

汇编语言程序设计

Assembly Language Programming

第四章

基本汇编语言程序设计

4.4 子程序设计

过程（子程序）定义伪操作

```
procedure_name  PROC  NEAR ( FAR )  
.....  
procedure_name  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)

调用程序和子程序在同一代码段中



```
code segment
```

```
assume .....
```

```
Start:
```

```
.....
```

```
call subr1
```

```
.....
```

```
Mov ah,4ch
```

```
int 21h
```

```
subr1 proc near
```

```
.....
```

```
ret
```

```
subr1 endp
```

```
code ends
```

```
code segment
```

```
main proc far
```

```
.....
```

```
call subr1
```

```
.....
```

```
ret
```

```
main endp
```

```
subr1 proc near
```

```
.....
```

```
ret
```

```
subr1 endp
```

```
code ends
```


调用程序和子程序不在同一代码段中

subseg segment

subt proc far

.....

ret

subt endp

subseg ends

mainseg segment

.....

call subt

.....

mainseg ends

子程序的调用和返回

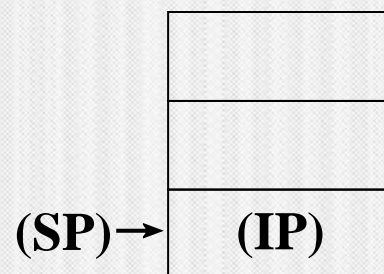
子程序调用（中断调用）：隐含使用堆栈保存返回地址

call **near** ptr subp

(1) 保存返回地址 PUSH IP

(2) 转子程序

(IP) ← subp的偏移地址



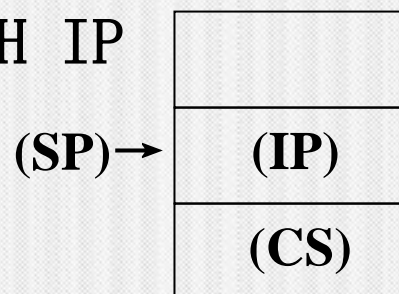
call **far** ptr subp

(1) 保存返回地址 PUSH CS; PUSH IP

(2) 转子程序

(CS) ← subp的段地址

(IP) ← subp的偏移地址



注意事项

- ♥ 在过程定义体内，必须有一条RET指令能被执行到。
- ♥ 调用时，最好不要强制改变调用类型。
- ♥ 子程序保护现场。
- ♥ 堆栈操作指令必须配对。
- ♥ 定义允许嵌套和递归。

保存与恢复现场

subt proc far

push ax

push bx

push cx

push dx

.....

pop dx

pop cx

pop bx

pop ax

ret

subt endp

实现回车换行

```
CSEG  SEGMENT
ASSUME CS:CSEG
```

```
START:
```

```
...
```

```
    CALL LF_CR
```

```
...
```

```
    MOV  AH,4CH
```

```
    INT  21H
```

```
LF_CR PROC
```

```
; 保存现场
```

```
    MOV DL,0DH
```

```
    MOV AH,2
```

```
    INT 21H
```

```
    MOV DL,0AH
```

```
    MOV AH,2
```

```
    INT 21H
```

```
;恢复现场
```

```
    RET
```

```
LF_CR ENDP
```

```
CSEG  ENDS
```

```
END  START
```


程序与子程序之间的参数传递

♥ 区分

- ❖ 传入参数和传出参数
- ❖ 传值和传地址

♥ 方式

- ❖ 用寄存器来传递参数
- ❖ 用内存单元传递参数
- ❖ 用堆栈来传递参数

用内存数传递参数

♥ 适用于传递全局变量的情况

```
DSEG SEGMENT
    STR DB 'GOOD$'
DSEG ENDS
```

```
CSEG  SEGMENT
ASSUME .....
START:
    .....
    CALL PRINTSTR
    MOV  AH,4CH
    INT  21H
```

```
PRINTSTR  PROC
            LEA DX,STR 内存数传递参数
            MOV AH,9
            INT 21H
            RET  返回被调用的地方
PRINTSTR  ENDP

