子程序 (Subroutine)

引言

- **call和ret 指令都是转移指令，都修改IP或同时修改CS和IP。**
- 它们经常被共同用来实现子程序的设计。
- 参考教材第**98**页

CALL	**调用**
RET	**返回**

call 指令

- CPU执行call指令，进行两步操作：
  - （1）将当前的 IP 或 CS和IP 压入栈中
  - （2）转移
- call 指令不能实现短转移，除此之外，call指令实现转移的方法和 jmp 指令的原理相同

Call指令的几种格式

段内直接调用

- 原理：依据位移进行转移的call指令
- 格式：call 标号
- 执行的操作：(1) sp←sp-2

  (2) 将CALL的下一条指令的IP入栈

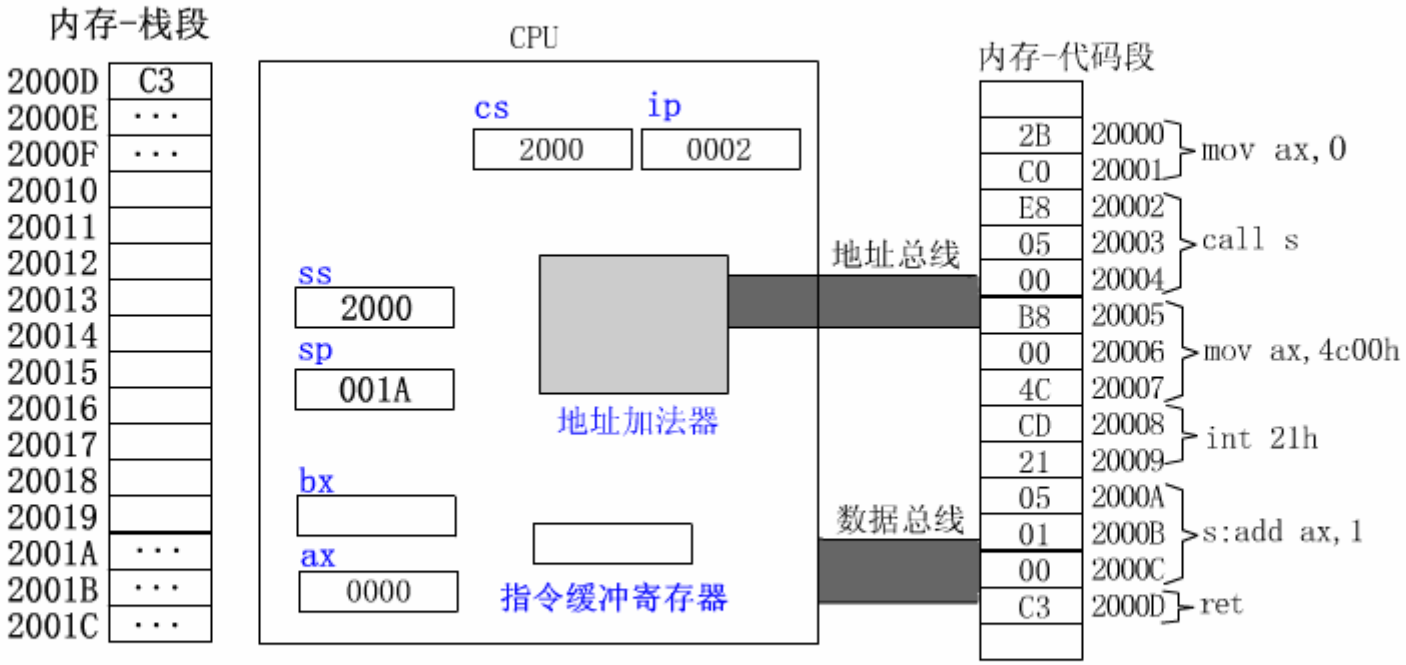
  (3) 将子程序名标号代表的偏移地址→IP

  即：将当前的 IP 压栈后，转到标号处执行指令
- CPU 执行指令“call 标号”时，相当于进行：

  push IP

  jmp near ptr 标号
- 演示

依据位移进行转移的call指令



依据位移进行转移的call指令

play step step stop

段内间接调用

- **call word ptr 内存单元地址**
  - 将调用点的IP内容入栈保护
  - 从内存地址字单元中取值送IP
  - 汇编语法解释：

    push IP

    jmp word ptr 内存单元地址

示例：CALL BX

- **call 16位寄存器**
  - 将IP的内容入栈保护
  - 从16位寄存器中取值送IP
  - 汇编语法解释：

    push IP

    jmp 16位寄存器

示例：
CALL WORD PTR [BX+SI]

