- 不同模块间共享时，需要声明



4、子程序的参数传递



■ 传递的方法:

(1) 寄存器

(2) 变量

(3) 堆栈

(4) 地址表

(5) 参数赋值

(6) 多个模块之间的参数传递



(3) 堆栈传递

堆栈：把主程序与子程序间传递的参数都放到堆栈中。

- 在调用子程序前，入口参数由主程序送到堆栈中，子程序从堆栈中取得这些参数，并将处理结果送到堆栈中。
- 返回主程序后，主程序从堆栈取得结果。



例6.4c

入口参数:

顺序压入偏移地址和元素个数

出口参数: Ax = 校验和

用堆栈传递参数

data segment

array dw 100dup(?)

count dw 100

result dw ?

data ends

例6.4c 主程序

```
.startup
```

```
mov ax, offset array
```

```
push ax
```

```
mov ax, count
```

```
push ax
```

```
call checksumc
```

```
add sp, 4
```

图示

堆栈段

SP

IP

要注意堆栈的分配情况，保证参数存取正确、子程序正确返回，并保持堆栈平衡

主程序实现平衡堆栈： add sp,n

子程序实现平衡堆栈： ret n

checksumc

sumc:

checksumc

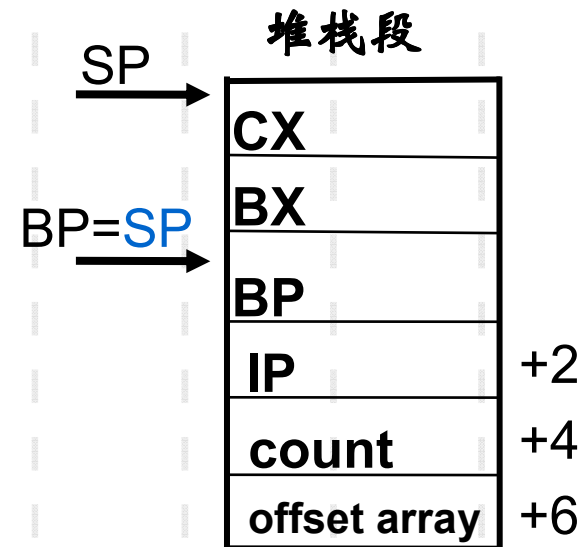
```
proc
push bp
mov bp, sp
push bx
push cx
mov bx, [bp+6]
mov cx, [bp+4]
xor ax, ax
add ax, [bx]
inc bx
inc bx
loop sumc
pop cx
pop bx
pop bp
ret
endp
```

例6.4c 子程序

;利用BP间接寻址存取参数







;SS:[BP+6]指向偏移地址

;SS:[BP+6]指向元素个数





用堆栈传递参数时，典型的过程结构如下：

```
stdProc proc near
    push bp
     mov bp,sp
    .....
     pop bp
     ret Parmsize
     stdProc endp
    
    
```


4、子程序的参数传递

■ 传递的方法：

(1) 寄存器

(2) 变量

(3) 堆栈

(4) 地址表

(5) 参数赋值

(6) 多个模块之间的参数传递

(4) 地址表传递

- **地址表**：在主程序中建立一个地址表，把要传递给子程序的参数都存放在地址表中
- 然后把地址表的首地址通过寄存器BX传送到子程序中
- 子程序通过地址表取得所需参数，并把结果存入指定的存储单元中去



用地址表传递参数

- 把要传送给子程序的参数都存放在地址表中，然后把地址表的首地址通过寄存器BX传送到子程序中。
- 子程序通过地址表取得所需参数，并把结果存入指定的存储单元中去。

例6.4d

入口参数：

地址表首地址

出口参数：

Ax = 校验和

例6.4d 主程序

```
.startup  
mov table,offset array  
mov table+2,offset count  
mov table+4,offset result  
mov bx,offset table  
call checksumc  
  
.exit 0
```



例6.4d 子程序

```
checksumc    proc
              push ax    (cx, si, di)
              mov si, [bx]
              mov di, [bx+2]
              mov cx, [di]
              mov di, [bx+4]
              xor ax, ax
sumc:         add ax, [si]
              inc si
              loop sumc
              mov [di], ax
              pop di
              pop si
              pop cx
              pop ax
              ret
checksumc     endn
```

```
data segment
    array    dw 100dup(?)
    count    dw 100
    result    dw ?
    table     dw 3dup(?)
data ends
```