CSEG  ENDS
        END  START
```


用寄存器来传递参数(1)

♥ 传输单个变量

❖ 适用于参数较少的情况

♥ 例子：编写一个子程序，完成求平方的功能，输入参数通过DL传递，输出通过DX传递

```
SQUARE PROC NEAR
```

```
    PUSH AX 将AX中原来的数据压入栈中
```

```
    MOV AL,DL
```

```
    MUL AL
```

```
    MOV DX,AX
```

```
    POP AX 当子程序运行结束后将原来AX中的值在返回给AX寄存器
```

```
    RET
```

```
SQUARE ENDP
```


用寄存器来传递参数(2)

♥ 传输指针

❖ 传递一个缓冲区

♥ 例子：编写子程序累加长度为100的无符号字节缓冲区buffer，首地址通过BX传递，结果通过DX传回

```
SUM PROC NEAR
    PUSH CX
    MOV CX, 100
    MOV DX, 0
AGAIN:
    ADD DL, [BX]
    ADC DH, 0
    INC BX
    LOOP AGAIN
    POP CX
    RET
SUM ENDP
```


用堆栈来传递参数

♥ 适用性极大，高级语言均采用此方式

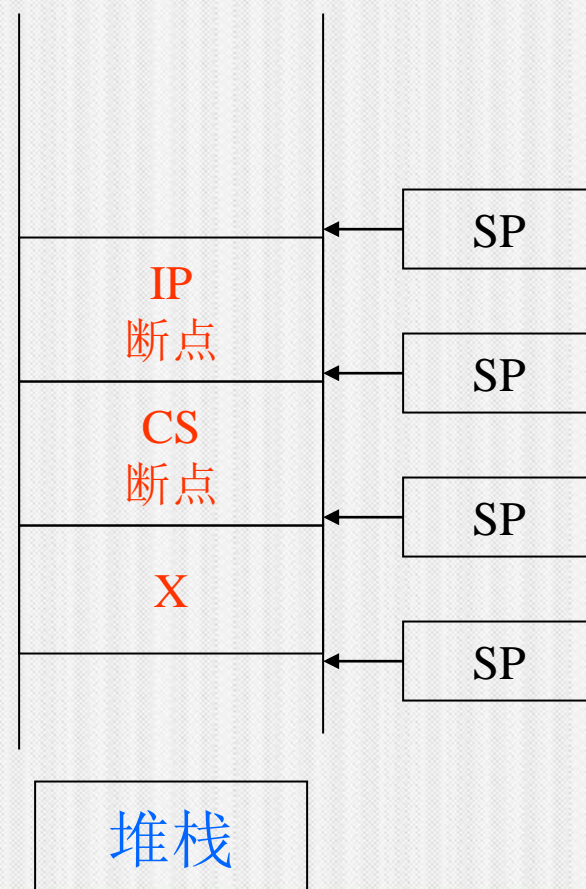
♥ 注意：

- ❖ 段内调用时只有IP入栈
- ❖ 段间调用时有CS，IP入栈

主程序通过堆栈传参数到子程序。设参数为X（WORD），子程序为FAR

♥主程序动作

- ❖ PUSH X
- ❖ PUSH CS
- ❖ PUSH IP



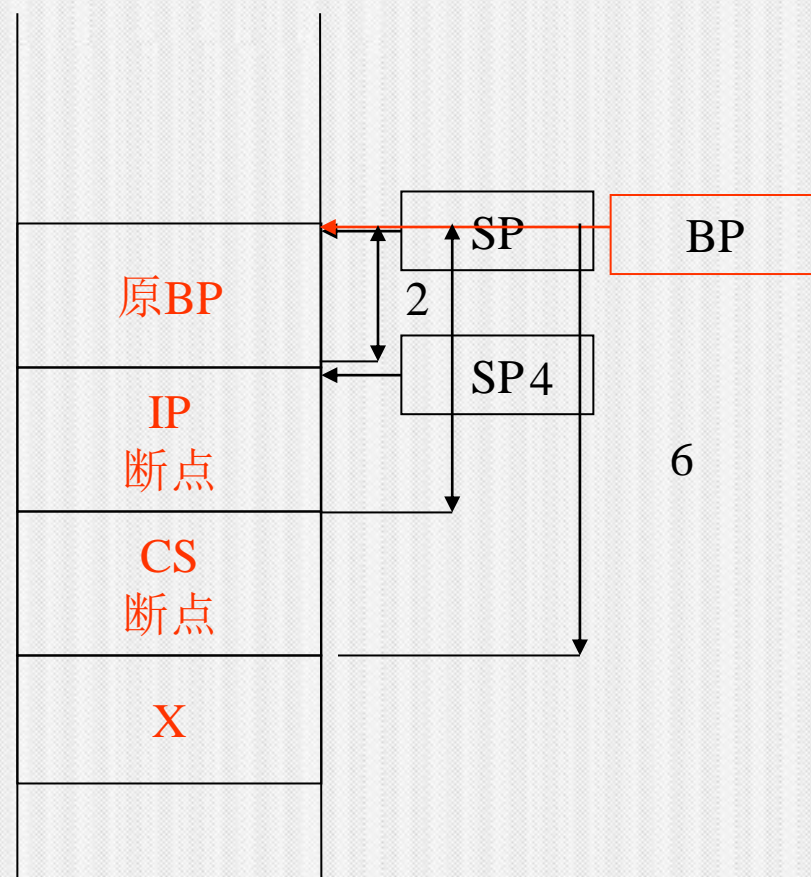
主程序通过堆栈传参数到子程序。设参数为X（WORD），子程序为FAR

♥子程序提取数据X

- ❖ PUSH BP
- ❖ MOV BP,SP
- ❖ WORD PTR [BP+6]

♥如果是Near调用

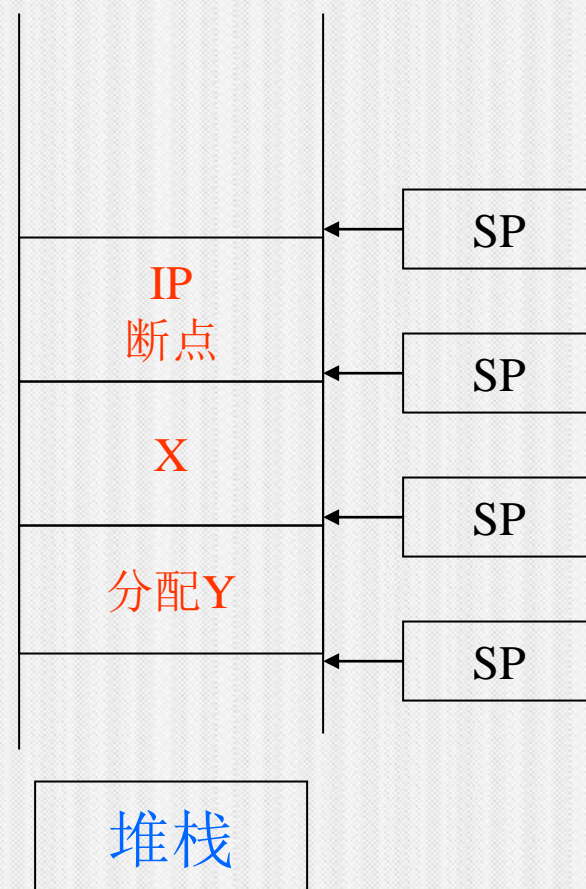
- ❖ WORD PTR [BP+4]



主程序通过堆栈传入和传出参数。设传入参数为X，