- **call far ptr 标号**
- 实现段间转移
- **call**指令的操作：
  - 将**call**的下一条指令的**CS**及**IP**值入栈保护
  - 将标号所在段的段地址→**CS**
  - 将标号所在段的偏移地址→ **IP**
- 相当于进行：

  push CS

  push IP

  jmp far ptr 标号

主程序与子程序不在同一个代码段中——远程调用

例：

```
CODE1  SEGMENT
  MAIN  PROC  FAR
          ⋮
        CALL  SUBRX
          ⋮
        MOV AH，4CH
        INT  21H
  MAIN  ENDP
CODE1  ENDS
```

```
CODE2  SEGMENT
          ⋮
       CALL  SUBRX
          ⋮
    SUBRX   PROC   FAR
          ⋮
         RET
    SUBRX  ENDP

CODE2  ENDS
```

- **call dword ptr 内存单元地址** 段间间接调用
  - 将call的下一条指令的CS及IP的内容入栈保护
  - 从内存地址双字单元中取第一个字值送IP

    即：**(EA)→IP**
  - 第二个字值送CS，即： **(EA+2)→CS**
  - 汇编语法解释：

    push CS

    push IP

    jmp dword ptr 内存单元地址

子程序名保存在双字单元中：
第一个字作为偏移地址
第二个字作为段地址

转移地址在内存中

```
mov sp,10h
mov ax,0123h
mov ds:[0],ax
call word ptr ds:[0]
```

(IP)=? 0123H

(sp)=? 0EH

```
mov sp,10h
mov ax,0123h
mov ds:[0],ax
mov word ptr
ds:[2],0
call dword ptr ds:[0]
```

(CS)=? 0

(IP)=? 0123H

(sp)=? 0CH

ret指令---段内返回

- 返回指令：通常作为一个子程序的最后一条指令用来返回高一层程序
- 执行时从栈顶弹出返回地址：用栈中的数据，修改IP的内容，实现近转移
- **CPU执行ret指令时，进行下面两步操作：**
  - **（1）(IP)=((ss)*16+(sp))**
  - **（2）(sp)=(sp)+2**

ret指令---段间返回

- 也可以用retf指令
- 用栈中的数据，修改CS和IP的内容，实现远转移，即：
- 从堆栈中弹出断点的偏移地址→IP
- 弹出断点的段地址→CS
- CPU执行retf指令时，进行下面两步操作：
  - （1）(IP)=((ss)*16+(sp))
  - （2）(sp)=(sp)+2
  - （3）(CS)=((ss)*16+(sp))
  - （4）(sp)=(sp)+2

ret 、retf、ret n

- 用汇编语法来解释ret和retf指令，则：
  - CPU执行ret指令时，相当于进行：

    pop IP
  - CPU执行retf指令时，相当于进行：

    pop IP

    pop CS
- ret n
  - 弹出段点后，再将堆栈指针SP+n之后返回
  - 例，在子程序返回时，将堆栈指针加6后返回的指令：
  - RET 6

```
assume cs:codesg
stack segment
        db 16 dup (0)
stack ends
codesg segment
        mov ax,4c00h
        int 21h
start:  mov ax,stack
        mov ss,ax
        mov sp,16
        mov ax,0
        push ax
        mov bx,0
        ret
codesg ends
end start
```

ret指令执行后

(IP)=？ 0

CS:IP指向？代码段的第一条指令

```
assume cs:codesg
stack segment
        db 16 dup (0)
stack ends
codesg segment
        mov ax,4c00h
        int 21h
start:  mov ax,stack
        mov ss,ax
        mov sp,16
        mov ax,0
        push cs
        push ax
        mov bx,0
        retf
codesg ends
end start
```