4、子程序的参数传递



■ 传递的方法:

(1) 寄存器

(2) 变量

(3) 堆栈

(4) 地址表

(5) 参数赋值

(6) 多个模块之间的参数传递



(5) 地址表传递

- **参数赋值**：将参数存放到CALL指令后的一串单元中
- 子程序通过返回地址存取参数并修改返回地址。



■ 例 统计字符串长度

设计一个子程序strlen，由AX返回字符串长度。设指定的字符串以0终止法表示。

调用形式为：

CALL strlen

**db “This parameter is in the code
stream”, 0**

例子程序 strlen

```
strlen    proc    near
           push    bp
           mov     bp, sp
           push    bx
           mov     ax, 0
           mov     bx, 2[bp]
next:      cmp     byte ptr cs:[bx], 0
           jz      Endstr
           inc     ax
           inc     bx
           jmp     next
Endstr:    inc     bx
           mov     2[bp], bx; 使BX指向字符串结束标
           志0的下一个字节, 修改返回地址
           pop     bx
           pop     bp
           ret
strlen    endp
```



例 求数组最大值

- 设计一个子程序Findmax，求带符号数组array的最大值，元素个数由count定义，结果存入max中。

数据段及调用语句为：

```
dseg segment
```

```
count dw 5
```

```
array dw 8, -1, 32766, 0, 100
```

```
Max dw ?
```

```
dseg ends
```

主程序

```
call far ptr Findmax
```

```
dd count, array, max
```

```
.....
```

例子程序Findmax

```
Findmax    proc    near
MINSW=8000H    ;16位带符号数的最小值
            push    bp
            mov     bp, sp
            push    ax    (bx, cx, si, ds, es)
            lds     bx, 2[bp]    ;装入返回地址到DS:BX
            les     si, [bx]    ;取元素个数所在地址到ES:SI
            mov     cx, es:[si];取元素个数
            les     si, 4[bx]    ;取数组首地址
            mov     ax, MINSW
next:       cmp     ax, es:[si]
            jg      skip
            mov     ax, es:[si]
skip:       add     si, 2
            loop    next
            les     si, 8[bx]; 取得存放最大值的地址
            mov     es:[si], ax
            add     bx, 12;      代码流有12个字节
            mov     2[bp], bx
            pop     es    (ds, si, cx, bx, ax, bp)
            ret
Findmax    endp
```



4、子程序的参数传递



■ 传递的方法:

(1) 寄存器

(2) 变量

(3) 堆栈

(4) 地址表

(5) 参数赋值

(6) 多个模块之间的参数传递



(6) 多个模块之间的参数传递问题

- 当调用程序与子程序不在同一个程序模块时，参数传递与外部符号有关。
- **局部符号**：在一个模块中定义，又在同一个模块中引用的符号。
- **外部符号**：在某一个模块中定义，而又在另一个模块中引用的符号。
- 与外部符号相关的两个伪操作是**PUBLIC**和**EXTRN**，两个伪操作必须匹配。

PUBLIC和EXTRN

- PUBLIC格式:

PUBLIC 符号[, ...]

在一个模块中定义的符号（包括变量、标号、过程名等）在提供给其他模块使用时，必须用

PUBLIC将其定义为外部符号。

变量: byte, word, dword
标号、过程名: near, far

- EXTRN格式:

EXTRN 符号名: 类型[, ...]

在另一个模块中定义而要在本模块中使用的符号必须使用**EXTRN**伪操作。

P209例6.5 – 模块1

```
;          source module1
;
extern     var2:word, lab2:far
public     var1, lab1
;
data1     segment
           var1  db  ?
           var3  dw  ?
           var4  dw  ?

data1     ends

;
code1     segment
           assume cs:code1, ds:data1
           .....

lab1:
           .....

code1     ends
end
```



P209例6.5 - 模块2

```
;          source module2
;
extern     var1:byte, var4:word
public    var2
;
data2      segment
            var2    dw    0
            var3    db    5dup(?)
data2      ends
;
code2      segment
            assume   cs:code2, ds:data2
            .....
code2      ends
            end
```