传出参数为Y（X,Y为WORD），子程序为NEAR

♥主程序动作

- ❖ SUB SP,2
- ❖ PUSH X
- ❖ PUSH IP



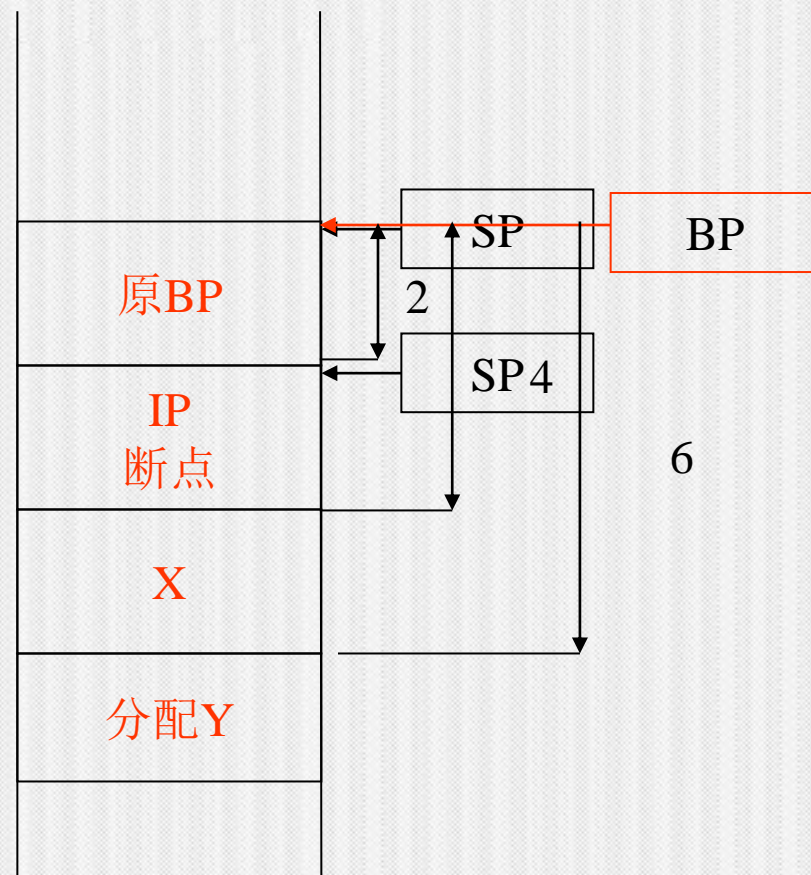
主程序通过堆栈传入和传出参数。设传入参数为X，
传出参数为Y（XY为WORD），子程序为NEAR

♥子程序提取数据X

- ❖ PUSH BP
- ❖ MOV BP,SP
- ❖ WORD PTR [BP+4]

♥传出数据

- ❖ WORD PTR [BP+6]



主程序通过堆栈传入和传出参数。设传入参数为X，
传出参数为Y（X,Y为WORD），子程序为NEAR

♥ RET

♥ 参数的销毁

❖ 主程序完成

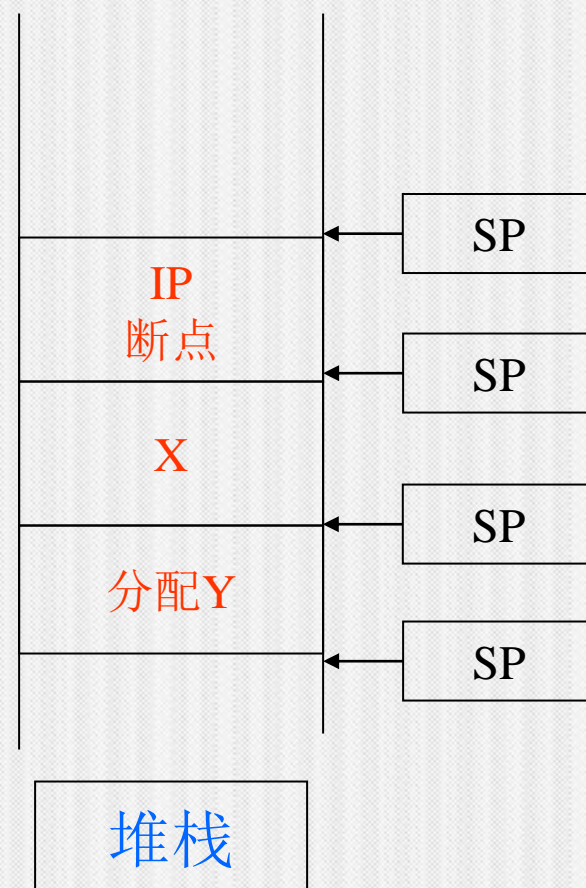
- ADD SP,2

❖ 子程序完成

- RET 2

❖ 取出传回的值

- POP AX



用堆栈来传递参数

编写子程序
完成 $a-b=c$

BP

原BP

CS: IP

a

b

预留给C

CS:IP是两个
占用四个字
节CS在高地
址, IP在低
地址