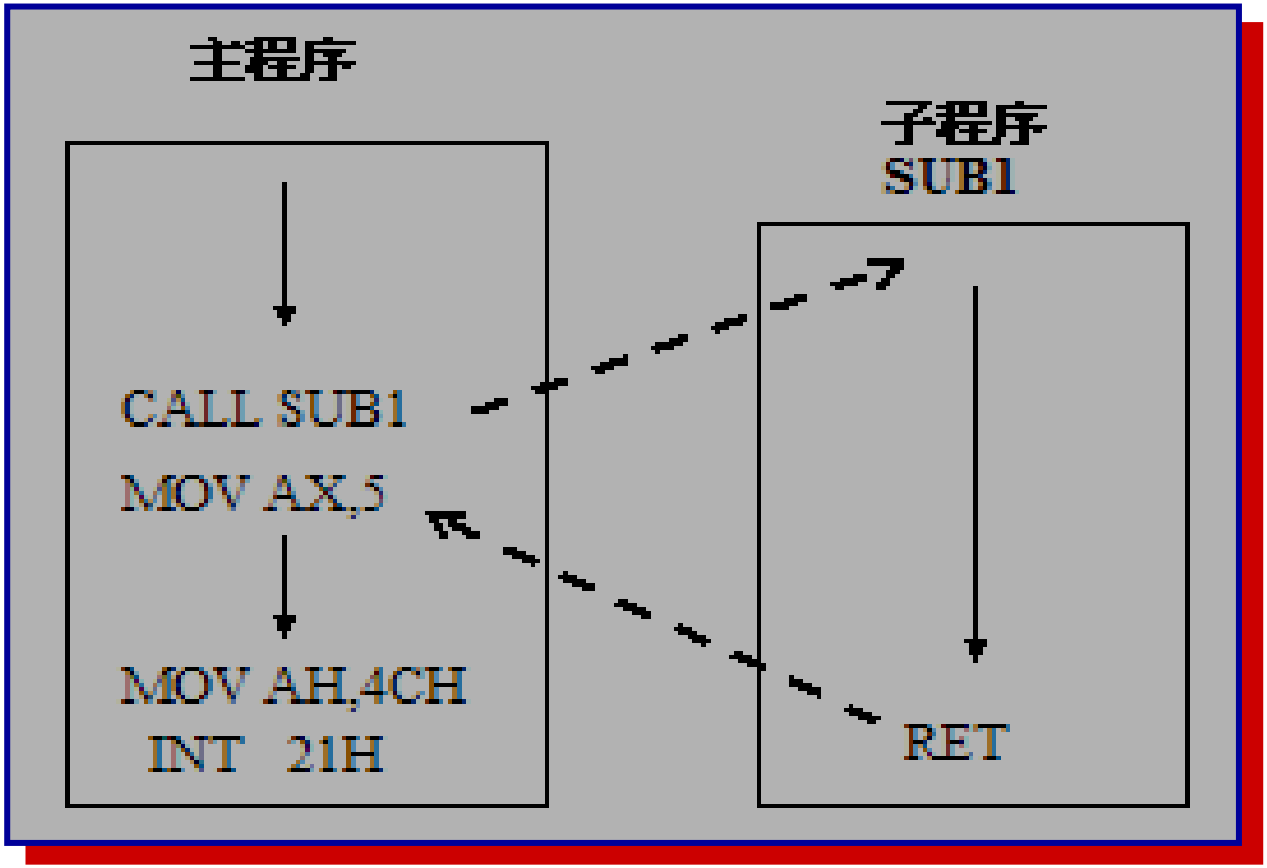
retf指令执行后

CS:IP指向？代码段的第一条指令

call 和 ret 的配合使用

- 利用**call**和**ret**来实现子程序的机制。
- 调用指令**call**执行时，先将其下一条指令的地址保存到栈中，然后转入调用子程序入口
- 必须在子程序中写一条**ret**指令，以便子程序执行完毕能顺利返回调用处

