



第六章 子程序结构





主要内容:

- 过程定义伪操作
- 子程序的调用与返回
- 保存与恢复寄存器
- 子程序的参数传送
- 子程序的嵌套与递归



1. 过程定义伪操作

```
过程名  PROC  NEAR ( FAR )  
:  
:  
过程名  ENDP
```

- (1) **NEAR**属性: 调用程序和子程序在同一代码段中
(段内调用)
- (2) **FAR**属性: 调用程序和子程序不在同一代码段中
(段间调用)



code segment

```
main proc far
.....
    call subr1
.....
    ret
main endp

subr1 proc near
.....
    ret
subr1 endp

code ends
```

段内调用

```
segx segment
subt proc far
.....
    ret
subt endp
.....
    call subt
.....
segx ends

segx segment
.....
    call subt
.....
segx ends
```

段间调用

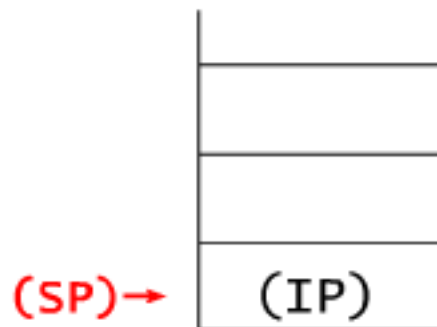


2. 子程序的调用与返回

子程序调用：隐含使用堆栈保存返回地址

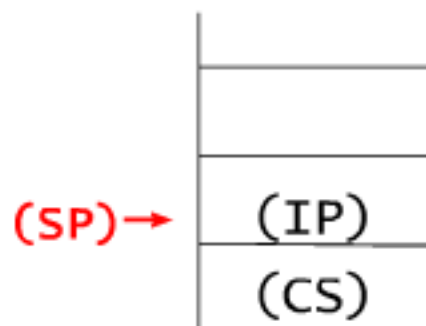
`call near ptr subp`

- (1) 保存返回地址
- (2) 转子程序



`call far ptr subp`

- (1) 保存返回地址
- (2) 转子程序



子程序返回： `ret`



3. 保存与恢复寄存器

```
subt    proc    far
        push    ax
        push    bx
        push    cx
        push    dx
        .....
        .....
        pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret
subt    endp
```



4. 子程序的参数传送

- (1) 通过寄存器传送参数
- (2) 通过存储器传送参数
- (3) 通过地址表传送参数地址
- (4) 通过堆栈传送参数或参数地址
- (5) 多个模块之间的参数传送



例：十进制到十六进制的转换（通过寄存器传送参数）

```
decihex segment ; 10→16
assume cs: decihex
main proc far
    push ds
    sub ax, ax
    push ax
repeat: call decibin ; 10→2
        call crlf ; 回车换行
        call binihex ; 2→16
        call crlf
        jmp repeat
        ret
main endp
.....
.....
..... ; 三个子程序
decihex ends
end main
```




```
decibin proc near
mov bx, 0
newchar: mov ah, 1
int 21h
sub al, 30h
jl exit
cmp al, 9
jg exit
cbw
xchg ax, bx
mov cx, 10
mul cx
xchg ax, bx
add bx, ax
jmp newchar
exit: ret
decibin endp
```

```
binihex proc near
rotate: mov ch, 4
mov cl, 4
rol bx, cl
mov al, bl
and al, 0fh
add al, 30h
cmp al, 3ah
jl printit
add al, 7
printit: mov dl, al
mov ah, 2
int 21h
dec ch
jnz rotate
ret
binihex endp
```

```
crlf proc near
mov dl, 0dh
mov ah, 2
int 21h
mov dl, 0ah
mov ah, 2
int 21h
ret
crlf endp
```



例：十六进制到十进制的转换（通过寄存器传送参数）

```
hexidec segment ; 16→10
main      proc far
start:    push ds
          sub  ax, ax
          push ax

repeat:   call hexibin ; 16→2
          call crlf
          call binidec ; 2→10
          call crlf
          jmp  repeat
          ret
main      endp
.....
.....
.....
hexidec   ends
          end    start
```



```
hexibin proc near
newchar:
    mov     ah, 1
    int     21h
    sub     al, 30h
    jl      exit
    cmp     al, 10
    jl      add_to
    sub     al, 27h
    cmp     al, 0ah
    jl      exit
    cmp     al, 10h
    jge     exit

add_to:
    mov     cl, 4
    shl     bx, cl
    mov     ah, 0
    add     bx, ax
    jmp     newchar

exit:     ret
hexibin endp
```

```
binidec proc near
    mov     cx, 10000d
    call    dec_div
    mov     cx, 1000d
    call    dec_div
    mov     cx, 100d
    call    dec_div
    mov     cx, 10d
    call    dec_div
    mov     cx, 1d
    call    dec_div
    ret
```

binidec endp

```
dec_div proc near
    mov     ax, bx
    mov     dx, 0
    div     cx
    mov     bx, dx
    mov     dl, al
    add     dl, 30h
    mov     ah, 2
    int     21h
    ret
dec_div endp
```





例：累加数组中的元素（通过存储器传送参数）

```
data    segment
        ary      dw  1,2,3,4,5,6,7,8,9,10
        count    dw  10
        sum      dw  ?
data    ends

code    segment
main    proc      far
        assume    cs:code,ds:data
        push      ds
        sub       ax, ax
        push      ax
        mov       ax, data
        mov       ds, ax
        call      proadd
        ret
main    endp

.....
code    ends
        end       main
```

;proadd子程序

```
proadd  proc  near
        push  ax
        push  cx
        push  si
next:   lea    si, ary
        mov   cx, count
        xor   ax, ax
        add   ax, [si]
        add   si, 2
        loop  next
        mov   sum, ax
        pop   si
        pop   cx
        pop   ax
        ret
proadd  endp
```



如果数据段定义如下：

```
data    segment

        ary      dw      1,2,3,4,5,6,7,8,9,10
        count    dw      10
        sum       dw      ?

        ary1     dw      10,20,30,40,50,60,70,80,90,100
        count1   dw      10
        sum1      dw      ?

data    ends
```

如果直接访问内存变量，那么累加**数组ary**
和**数组ary1**中的元素不能用同一个子程序 proadd。



例：累加数组中的元素（通过地址表传送参数地址）

```
data segment
    ary      dw  1,2,3,4,5,6,7,8,9,10
    count    dw  10
    sum       dw  ?
    table     dw  3   dup (?)           ; 地址表
data ends
code segment
main proc far
    assume cs:code, ds:data
    push     ds
    sub      ax, ax
    push     ax
    mov      ax, data
    mov      ds, ax
    mov      table, offset ary
    mov      table+2, offset count
    mov      table+4, offset sum      ; 建立地址表
    mov      bx, offset table        ; 地址表的地址->bx
    call     proadd
    ret
main endp
```



proadd proc near

push ax

push cx

push si

push di

mov si, [bx]

mov di, [bx+2]

mov cx, [di]

mov di, [bx+4]

xor ax, ax

next:

add ax, [si]

add si, 2

loop next

mov [di], ax

pop di

pop si

pop cx

pop ax

ret

proadd endp

code ends

end main

ary→ 1d 0000

2d

3d

4d

5d

6d

7d

8d

9d

10d

count→ 10d 0014

sum→ 55d 0016

table→ 0000 0018 →(bx)