P209例6.5 - 模块3

```
;          source module3
;
extern     lab1:far
public    lab2, lab3
;
code3     segment
          assume  cs:code3
          .....
lab2:
          .....
lab3:
          .....
code3:    ends
;
          end
```



多个模块之间的参数传递方法

- 使用公共数据段——把data段用common合并成为一个覆盖段
- 把变量定义为外部符号——允许其他模块引用在某一模块中定义的变量名。注意段寄存器的使用。



使用公共数据段—— ——模块1 (1/2)

```
;          source module1
public     proadd
data       segment common
            ary      dw  100dup(?)
            count    dw  100
            sum      dw  ?

data       ends
code1      segment
proadd     proc far
            assume   cs:code1, ds:data
            mov      ax, data
            mov      ds, ax
            push     ax (cx, si)
            lea      si, ary
            mov      cx, count
            xor      ax, ax
```

使用公共数据段—— 模块2 (2/2)

```
next:      add  ax, [si]
           add  si, 2
           loop next
           mov  sum, ax
           pop  si
           pop  cx
           pop  ax
           ret
proadd     endp

code1:     ends
end
```



使用公共数据 段——模块2

```

;          source module2

extern    proadd:far

data      segment common
            ary    dw  100dup(?)
            count  dw  100
            sum    dw  ?

data      ends

code2     segment
            assume  cs:code2, ds:data
            .....

            call   far ptr proadd

.....

code2:    ends
            end
```

Common把不同模块中的同名段重叠而形成一个段，由于各同名段有相同的起始地址，所以会产生覆盖。

Data段用common合并成一个覆盖段，模块1只引用了本模块中的变量



把变量定义为外部 符号——模块1 (1/2)

把变量定义为外部符号

```
source module1
extern    var1:word, output:far
extern    var2:word
public    exit
local_data segment
            var dw 5
            .....
local_data ends
code segment
    assume cs:code, ds:local_data
main proc far
start:
    mov    ax, local_data
    mov    ds, ax
    .....

```

```
var: module1
var1:module2
var2:module3
var1+var→var
var2-50 →var2

```

把变量定义为外部符号——模块1 (2/2)

```
mov    bx, var
mov    ax, seg var1
mov    es, ax
add    bx, es:var1
.....
mov    ax, seg var2
mov    es, ax
sub    es:var2, 50
.....
jmp    output
.....
exit:  mov    ax, 4c00h
      int    21h
main  endp
code  ends
      end    start
```



```
source module2  
public var1  
extdata1 segment  
    var1 dw 10  
    .....  
extdata1 ends  
    .....  
end
```

把变量定义为外部
符号——模块2



把变量定义为外部 符号——模块3

```
source module3

public  var2

extrn   exit:far
extdata2 segment
        var2 dw 3
        .....
extdata2 ends
public output
program segment
        assume cs:program, ds:extdata2
        .....
output: jmp  exit
        .....
program ends
end
```



教学内容

一、子程序设计方法

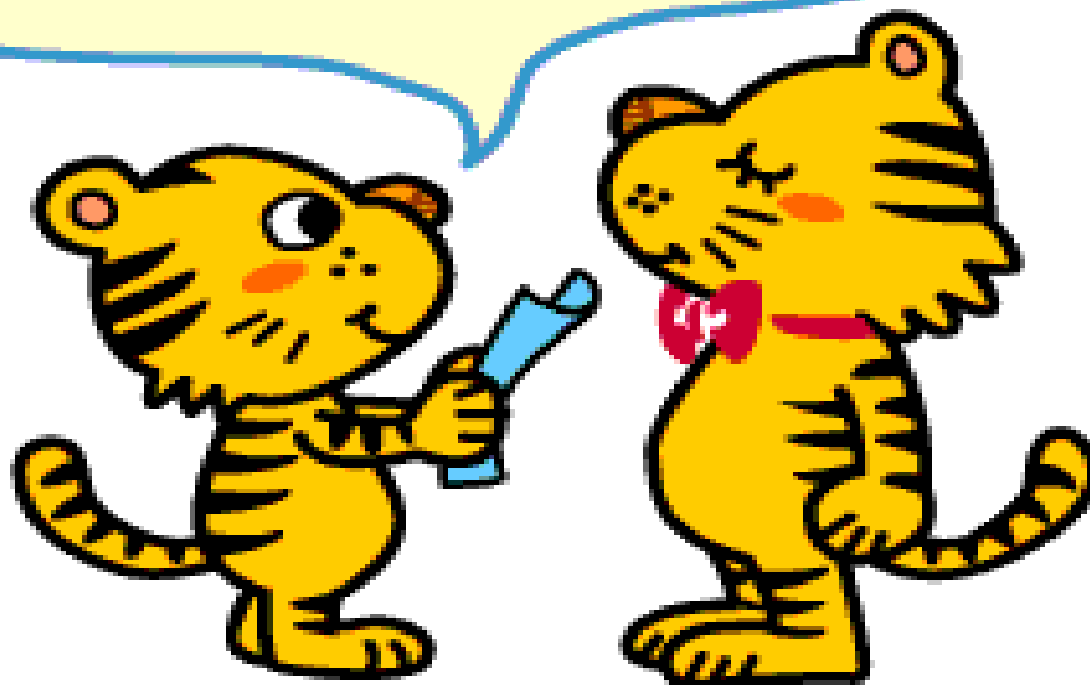
- 1、过程定义伪操作
- 2、子程序的调用和返回
- 3、保护与恢复寄存器
- 4、子程序的参数传递