子程序与分支程序的最大区别是子程序执行完要返回到主程序，也就是返回到**CALL**指令的下一条继续执行。在子程序中用**RET**指令作为返回指令。主程序和子程序的关系如图所示。



过程定义

- 见教材第196页
- 过程定义伪指令格式：

```
子程序名   PROC 属性
          ……
子程序名   ENDP
```

- 说明：
  - PROC 和ENDP必须成对出现
  - 属性指子程序的类型属性，分为NEAR近程和FAR远程，隐含属性为NEAR型

过程定义示例

- 定义近程子程序**SUBR1**

```
SUBR1   PROC  NEAR
          ……
SUBR1   ENDP
```

- 定义远程子程序**SUBRX**

```
SUBRX   PROC  FAR
          ……
SUBRX   ENDP
```

现场保护

在进入子程序时，先要把某些寄存器保存起来，称为现场保护。在子程序返回主程序之前要恢复现场。

例：保护现场和恢复现场。

```
SUBT  PROC  NEAR
    PUSH  AX
    PUSH  BX
    PUSH  CX
    PUSH  DX
    MOV  AL，4
    MOV  BL，5
    IMUL  BL
    MOV  Y，AX
     ┆
    POP  DX
    POP  CX
    POP  BX
    POP  AX
    RET
SUBR1  ENDP
```

应用实例

- 设计一个子程序，可以根据提供的N，计算N的3次方
- 问题分析：
  - 将参数N存储在什么地方
  - 计算得到的数值，存储在什么地方
- 解决方法：
  - 可以用寄存器来存储，将参数放BX
  - 要计算N*N*N，可以使用多个mul指令，可将结果放到DX和AX中

参数和结果传递的问题

- 子程序：
  - 说明：计算N的3次方
  - 参数： (bx) = N
  - 结果： (dx:ax) = N^3

```
cube: mov ax, bx
      mul bx
      mul bx
      ret
```

寄存器传参

- 用寄存器传参是最常使用的方法
- 存放参数的寄存器和存放结果的寄存器，调用者和子程序的读写操作恰恰相反
- 调用者
  - 将参数送入参数寄存器
  - 从结果寄存器去返回值
- 子程序
  - 从参数寄存器取参数
  - 将返回值送结果寄存器

参数和结果传递的问题

- 编程：计算data段中第一组数据的 3 次方，结果保存在后面一组dword单元中。

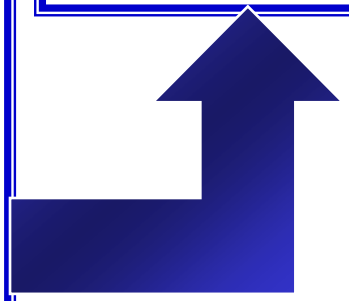
```
assume cs:code
data segment
    dw 1,2,3,4,5,6,7,8
    dd  0,0,0,0,0,0,0,0
data ends
```

- 使用已经写好的子程序
- 程序代码

```
cube:   mov ax, bx
        mul bx
        mul bx
        ret
```

```
assume cs:code
data segment
dw 1,2,3,4,5,6,7,8
dd 0,0,0,0,0,0,0,0
data ends

code segment

start: mov ax, data
       mov ds, ax
       mov si, 0    ;ds:si指向第一组word单元
       mov di, 16   ;ds:di指向第二组dword单元

       mov cx, 8
   s:  mov bx, [si]
       call cube
       mov [di], ax
       mov [di].2, dx
       add si, 2      ;ds:si指向下一个word单元
       add di, 4      ;ds:di指向下一个dword单元
       loop s
```

```
       mov ax,4c00h
       int 21h

cube: mov ax, bx
                mul bx
                mul bx
                ret

                code ends
                End start
```



存储单元传参

- 寄存器有限，当有多个参数需传递时，如何解决
- 将批量数据放在内存，将内存空间的首地址放在寄存器中，传递给需要的子程序
- 同样适用于结果返回方式
- 例：用子程序编程实现，将一个全是字母的字符串转换为大写