0014

0016



通过**堆栈**传送参数或参数地址:

```
stack segment
    dw 100 dup (?)
    tos label word
stack ends
start:
    mov ax, stack
    mov ss, ax
    mov sp, offset tos
    push ds
    sub ax, ax
    push ax
    .....
```



```
stack segment      stack
    dw 100 dup (?)
    tos label word
stack ends
```





例：累加数组中的元素（通过堆栈传送参数地址）

```
data    segment
        ary      dw  1,2,3,4,5,6,7,8,9,10
        count    dw  10
        sum      dw  ?
data    ends
```

```
stack   segment
        dw  100      dup (?)
        tos  label  word
stack   ends
```

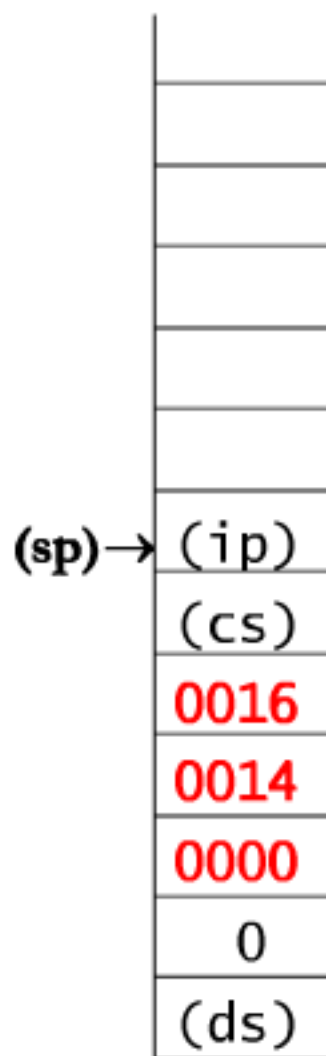


```
code1 segment
main  proc      far
      assume    cs:code1, ds:data, ss:stack
start:
```

```
      mov       ax, stack
      mov       ss, ax
      mov       sp, offset tos
      push     ds
      sub      ax, ax
      push     ax
      mov      ax, data
      mov      ds, ax
      mov      bx, offset ary
      push     bx
      mov      bx, offset count
      push     bx
      mov      bx, offset sum
      push     bx

      call     far ptr proadd

      ret
main  endp
code1 ends
```





```
code2    segment
        assume cs:code2
```

```
proadd   proc    far
```

```
    push    bp
    mov     bp, sp
```

```
    push    ax
    push    cx
    push    si
    push    di
```

```
    mov     si, [bp+0ah]
    mov     di, [bp+8]
    mov     cx, [di]
    mov     di, [bp+6]
```

```
        xor     ax, ax
next:    add     ax,  [si]
        add     si,  2
        loop    next
        mov     [di], ax
```

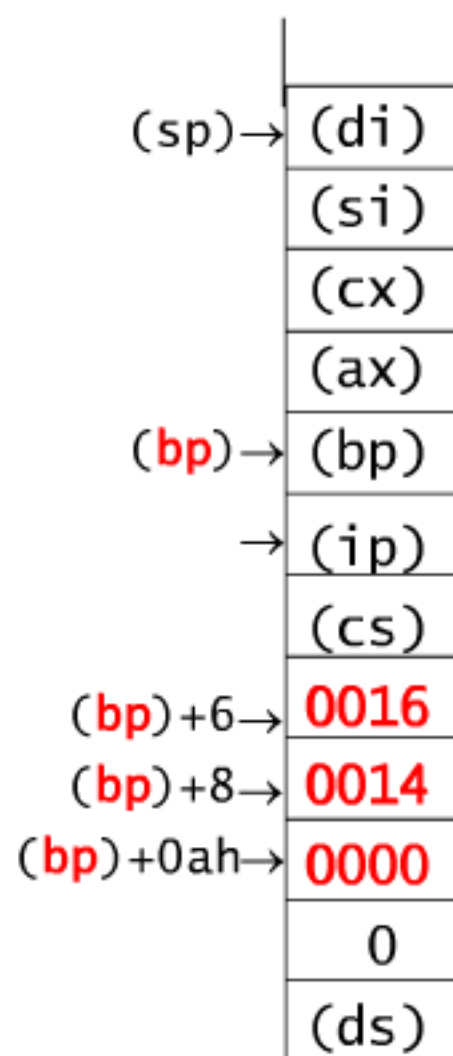
```
        pop     di
        pop     si
        pop     cx
        pop     ax
```

```
        pop     bp
```

```
        ret     6
```

```
proadd   endp
```

```
code2    ends
end start
```





结构伪操作 **STRUC**:

定义一种可包含不同类型数据的结构模式

格式: 结构名 **STRUC**
 字段名1 **DB** ?
 字段名2 **DW** ?
 字段名3 **DD** ?

 结构名 **ENDS**

例: 学生个人信息

```
STUDENT_DATA    STRUC  
                        NAME    DB    5    DUP (?)  
                        ID    DW    0  
                        AGE    DB    ?  
                        DEP    DB    10    DUP (?)  
STUDENT_DATA    ENDS
```