二、子程序的嵌套



子程序的嵌套

子程序内包含有子程序
的调用就是子程序嵌套
没有什么特殊要求



例6.6嵌套子程序 - 1/4

..... 主程序

mov bx, offset array; 调用程序段开始

mov cx, count

displp: mov al, [bx]

call ALdisp ; 调用显示过程

mov dl, ',' ; 显示一个逗号, 分隔数据

mov ah, 2

int 21h

inc bx

loop displp ; 调用程序段结束

.exit 0

... ; 过程定义

end



例6.6 嵌套子程序 - 2/4

ALdisp	proc	
	push ax	
	push cx	; 实现al内容的显示
	push ax	; 暂存ax
	mov cl, 4	
	shr al, cl	; 转换al的高4位
	call htoasc	; 子程序调用 (嵌套)
	pop ax	; 转换al的低4位
	call htoasc	; 子程序调用 (嵌套)
	pop cx	
	pop ax	
	ret	
ALdisp	endp	



例6.6 嵌套子程序 - 3/4

; 将AL低4位表达的一位16进制数转换为ASCII码

```
HTOASC  proc  
        push ax  
        push bx  
        push dx  
        mov bx, offset ASCII; BX指向ASCII码表  
        and al, 0fh        ; 取得一位16进制数  
        xlat CS:ASCII
```

; 换码: $AL \leftarrow CS:[BX + AL]$, 注意数据在代码段CS

XLAT OPR

OPR是表格的首地址, 只是为了提高程序的可读性而设置, 指令执行时只使用预先存入BX中的表格首地址, 而并不采用汇编格式中指定的值

例6.6 嵌套子程序 - 4/4

```
mov dl, al      ;显示  
mov ah, 2  
int 21h  
pop dx  
pop bx
```

☼ 这是一个具有局部变量的子程序。因为数据区与子程序都在代码段，所以利用了换码指令XLAT的另一种助记格式。

☼ 除采用段超越方法外，子程序与主程序的数据段不同时，我们还可以通过修改DS值实现数据存取；但需要保护和恢复DS寄存器



子程序的递归

- 当子程序直接或间接地嵌套调用自身时称为递归调用，含有递归调用的子程序称为递归子程序
- 递归子程序必须采用寄存器或堆栈传递参数，递归深度受堆栈空间的限制

例6.7：求阶乘

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

例6.7 主程序 - 1/3

```
N      .model small
result .stack 256
      .data
      dw 3
      dw ?
      .code
      .startup
      mov bx,N
      push bx           ;入口参数: N
      call fact         ;调用递归子程序
      pop result        ;出口参数: N!
      .exit 0
```