大小写转换问题

- **ASCII码：61H→a　　62H →b**
- 改变一个字母的大小写，就是改变它所对应的ASCII 码

A → 41H　　a → 61H

大写	二进制	小写	二进制
A	01000001	a	01100001
B	01000010	b	01100010
C	01000011	c	01100011
D	01000100	d	01100100

大小写转换的问题

- 对比：小写字母的ASCII码值比大写字母的ASCII码值大20H。

  因此，如果将 “a” 的ASCII码值减去20H，就可以得到“A”；如果将“A”的ASCII码值加上20H 就可以得到“a”

以字符形式给出的数据

```
assume cs;code,ds:data
data segment
   db 'unIX'
   db 'foRK'
data ends
code segment
    start: mov al, 'a'
           mov bl, 'b'
           mov ax, 4c00h
           int 21h
code ends
end start
```

= “db 75H,6EH,49H,58H”
u n I X

= “db 66H,6FH,52H,4BH”
f o R K

= “mov al,61H”

= “mov al,62H”

大小写转换的问题

问题：程序补全

将datasg中的第一个字符串转化为大写，第二个字符串转化为小写

```
assume cs;codesg,ds:datasg
datasg segment
   db ‘BaSiC'
   db ‘iNfOrMaTiOn'
datasg ends
codesg segment
    start:

                ?


codesg ends
end start
```

大小写转换的问题

- 分析：
  - 字符串**“BaSic”**，应只需对其中的小写字母所对应的**ASCII**码进行减**20H** 的处理，将其转为大写，其中的大写字母不变；
  - 字符串 **“ iNforMaTIOn ”**，只对其中的大写字母所对应的**ASCII**码进行加**20H** 的处理，将其转为小写，其中的小写字母不变
  - 程序必须能够判断一个字母是大写还是小写

7.4 大小写转换的问题

- 就ASCII码的二进制，除第5位（位数从0开始计算）外，大写字母和小写字母的其他各位都一样
- 大写字母ASCII码的第5位（位数从0开始计算）为0，小写字母的第5位为1
  - 将它的第5位置0，它就必将变为大写字母
  - 将它的第5位置1，它就必将变为小写字母

7.4 大小写转换的问题

- 用什么方法将一个数据中的某一位置0还是置1?

当然是用我们学过的or和and指令

```
assume cs:codesg,ds:datasg
datasg segment
db 'BaSiC'
db 'iNfOrMaTiOn'
datasg ends
codesg segment
start: mov ax,datasg
      mov ds,ax      ;设置ds指向datasg段
      mov bx,0       ;设置(bx)=0，ds:bx指向“BaSiC”的第一个字母
      mov cx,5                ;设置循环次数5，因为“BaSiC”的有5个字母
    s: mov al,[bx]          ;将ASCII码从ds:bx所指向的单元中取出
      and al,11011111b        ;将al中的ASCII码的第5位置为0，变为大写字母
      mov [bx],al           ;将转变后的ASCII码写回原单元
      inc bx                ;(bx)加1，ds:bx指向下一个字母
      loop s
      mov bx,5                ;设置(bx)=5，ds:bx指向“iNfOrMaTiOn”的第一个字母
      mov cx,11             ;设置循环次数11，因为“iNfOrMaTiOn”的有11个字母
   s0: mov al,[bx]
      or al,00100000b         ;将al中的ASCII码的第5位置为0，变为小写字母
      mov [bx],al
      inc bx
      loop s0
      mov ax,4c00h
      int 21h
codesg ends
end start
```

编程：将data段中的字符串转化为大写

```
capital: and byte ptr [si], 11011111b
                ;将ds:si所指单元中的字母转化为大写
        inc si    ;ds:si指向下一个单元
        loop capital
        ret
```

```
assume cs:code
        data segment
            db 'conversation'
        data ends
        code segment
          start: ……
        code ends
        end start
```