例：累加数组中的元素（通过堆栈传送参数地址）

```
code2    segment
         assume    cs:code2
```

```
stack_strc    struc
               save_bp    dw    ?
               save_cs_ip  dw    2 dup(?)
               par3_addr   dw    ?
               par2_addr   dw    ?
               par1_addr   dw    ?
stack_strc    ends
```

```
proadd    proc    far
         .....
         .....
proadd    endp

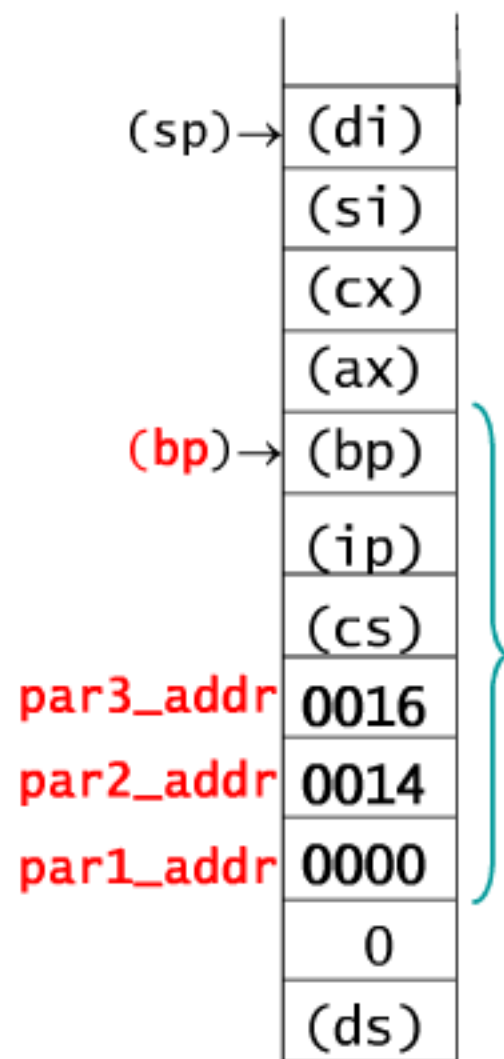
code2     ends
         end      start
```



```
proadd    proc    far
          push    bp
          mov     bp, sp
          push    ax
          push    cx
          push    si
          push    di
          mov     si, [bp].par1_addr
          mov     di, [bp].par2_addr
          mov     cx, [di]
          mov     di, [bp].par3_addr
          xor     ax, ax

next:
          add     ax, [si]
          add     si, 2
          loop    next
          mov     [di], ax
          pop     di
          pop     si
          pop     cx
          pop     ax
          pop     bp
          ret     6

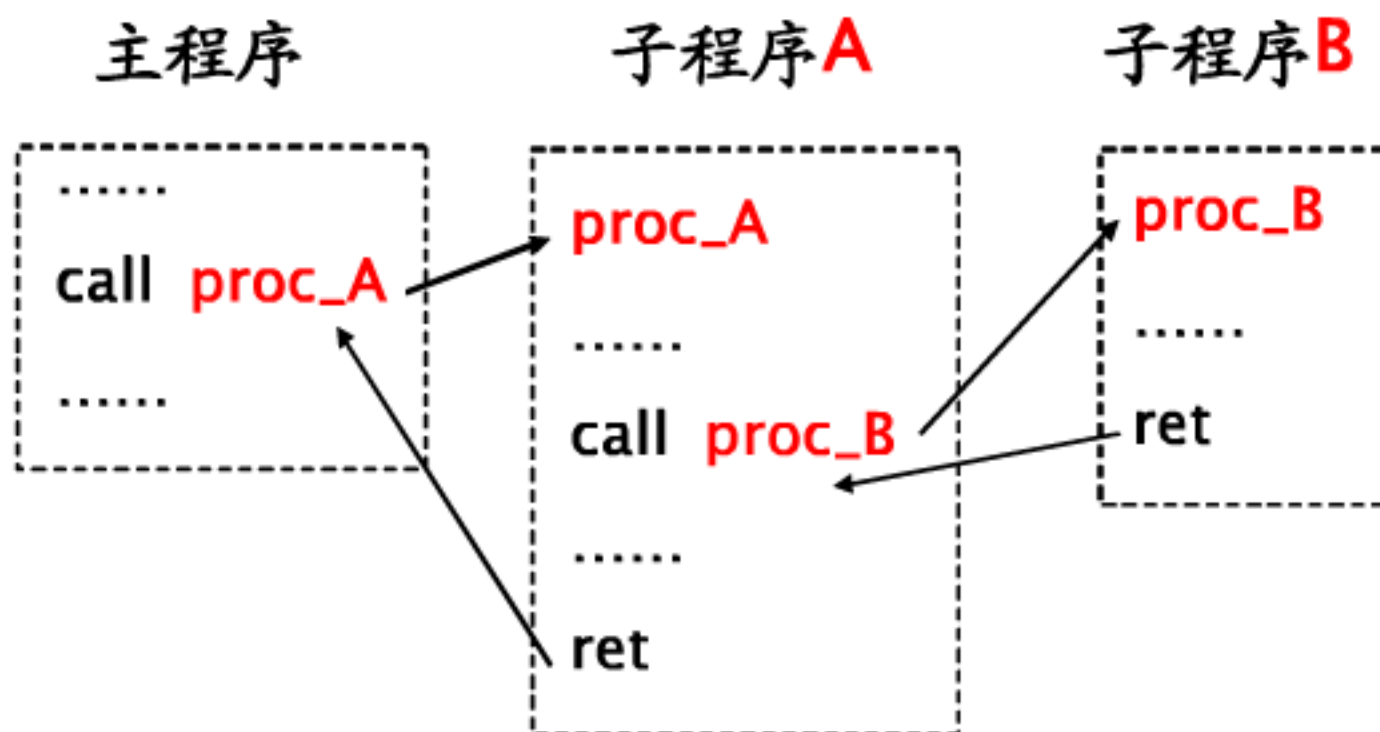
proadd    endp
```





5. 子程序的嵌套与递归

子程序的嵌套:



递归子程序: **n!**



例：计算 $n!$

```
frame    struc
save_bp  dw    ?
save_cs_ip dw   2  dup(?)
n         dw    ?
result_addr dw   ?
frame    ends

data     segment
n_v      dw    3
result   dw    ?
data     ends

stack    segment
dw       128      dup (?)
tos      label    word
stack    ends
```




code segment

main proc far

assume cs:code, ds:data, ss:stack

start:

mov ax, stack

mov ss, ax

mov sp, offset tos

push ds

sub ax, ax

push ax

mov ax, data

mov ds, ax

mov bx, offset result

push bx

mov bx, n_v

push bx

call far ptr fact

ret

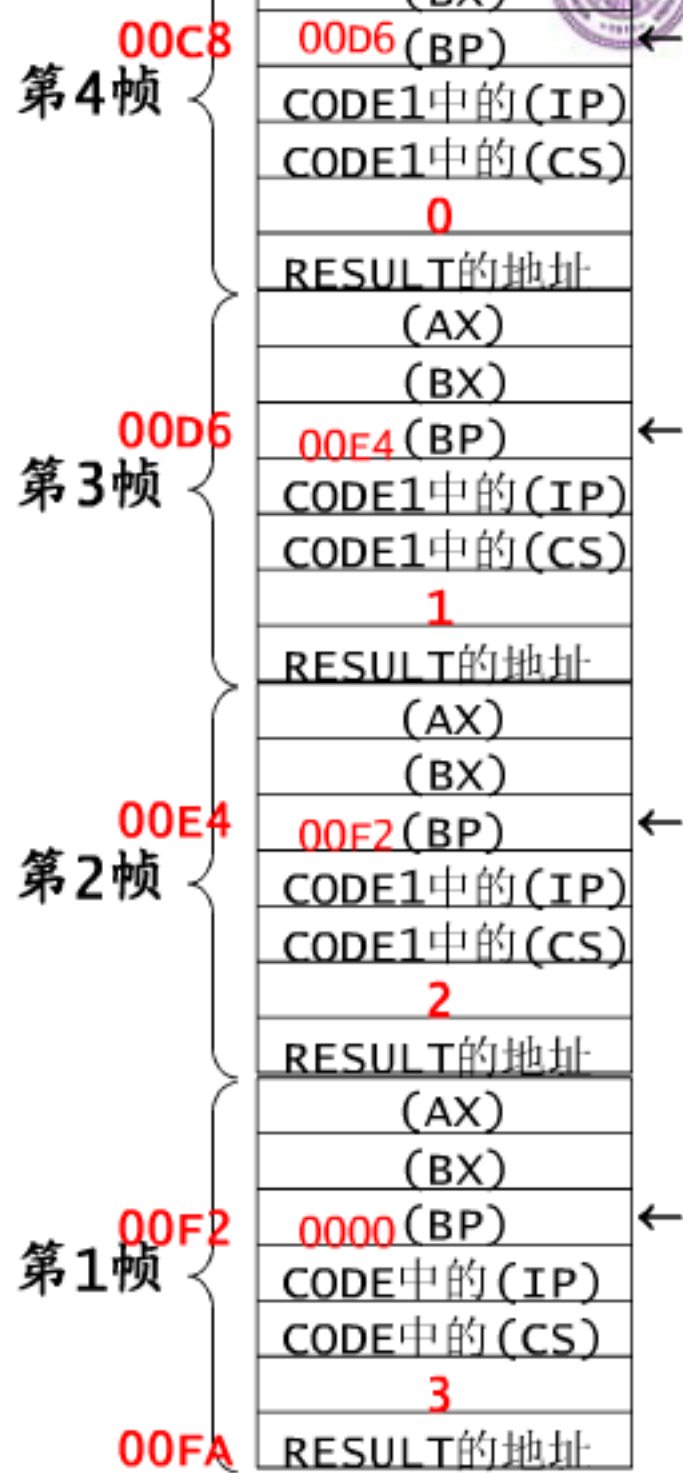
main endp

code ends



```
code1 segment
assume cs:code1
fact proc far
    push bp
    mov bp, sp
    push bx
    push ax

    mov bx, [bp].result_addr
    mov ax, [bp].n
    cmp ax, 0
    je done
    push bx
    dec ax
    push ax
    call far ptr fact
    mov bx, [bp].result_addr
    mov ax, [bx]
    mul [bp].n
    jmp short return
done:
    mov ax, 1
return:
    mov [bx], ax
    pop ax
    pop bx
    pop bp
    ret 4
fact endp
code1 ends
```



```

C:\ASM>debug n.exe
-r
AX=0000 BX=0000 CX=015D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0CEF ES=0CEF SS=0CFF CS=0D10 IP=0000  NU UP EI PL NZ NA PO NC
0D10:0000 B8000D          MOV     AX,0D00
-u
0D10:0000 B8000D          MOV     AX,0D00
0D10:0003 8ED0          MOV     SS,AX
0D10:0005 BC0001          MOV     SP,0100
0D10:0008 1E             PUSH    DS
0D10:0009 2BC0          SUB     AX,AX
0D10:000B 50             PUSH    -u
0D10:000C B8FF0C          MOV     0D10:0020 55          PUSH    BP
0D10:000F 8ED8          MOV     0D10:0021 8BEC        MOV     BP,SP
0D10:0011 BB0200          MOV     0D10:0023 53          PUSH    BX
0D10:0014 53             PUSH    0D10:0024 50          PUSH    AX
0D10:0015 8B1E0000        MOV     0D10:0025 8B5E08        MOV     BX,[BP+08]
0D10:0019 53             PUSH    0D10:0028 8B4606        MOV     AX,[BP+06]
0D10:001A 9A00000120D     CALL    0D10:002B 3D0000        CMP     AX,0000
0D10:001F CB             RETF    0D10:002E 7412         JZ      0042
                                0D10:0030 53          PUSH    BX
                                0D10:0031 48          DEC     AX
                                0D10:0032 50          PUSH    AX
                                0D10:0033 9A00000120D   CALL    0D12:0000
                                0D10:0038 8B5E08        MOV     BX,[BP+08]
                                0D10:003B 8B07          MOV     AX,[BX]
                                0D10:003D F76606        MUL     WORD PTR [BP+06]

```

ES 0CEFh
DS 0CFFh
SS 0D00h
CS 0D10h



code中的CS和IP 0D10H 001FH
code1中的CS和IP 0D12H 0018H



```
-r
AX=0001 BX=0002 CX=015D DX=0000 SP=00C4 BP=00C8 SI=0000
DS=0CFF ES=0CEF SS=0D00 CS=0D12 IP=0027  NV UP EI PL NZ
0D12:0027 58          POP     AX
-dss:80
0D00:0080  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0D00:0090  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0D00:00A0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0D00:00B0  00 00 00 00 00 00 00 00 00-00 00 01 00 C8 00 27
0D00:00C0  12 0D EB 06 00 00 02 00-D6 00 18 00 12 0D 00 00
0D00:00D0  02 00 01 00 02 00 E4 00-18 00 12 0D 01 00 02 00
0D00:00E0  02 00 02 00 F2 00 18 00-12 0D 02 00 02 00 FF 0C
0D00:00F0  03 00 00 00 1F 00 10 0D-03 00 02 00 00 00 EF 0C
```

00C8

00D6

00D6

00E4

00E4

00F2

00F2

0000

00FA

(AX)
(BX)
(BP)
CODE1中的(IP)
CODE1中的(CS)
0
RESULT的地址
(AX)
(BX)
(BP)
CODE1中的(IP)
CODE1中的(CS)
1
RESULT的地址
(AX)
(BX)
(BP)
CODE1中的(IP)
CODE1中的(CS)
2
RESULT的地址
(AX)
(BX)
(BP)
CODE中的(IP)
CODE中的(CS)
3
RESULT的地址

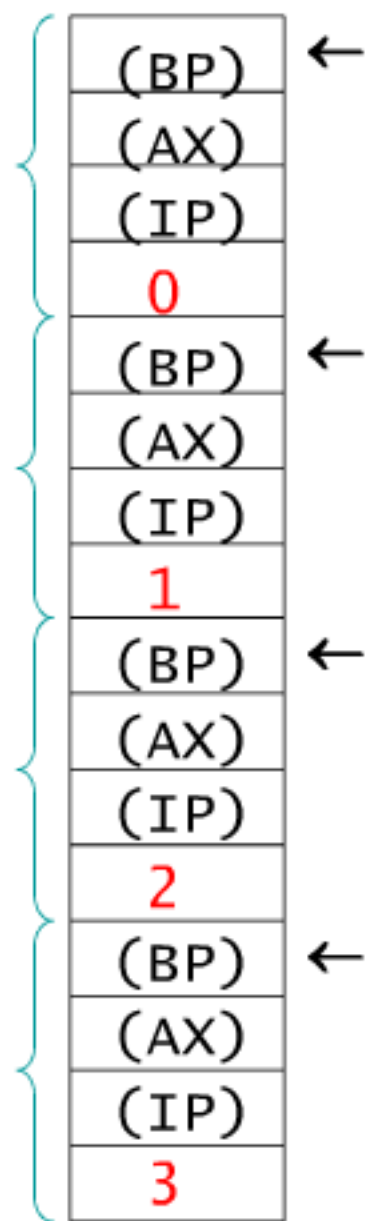


例：计算n!

主程序：

```
mov  bx, n_v
push bx
call fact
pop  result
```

```
fact  proc  near
      push  ax
      push  bp
      mov   bp, sp
      mov   ax, [bp+6]
      cmp   ax, 0
      jne   fact1
      inc   ax
      jmp   exit
fact1: dec   ax
      push  ax
      call  fact
      pop   ax
      mul   word ptr [bp+6]
exit:  mov   [bp+6], ax
      pop   bp
      pop   ax
      ret
fact  endp
```





多个模块之间的参数传送:

局部符号: 在本模块中定义, 在本模块中引用的符号

外部符号: 在某一模块中定义, 在另一模块中引用的符号

PUBLIC 符号 **EXTRN** 符号: 类型

; proadd1.asm

```
extrn  proadd : far
.....
.....
code1  segment
start:
        .....
        call  far ptr proadd
        .....
code1  ends
end      start
```

; proadd2.asm

```
public proadd
.....
.....
code2  segment
proadd proc far
        .....
        ret
proadd endp
code2  ends
end
```



例: ; proadd1.asm

```
extrn  proadd : far
data   segment common
        ary      dw  1,2,3,4,5,6,7,8,9,10
        count    dw  10
        sum       dw  ?
data   ends
code1  segment
main   proc      far
        assume   cs:code1,  ds:data
start: push      ds
        sub      ax, ax
        push     ax
        mov      ax, data
        mov      ds, ax
        call     far ptr proadd
        ret
main   endp
code1  ends
end     start
```



; proadd2.asm

public proadd

```
data segment common
    ary dw 1,2,3,4,5,6,7,8,9,10
    count dw 10
    sum dw ?
```

```
data ends
```

```
code2 segment
```

```
proadd proc far
```

```
    assume cs:code2,ds:data
```

```
    mov ax, data
```

```
    mov ds, ax
```

```
    push ax
```

```
    push cx
```

```
    push si
```

```
    lea si, ary
```

```
    mov cx, count
```

```
    xor ax, ax
```

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

```
proadd endp
```

```
code2 ends
```

```
end
```




第六章 子程序结构





主要内容:

- 过程定义伪操作
- 子程序的调用与返回
- 保存与恢复寄存器
- 子程序的参数传送
- 子程序的嵌套与递归



1. 过程定义伪操作

```
过程名  PROC  NEAR ( FAR )  
:  
:  
过程名  ENDP
```

- (1) **NEAR**属性: 调用程序和子程序在同一代码段中
(段内调用)
- (2) **FAR**属性: 调用程序和子程序不在同一代码段中
(段间调用)



code segment

```
main proc far
.....
    call subr1
.....
    ret
main endp

subr1 proc near
.....
    ret
subr1 endp

code ends
```

段内调用

```
segx segment
subt proc far
.....
    ret
subt endp
.....
    call subt
.....
segx ends

segx segment
.....
    call subt
.....
segx ends
```

段间调用

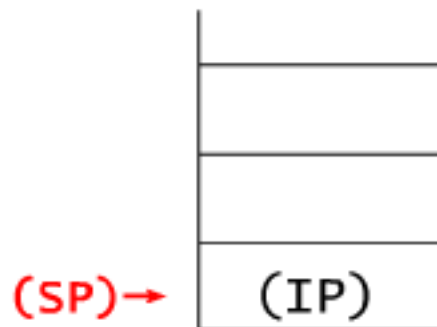


2. 子程序的调用与返回

子程序调用：隐含使用堆栈保存返回地址

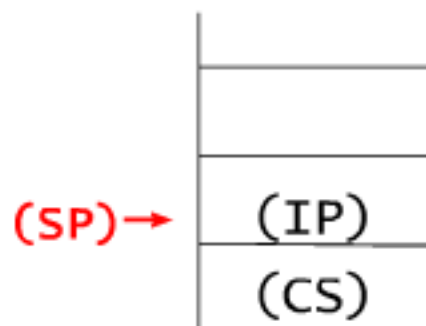
`call near ptr subp`

- (1) 保存返回地址
- (2) 转子程序



`call far ptr subp`

- (1) 保存返回地址
- (2) 转子程序



子程序返回： `ret`



3. 保存与恢复寄存器

```
subt    proc    far
        push    ax
        push    bx
        push    cx
        push    dx
        .....
        .....
        pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret
subt    endp
```



4. 子程序的参数传送

- (1) 通过寄存器传送参数
- (2) 通过存储器传送参数
- (3) 通过地址表传送参数地址
- (4) 通过堆栈传送参数或参数地址
- (5) 多个模块之间的参数传送



例：十进制到十六进制的转换（通过寄存器传送参数）

```
decihex segment ; 10→16
        assume cs: decihex
main     proc far
        push    ds
        sub     ax, ax
        push    ax
repeat:  call    decibin    ; 10→2
        call    crlf       ; 回车换行
        call    binihex    ; 2→16
        call    crlf
        jmp     repeat
        ret
main     endp
.....
.....
..... ; 三个子程序
decihex ends
end      main
```




```
decibin proc near  
      mov bx, 0  
newchar: mov ah, 1  
          int 21h  
          sub al, 30h  
          jl exit  
          cmp al, 9  
          jg exit  
          cbw  
          xchg ax, bx  
          mov cx, 10  
          mul cx  
          xchg ax, bx  
          add bx, ax  
          jmp newchar  
exit: ret  
decibin endp
```

```
binihex proc near  
      mov ch, 4  
rotate: mov cl, 4  
          rol bx, cl  
          mov al, bl  
          and al, 0fh  
          add al, 30h  
          cmp al, 3ah  
          jl printit  
          add al, 7  
printit: mov dl, al  
          mov ah, 2  
          int 21h  
          dec ch  
          jnz rotate  
          ret  
binihex endp
```

```
crlf proc near  
      mov dl, 0dh  
      mov ah, 2  
      int 21h  
      mov dl, 0ah  
      mov ah, 2  
      int 21h  
      ret  
crlf endp
```



例：十六进制到十进制的转换（通过寄存器传送参数）

```
hexidec segment ; 16→10
main      proc far
start:    push ds
          sub  ax, ax
          push ax

repeat:   call hexibin ; 16→2
          call crlf
          call binidec ; 2→10
          call crlf
          jmp  repeat
          ret
main      endp
.....
.....
.....
hexidec   ends
          end    start
```



```
hexibin  proc  near
newchar:  mov    bx, 0
          mov    ah, 1
          int    21h
          sub    al, 30h
          jl     exit
          cmp    al, 10
          jl     add_to
          sub    al, 27h
          cmp    al, 0ah
          jl     exit
          cmp    al, 10h
          jge    exit
add_to:   mov    cl, 4
          shl    bx, cl
          mov    ah, 0
          add    bx, ax
          jmp    newchar
exit:     ret
hexibin  endp
```

```
binidec  proc  near
          mov    cx, 10000d
          call   dec_div
          mov    cx, 1000d
          call   dec_div
          mov    cx, 100d
          call   dec_div
          mov    cx, 10d
          call   dec_div
          mov    cx, 1d
          call   dec_div
          ret
```

binidec endp

```
dec_div  proc  near
          mov    ax, bx
          mov    dx, 0
          div    cx
          mov    bx, dx
          mov    dl, al
          add    dl, 30h
          mov    ah, 2
          int    21h
          ret
dec_div  endp
```





例：累加数组中的元素（通过存储器传送参数）

```

data    segment
        ary      dw  1,2,3,4,5,6,7,8,9,10
        count    dw  10
        sum      dw  ?
data    ends

code    segment
main    proc      far
        assume   cs:code,ds:data
        push     ds
        sub      ax, ax
        push     ax
        mov      ax, data
        mov      ds, ax
        call     proadd
        ret
main    endp
.....
code    ends
        end      main
    
```

;proadd子程序

```

proadd  proc  near
        push     ax
        push     cx
        push     si
        lea      si, ary
        mov      cx, count
        xor      ax, ax
next:   add      ax, [si]
        add      si, 2
        loop     next
        mov      sum, ax
        pop      si
        pop      cx
        pop      ax
        ret
proadd  endp
    
```



如果数据段定义如下：

```
data segment

    ary      dw  1,2,3,4,5,6,7,8,9,10
    count    dw  10
    sum       dw  ?

    ary1     dw  10,20,30,40,50,60,70,80,90,100
    count1   dw  10
    sum1      dw  ?

data ends
```

如果直接访问内存变量，那么累加**数组ary**
和**数组ary1**中的元素不能用同一个子程序 proadd。



例：累加数组中的元素（通过地址表传送参数地址）

```
data segment
    ary      dw 1,2,3,4,5,6,7,8,9,10
    count    dw 10
    sum       dw ?
    table     dw 3 dup (?)           ; 地址表
data ends
code segment
main proc far
    assume cs:code, ds:data
    push     ds
    sub      ax, ax
    push     ax
    mov      ax, data
    mov      ds, ax
    mov      table, offset ary
    mov      table+2, offset count
    mov      table+4, offset sum    ; 建立地址表
    mov      bx, offset table      ; 地址表的地址->bx
    call     proadd
    ret
main endp
```



proadd proc near

push ax

push cx

push si

push di

mov si, [bx]

mov di, [bx+2]

mov cx, [di]

mov di, [bx+4]

xor ax, ax

next:

add ax, [si]

add si, 2

loop next

mov [di], ax

pop di

pop si

pop cx

pop ax

ret

proadd endp

code ends

end main

ary→	1d	0000
	2d	
	3d	
	4d	
	5d	
	6d	
	7d	
	8d	
	9d	
	10d	
count→	10d	0014
sum→	55d	0016
table→	0000	0018 →(bx)
	0014	
	0016	