例6.7 递归子程序 - 2/3

; 计算N! 的近过程

; 入口参数: 压入 N

; 出口参数: 弹出 N!

```
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
```

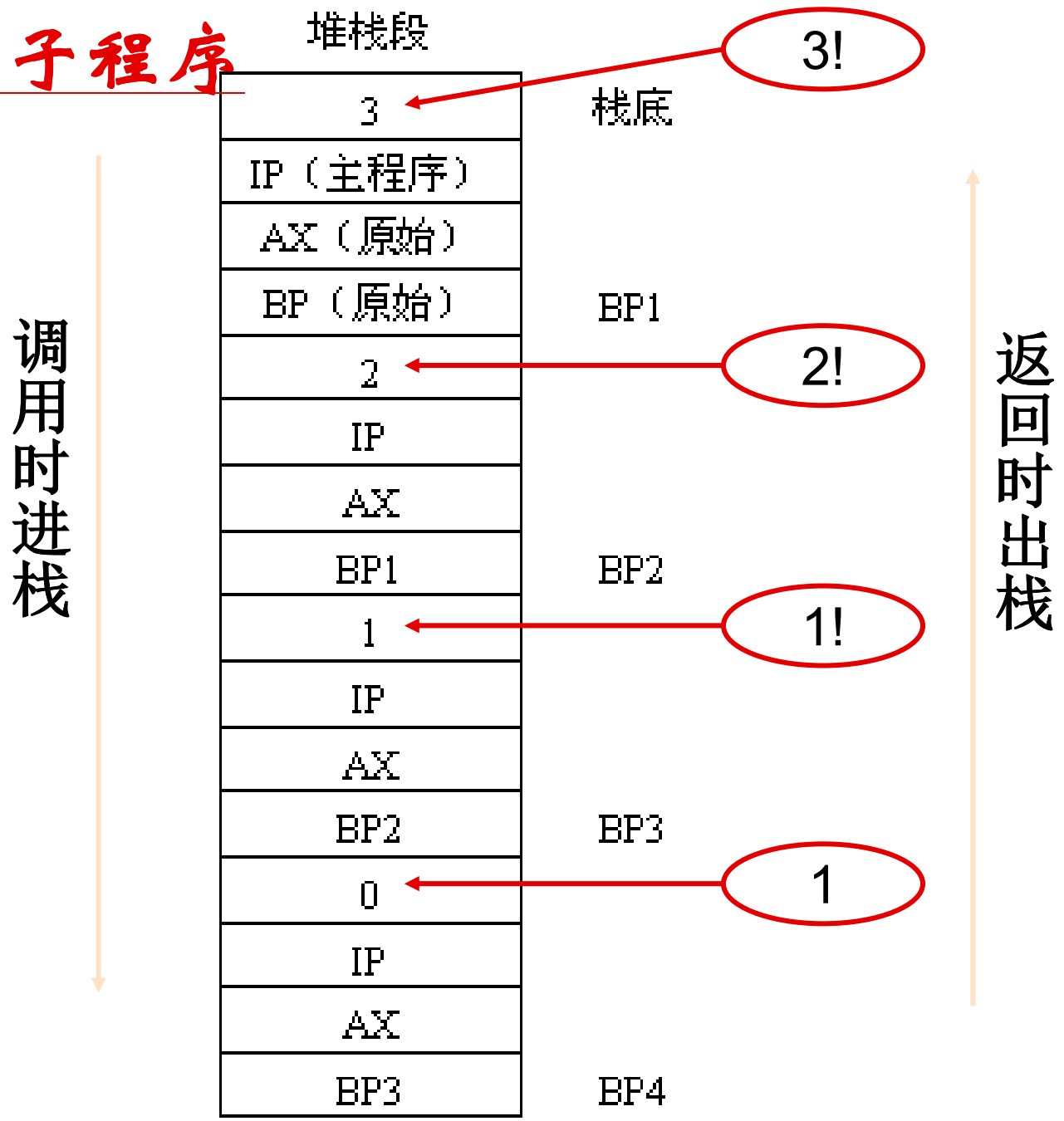


例6.7 递归子程序 - 3/3

```
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  $N \times (N-1)!$ 
fact2:    mov [bp+6], ax  ;存入出口参数 N!
          pop bp
          pop ax
          ret
fact      endp
```



递归子程序



子程序举例

- P225-----例 6.9
- P231-----例 6.11



第6章 教学要求

1. 掌握子程序及其汇编语言程序设计
 2. 熟悉主程序与子程序之间的参数传递问题
- 作业：P242 6.4 6.5 6.7



放鬆一下