```
DATA    SEGMENT
        A DW 2
        B DW 1
        SUM DW 0
DATA    ENDS
```

```
SUBSEG  SEGMENT
ASSUME  CS:SUBSEG,DS:DATA
HTON_F  PROC    FAR
        PUSH    BP
        MOV     BP,SP
        PUSH    AX
        PUSH    BX

        MOV     BX,[BP+8]
        MOV     AX,[BP+6]
        SUB     AX,BX
        MOV     [BP+10],AX
        POP     BX
        POP     AX
        POP     BP

        清除a,b但不需要清除C
        RET     4
HTON_F  ENDP
SUBSEG  ENDS
```

```
CSEG  SEGMENT
ASSUME CS:CSEG, DS:DATA
START:

        MOV     AX,DATA
        MOV     DS,AX

        预留给C
        SUB     SP, 2
        PUSH    B
        PUSH    A
        CALL    HTON_F
        POP     SUM

        MOV     AH, 4CH
        INT     21H

CSEG  ENDS
        END     START
```


递归求解:

 $\text{Sum}(n)=1+2+\dots+n$ 若 $n=0$, $\text{Sum}(n)=0$;否则 $= \text{Sum}(n-1) + n$

```

n    DW    ...
Result DW    ?

```

```

SUB     SP,2
PUSH    n

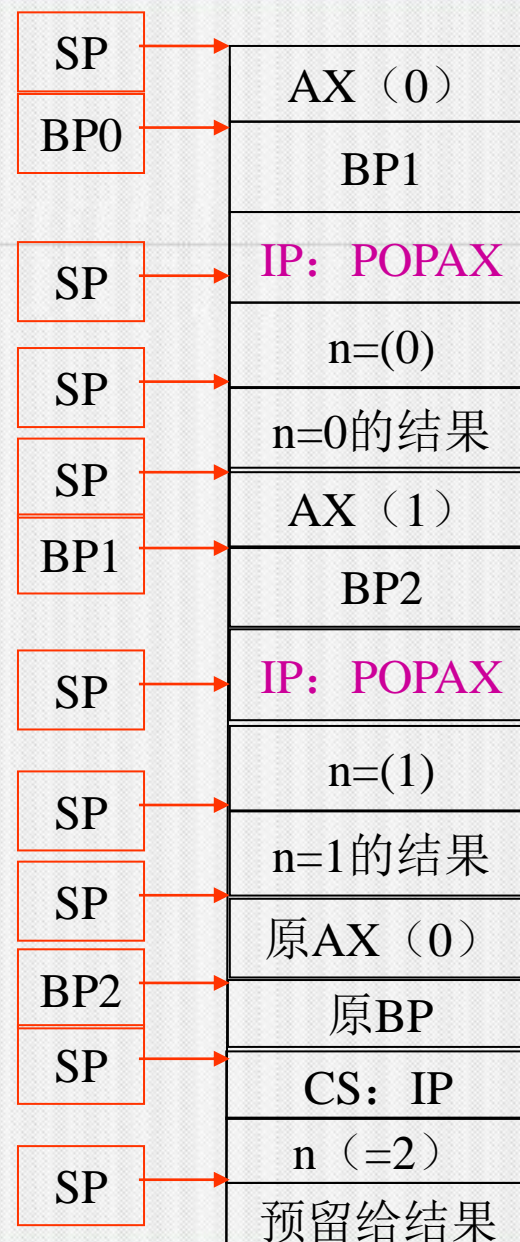
CALL    SUM
POP     Result

```

```

SUM PROC FAR
    PUSH BP
    MOV BP,SP
    PUSH AX
    MOV AX,[BP+6]
    CMP AX,0
    JNZ SUM1
    MOV AX,0
    JMP EXIT
SUM1: SUB SP,2
    DEC AX
    PUSH AX
    CALL SUM
    POP AX
    ADD AX,[BP+6]
EXIT:
    MOV [BP+8],AX
    POP AX
    POP BP
    RET 2

```



例4.8 从键盘输入有符号十进制数子程序

- ❖ 1) 读入一个字符：+、-、数字或其他，前两个做正负标志，转2) 否则到3)
- ❖ 2) 读入字符
- ❖ 3) 如果不是数字字符，则退出，否则取出数字，将前面保存的值乘以10，再加上输入的数，跳到2)，否则 转4)
- ❖ 4) 判断正负标志，若为负数，则取反。