通过**堆栈**传送参数或参数地址:

```
stack segment
    dw 100 dup (?)
    tos label word
stack ends
start:
    mov ax, stack
    mov ss, ax
    mov sp, offset tos
    push ds
    sub ax, ax
    push ax
    .....
```



```
stack segment      stack
    dw 100 dup (?)
    tos label word
stack ends
```





例：累加数组中的元素（通过堆栈传送参数地址）

```
data    segment
        ary      dw  1,2,3,4,5,6,7,8,9,10
        count    dw  10
        sum      dw  ?
data    ends
```

```
stack   segment
        dw  100      dup (?)
        tos  label  word
stack   ends
```

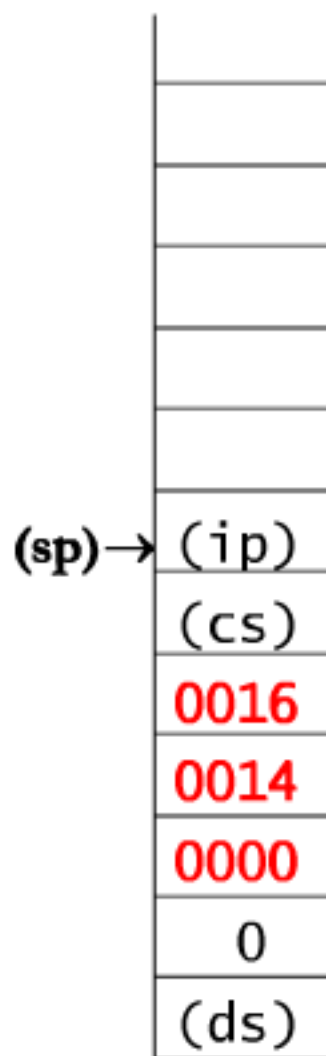


```
code1 segment
main  proc      far
      assume    cs:code1, ds:data, ss:stack
start:
```

```
      mov       ax, stack
      mov       ss, ax
      mov       sp, offset tos
      push     ds
      sub      ax, ax
      push     ax
      mov      ax, data
      mov      ds, ax
      mov      bx, offset ary
      push     bx
      mov      bx, offset count
      push     bx
      mov      bx, offset sum
      push     bx

      call     far ptr proadd

      ret
main  endp
code1 ends
```





```
code2    segment
        assume cs:code2
```

```
proadd   proc    far
```

```
    push    bp
    mov     bp, sp
```

```
    push    ax
    push    cx
    push    si
    push    di
```

```
    mov     si, [bp+0ah]
    mov     di, [bp+8]
    mov     cx, [di]
    mov     di, [bp+6]
```

```
        xor     ax, ax
next:    add     ax, [si]
        add     si, 2
        loop    next
        mov     [di], ax
```

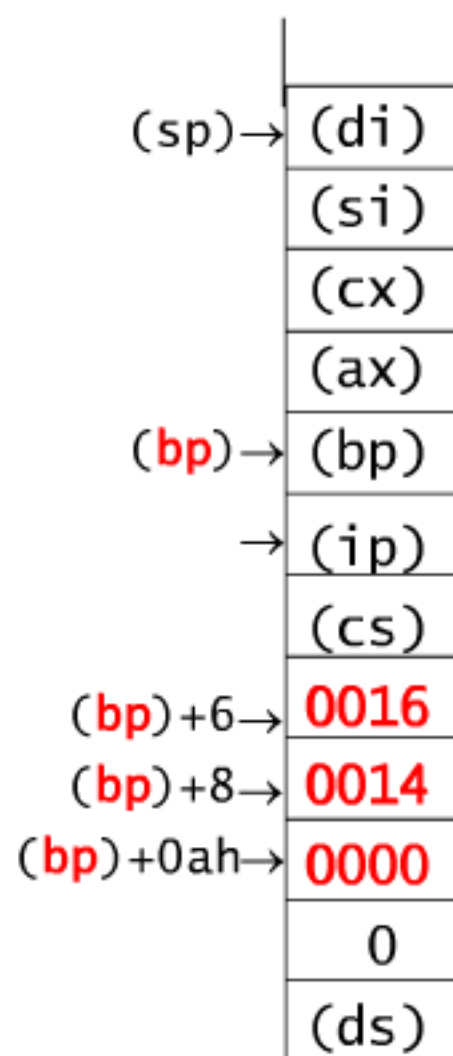
```
        pop     di
        pop     si
        pop     cx
        pop     ax
```

```
        pop     bp
```

```
        ret     6
```

```
proadd   endp
```

```
code2    ends
end start
```





结构伪操作 **STRUC**:

定义一种可包含不同类型数据的结构模式

格式: 结构名 **STRUC**
 字段名1 **DB** ?
 字段名2 **DW** ?
 字段名3 **DD** ?

 结构名 **ENDS**

例: 学生个人信息

```
STUDENT_DATA    STRUC  
                        NAME    DB    5    DUP (?)  
                        ID    DW    0  
                        AGE    DB    ?  
                        DEP    DB    10    DUP (?)  
STUDENT_DATA    ENDS
```



例：累加数组中的元素（通过堆栈传送参数地址）

```
code2    segment
         assume    cs:code2
```

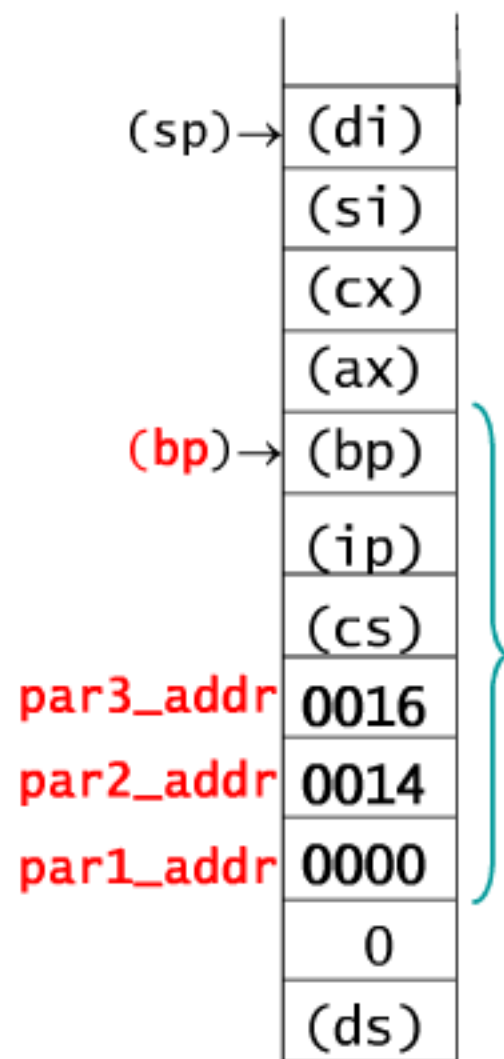
```
stack_strc    struc
               save_bp      dw      ?
               save_cs_ip    dw      2  dup(?)
               par3_addr     dw      ?
               par2_addr     dw      ?
               par1_addr     dw      ?
stack_strc    ends
```

```
proadd    proc    far
           .....
           .....
proadd    endp

code2     ends
end       start
```