批量数据的传递

- 因为字符串中的字母可能很多，所以我们不便将整个字符串中的所有字母都直接传递给子程序
- 子程序需要知道两件事
  - 字符串的内容
  - 字符串的长度
- 将字符串在内存中的首地址放在寄存器中传递给子程序
- 用 **loop** 指令循环，循环的次数是字符串的长度，将字符串的长度放到**cx**中

```
assume cs:c ode
data segment
  db 'conversation'
data ends
code segment
start:    mov ax, data
          mov ds, ax
          mov si, 0        ;ds: si指向字符串（批量数据）所在空间的首地址
          mov cx, 12               ;cx存放字符串的长度
          call capital
          mov ax, 4c00h
          int 21h
capital:and byte ptr [si],11011111b
          inc si
          loop capital
          ret
code ends
end start
```

堆栈传参

堆栈是一种特殊的存储结构，利用**PUSH**入栈和**POP**出栈指令，可以方便地保存和读取数据。

一个例子

示例：十进制与十六进制转换。将键盘输入的一个两位十进制数以十六进制形式显示在屏幕上。可多次输入直到按下ESC键。

运行结果：

```
C:\hb>7-1

input dec=33
out HEX=0021
input dec=3
out HEX=0003
input dec=21
out HEX=0015
input dec=100
out HEX=0064
input dec=666
out HEX=029A
input dec=65535
out HEX=FFFF
input dec=←
C:\hb>
```

分析

- 用DOS的1号功能输入一个两位数，一回车结束
- 将输入的数字减去30H保存在X单元，第1个数字扩大10倍再与第2个数字相加，变为两位十进制数
- 用9号功能显示提示信息
- 将十进制数除以16，形成十六进制数
- 再将十六进制数转换为ASCII码，用2号功能显示

```
;多次输入一个两位十进制数并以十六进制显示出来。以ESC键结束

data segment
  x db 2 dup(?)
  mess1 db 0dh,0ah,'decimal=$'
  mess2 db 0dh,0ah,'HEX=$'
data ends
code segment
  assume cs:code,ds:data
start:
  mov ax,data
  mov ds,ax
let0:
  mov x,0
  mov x+1,0
  mov si,0
  mov dx,offset mess1    ;显示提示1
  mov ah,9
```