```
read      proc      ;输入十进制数的通用子程序:read
            push bx  ;出口参数:AX = 补码表示的二进制数值
            push cx  ;说明:负数用“-”引导,数据范围是 +32767 ~ -32768
            push dx
            xor bx,bx ;BX 保存结果
            xor cx,cx ;CX 为正负标志,0 为正,1 为负
            mov ah,1  ;输入一个字符
            int 21h
            cmp al,'+' ;是“+”,继续输入字符
            jz read1
            cmp al,'-' ;是“-”,设置-1 标志
            jnz read2
            mov cx,-1
read1:     mov ah,1    ;继续输入字符
            int 21h
```



```

read2:  cmp al,'0'           ;不是 0 ~ 9 之间的字符,则输入数据结束
        jb read3
        cmp al,'9'
        ja read3
        sub al,30h           ;是 0 ~ 9 之间的字符,则转换为二进制数
                                ;利用移位指令,实现数值乘 10: BX ← BX × 10

        shl bx,1
        mov dx,bx
        shl bx,1
        shl bx,1
        add bx,dx
        ;
        mov ah,0
        add bx,ax           ;已输入数值乘 10 后,与新输入数值相加
        jmp read1          ;继续输入字符
read3:  cmp cx,0             ;是负数,进行求补
        jz read4
        neg bx
read4:  mov ax,bx            ;设置出口参数
        pop dx
        pop cx
        pop bx
        ret                 ;子程序返回

```


例4.9 输出有符号十进制数子程序

- 1) 首先判断数据是否是0、正数或负数，是0直接退出
- 2) 是负数，显示“-”，并求绝对值
- 3) 将数除以10，余数加30H，压栈（用于显示）；
- 4) 重复3)，直到商为0；
- 5) 依次从堆栈中取值显示。


```
write proc ;显示有符号十进制数的通用子程序:write
    push ax ;入口参数:共享变量 wtemp
    push bx
    push dx
    mov ax,wtemp ;取出显示数据
    test ax,ax ;判断数据是零、正数或负数
    jnz writel ;是零,显示“0”后退出
    mov dl,'0'
    mov ah,2
    int 21h
    jmp write5
writel: jns write2 ;是负数,显示“-”
    mov bx,ax ;AX 数据暂存于 BX
    mov dl,'-'
    mov ah,2
    int 21h
    mov ax,bx
    neg ax ;数据求补(绝对值)
write2: mov bx,10 ;10 压入堆栈,作为退出标志
    push bx
```



```
write2: mov bx,10  
        push bx                ;10 压入堆栈,作为退出标志  
write3: cmp ax,0                ;数据(商)为零,转向显示  
        jz write4  
        sub dx,dx               ;扩展被除数 DX. AX  
        div bx                  ;数据除以 10:DX. AX ÷ 10  
        add dl,30h              ;余数(0~9)转换为 ASCII 码  
        push dx                 ;数据各位先低位后高位压入堆栈  
        jmp write3  
  
write4: pop dx                  ;数据各位先高位后低位弹出堆栈  
        cmp dl,10               ;是结束标志 10,则退出  
        je write5  
        mov ah,2                ;进行显示  
        int 21h  
        jmp write4  
write5: pop dx  
        pop bx  
        pop ax  
        ret                     ;子程序返回  
write   endp
```


C语言传入、传出参数约定

♥ 返回值：通过寄存器返回

- ❖ $AL \leftarrow \text{char}$
- ❖ $AX \leftarrow \text{short}$
- ❖ $EAX \leftarrow \text{int}$
- ❖ $(EDX, EAX) \leftarrow \text{_int64}$
- ❖ $EAX \leftarrow \text{指针}$

♥ 传入参数：通过堆栈

- ❖ 由右向左