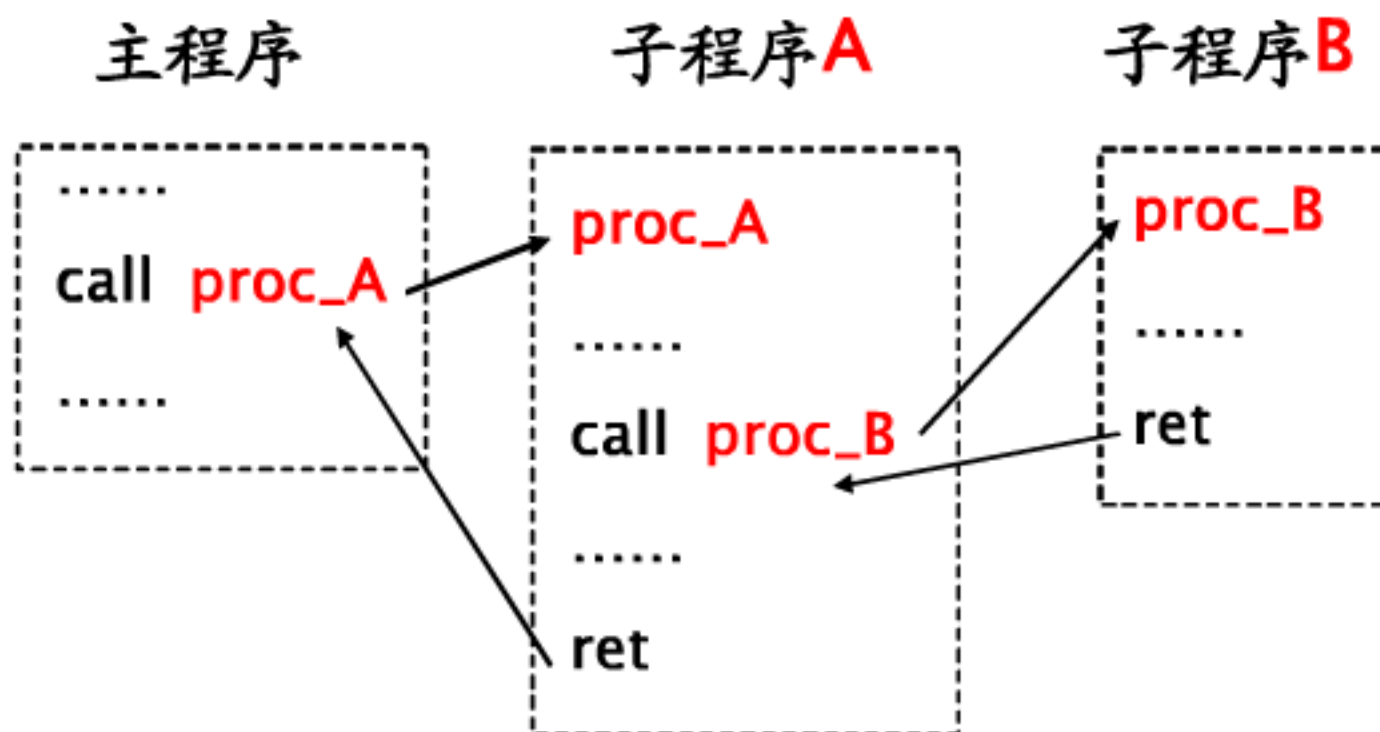
```
proadd  proc  far
        push  bp
        mov   bp, sp
        push  ax
        push  cx
        push  si
        push  di
        mov   si, [bp].par1_addr
        mov   di, [bp].par2_addr
        mov   cx, [di]
        mov   di, [bp].par3_addr
        xor   ax, ax
next:
        add   ax, [si]
        add   si, 2
        loop  next
        mov   [di], ax
        pop   di
        pop   si
        pop   cx
        pop   ax
        pop   bp
        ret   6
proadd  endp
```





5. 子程序的嵌套与递归

子程序的嵌套:



递归子程序: **n!**



例：计算 $n!$

```
frame    struc
save_bp   dw    ?
save_cs_ip dw    2  dup(?)
n         dw    ?
result_addr dw   ?
frame    ends

data     segment
n_v      dw    3
result   dw    ?
data     ends

stack    segment
dw       128      dup (?)
tos      label    word
stack    ends
```




```
code segment
main proc far
    assume cs:code, ds:data, ss:stack
start:
    mov ax, stack
    mov ss, ax
    mov sp, offset tos
    push ds
    sub ax, ax
    push ax
    mov ax, data
    mov ds, ax
    mov bx, offset result
    push bx
    mov bx, n_v
    push bx

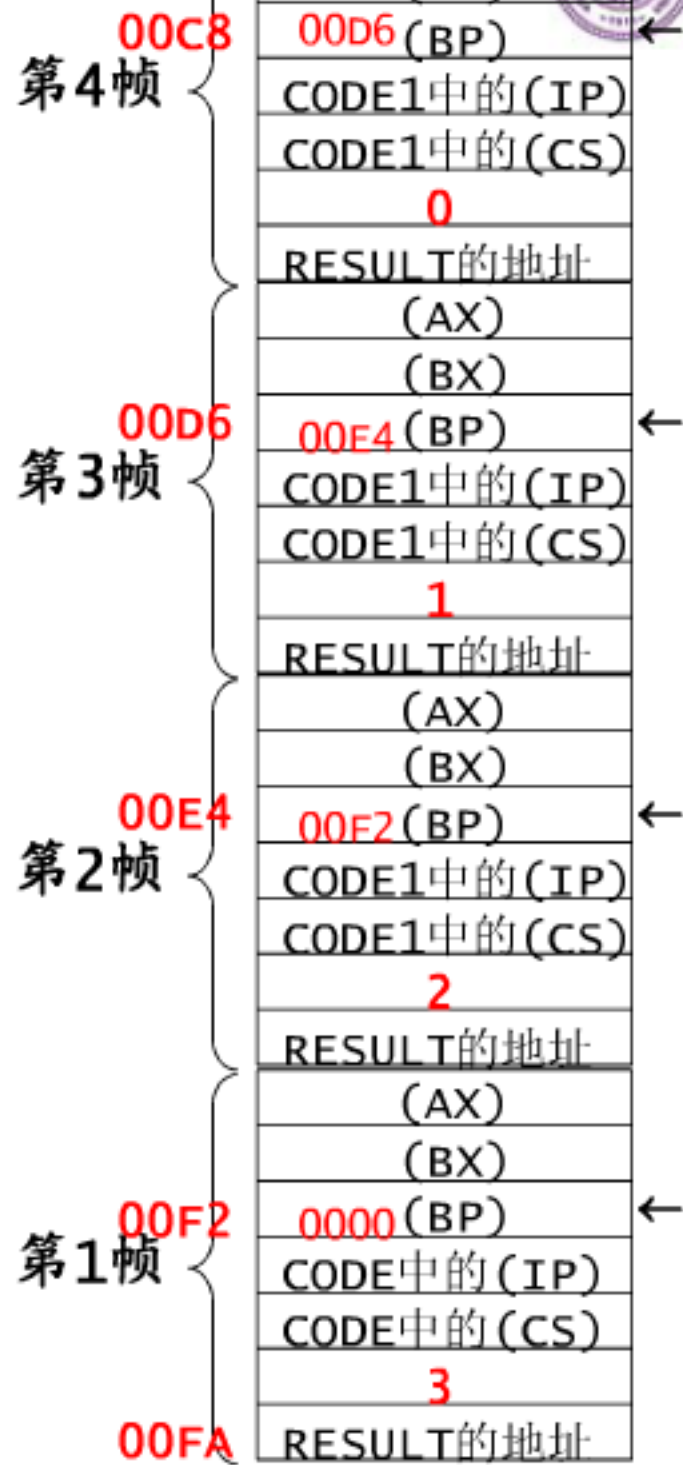
    call far ptr fact

    ret
main endp
code ends
```



```
code1 segment
assume cs:code1
fact proc far
    push bp
    mov bp, sp
    push bx
    push ax

    mov bx, [bp].result_addr
    mov ax, [bp].n
    cmp ax, 0
    je done
    push bx
    dec ax
    push ax
    call far ptr fact
    mov bx, [bp].result_addr
    mov ax, [bx]
    mul [bp].n
    jmp short return
done: mov ax, 1
return: mov [bx], ax
        pop ax
        pop bx
        pop bp
        ret 4
fact endp
code1 ends
```



```

C:\ASM>debug n.exe
-r
AX=0000 BX=0000 CX=015D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0CEF ES=0CEF SS=0CFF CS=0D10 IP=0000  NU UP EI PL NZ NA PO NC
0D10:0000 B8000D          MOV     AX,0D00
-u
0D10:0000 B8000D          MOV     AX,0D00
0D10:0003 8ED0          MOV     SS,AX
0D10:0005 BC0001          MOV     SP,0100
0D10:0008 1E             PUSH    DS
0D10:0009 2BC0          SUB     AX,AX
0D10:000B 50             PUSH    -u
0D10:000C B8FF0C          MOV     0D10:0020 55          PUSH    BP
0D10:000F 8ED8          MOV     0D10:0021 8BEC        MOV     BP,SP
0D10:0011 BB0200          MOV     0D10:0023 53          PUSH    BX
0D10:0014 53             PUSH    0D10:0024 50          PUSH    AX
0D10:0015 8B1E0000        MOV     0D10:0025 8B5E08        MOV     BX,[BP+08]
0D10:0019 53             PUSH    0D10:0028 8B4606        MOV     AX,[BP+06]
0D10:001A 9A00000120D     CALL    0D10:002B 3D0000        CMP     AX,0000
0D10:001F CB             RETF   0D10:002E 7412          JZ      0042
                                0D10:0030 53          PUSH    BX
                                0D10:0031 48          DEC     AX
                                0D10:0032 50          PUSH    AX
                                0D10:0033 9A00000120D     CALL    0D12:0000
                                0D10:0038 8B5E08        MOV     BX,[BP+08]
                                0D10:003B 8B07          MOV     AX,[BX]
                                0D10:003D F76606        MUL     WORD PTR [BP+06]
-
ES 0CEFh
DS 0CFFh
SS 0D00h
CS 0D10h