设计思路：

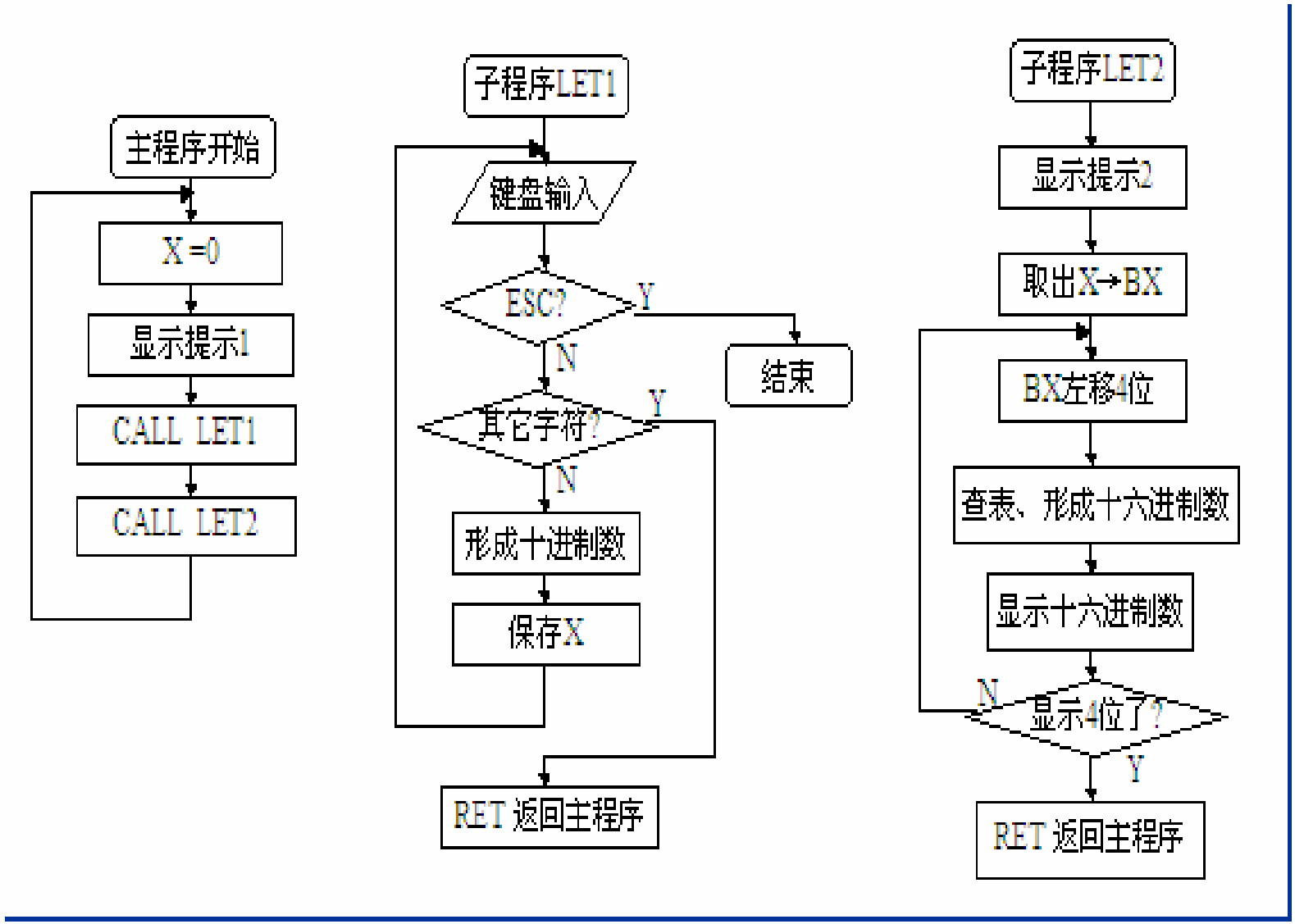
1.主程序标号LET0，一个子程序标号为LET1，另一个子程序标号LET2；

2.主程序是一个死循环，只有当按下ESC键时才能退出、结束程序；

3.子程序LET1功能是键盘输入，并把输入的数字变为十进制数X；

4.子程序LET2功能是通过查表将十进制数X变为十六进制，并显示出来。

程序框图：


主程序开始
X=0
显示提示1
CALL LET1
CALL LET2
子程序LET1
键盘输入
ESC?
Y
N
结束
其它字符?
Y
N
形成十进制数
保存X
RET 返回主程序
子程序LET2
显示提示2
取出X→BX
BX左移4位
查表、形成十六进制数
显示十六进制数
N
显示4位了?
Y
RET 返回主程序

程序如下：

```
; 十进制数并以十六进制显示出来。按ESC键结束。
data segment
    x dw 0
    mess1 db 0dh,0ah,'input dec=$'
    mess2 db 0dh,0ah,'out hex=$'
    hex db '0123456789ABCDEF'
data ends
code segment
assume cs:code,ds:data
start:
mov ax,data
mov ds,ax
```

```
;主程序
let0:
mov x,0
mov dx,offset mess1                    ;显示提示1
mov ah,9
int 21h
call let1
call let2
jmp let0
```

```
;子程序1：键盘输入、形成十进制
let1:
    mov ah,1                    ;键盘输入十进制数
    int 21h
    cmp al,27                   ;是ESC键?
    jz out1
    sub al,30h                  ;其它字符?
    jl exit                     ;是，转exit
    cmp al,9
    jg exit
    mov ah,0
    xchg ax,x                   ;形成十进制数
    mov cx,10
    mul cx
    xchg ax,x
    add x,ax                    ;保存
    jmp let1
exit:ret
```

```
;子程序2：查表，显示十六进制
let2:
    mov dx,offset mess2     ;显示提示2
    mov ah,9
    int 21h
    mov bx,x
    mov ch,4
    mov cl,4
rept1:
    rol bx,cl               ;0031→0310→3100→1003→0031
    mov al,bl
    and ax,000fh                    ;保留最低4位
    mov si,ax
    mov dl,hex[si]          ;查表显示高位、低位
    mov ah,2
    int 21h
    dec ch
    jnz rept1
    ret
```

```
out1:
mov ah,4ch
int 21h
code ends
end start
```