```



code中的CS和IP 0D10H 001FH
code1中的CS和IP 0D12H 0018H



```
-r
AX=0001 BX=0002 CX=015D DX=0000 SP=00C4 BP=00C8 SI=0000
DS=0CFF ES=0CEF SS=0D00 CS=0D12 IP=0027  NV UP EI PL NZ
0D12:0027 58          POP      AX
-dss:80
0D00:0080  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0D00:0090  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0D00:00A0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0D00:00B0  00 00 00 00 00 00 00 00 00-00 00 01 00 C8 00 27
0D00:00C0  12 0D EB 06 00 00 02 00-D6 00 18 00 12 0D 00 00
0D00:00D0  02 00 01 00 02 00 E4 00-18 00 12 0D 01 00 02 00
0D00:00E0  02 00 02 00 F2 00 18 00-12 0D 02 00 02 00 FF 0C
0D00:00F0  03 00 00 00 1F 00 10 0D-03 00 02 00 00 00 EF 0C
```

00C8

00D6

(AX)

(BX)

(BP)

CODE1中的(IP)

CODE1中的(CS)

0

RESULT的地址

(AX)

(BX)

(BP)

00D6

00E4

CODE1中的(IP)

CODE1中的(CS)

1

RESULT的地址

(AX)

(BX)

(BP)

00E4

00F2

CODE1中的(IP)

CODE1中的(CS)

2

RESULT的地址

(AX)

(BX)

(BP)

00F2

0000

CODE中的(IP)

CODE中的(CS)

3

RESULT的地址

00FA

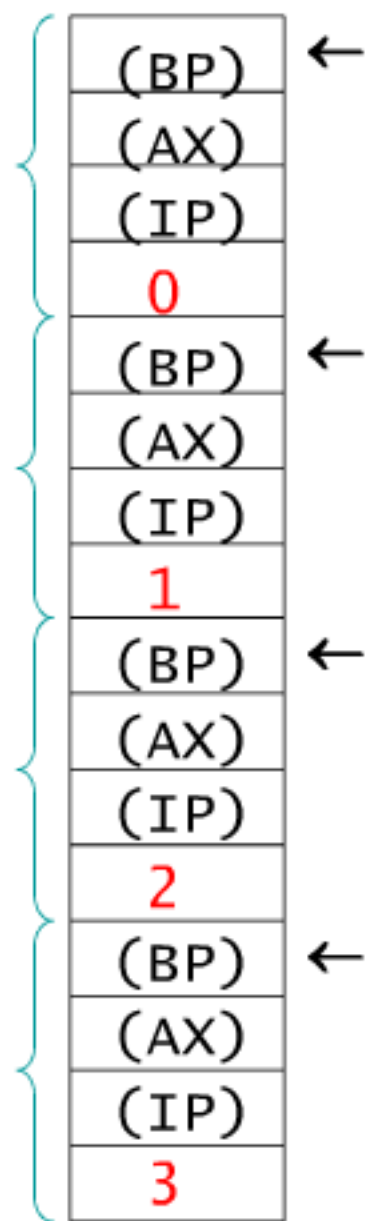


例：计算n!

主程序：

```
mov  bx, n_v
push bx
call fact
pop  result
```

```
fact  proc  near
      push  ax
      push  bp
      mov   bp, sp
      mov   ax, [bp+6]
      cmp   ax, 0
      jne   fact1
      inc   ax
      jmp   exit
fact1: dec   ax
      push  ax
      call  fact
      pop   ax
      mul   word ptr [bp+6]
exit:  mov   [bp+6], ax
      pop   bp
      pop   ax
      ret
fact  endp
```





多个模块之间的参数传送:

局部符号: 在本模块中定义, 在本模块中引用的符号

外部符号: 在某一模块中定义, 在另一模块中引用的符号

PUBLIC 符号 **EXTRN** 符号: 类型

; proadd1.asm

```
extrn  proadd : far
.....
.....
code1  segment
start:
        .....
        call  far ptr proadd
        .....
code1  ends
end      start
```

; proadd2.asm

```
public proadd
.....
.....
code2  segment
proadd proc far
        .....
        ret
proadd endp
code2  ends
end
```



例: ; proadd1.asm

```
extrn  proadd : far
data   segment common
        ary      dw  1,2,3,4,5,6,7,8,9,10
        count    dw  10
        sum       dw  ?
data   ends
code1  segment
main   proc      far
        assume   cs:code1,  ds:data
start: push      ds
        sub      ax, ax
        push     ax
        mov      ax, data
        mov      ds, ax
        call     far ptr proadd
        ret
main   endp
code1  ends
        end      start
```



; proadd2.asm

public proadd

```
data segment common
    ary dw 1,2,3,4,5,6,7,8,9,10
    count dw 10
    sum dw ?
```

```
data ends
```

```
code2 segment
```

```
proadd proc far
```

```
    assume cs:code2,ds:data
```

```
    mov ax, data
```

```
    mov ds, ax
```

```
    push ax
```

```
    push cx
```

```
    push si
```

```
    lea si, ary
```

```
    mov cx, count
```

```
    xor ax, ax
```

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

```
proadd endp
```

```
code2 ends
```

```
end
```