汇编语言程序设计

Assembly Language Programming

主讲：徐娟

计算机与信息学院　计算机系 分布式控制研究所

E-mail: xujuan@hfut.edu.cn,

Mobile: 18055100485

第二章　8086指令系统

- 数据传送指令
- 算术运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- CPU控制指令

数据传送指令

- 通用数据传送指令
- 堆栈操作指令
- 标志传送指令
- 地址传送指令

算术运算指令

- 加法指令
- 减法指令
- 乘法指令
- 除法指令
- 十进制/BCD码调整指令

2.3 位操作类指令

- 位操作类指令以二进制位为基本单位进行数据的操作
- 这是一类常用的指令，都应该掌握
- 注意这些指令对标志位的影响

1、逻辑运算指令

AND OR XOR NOT TEST

2、移位指令

SHL SHR SAR

3、循环移位指令

ROL ROR RCL RCR

逻辑运算指令

逻辑与指令：AND DST, SRC
执行操作：(DST) ← (DST) ∧ (SRC)
用途：用于屏蔽一个数的某些位。

逻辑或指令：OR DST, SRC
执行操作：(DST) ← (DST) ∨ (SRC)
用途：用于置位一个数的某些位。

异或指令：XOR DST, SRC
执行操作：(DST) ← (DST) ∀ (SRC)
用途：将一个数的某些位取反。

测试指令：TEST OPR1, OPR2
执行操作：(OPR1) ∧ (OPR2)
用途：用于测试一个数的某些位。

CF OF SF ZF PF AF

0 0 * * * 无定义

根据运算结果设置

AND VS TEST

- AND具有破坏性，TEST没有
  - AL = 0FFH
  - AND AL, 0
  - TEST AL, 0
- 同SUB和CMP

逻辑非指令：NOT OPR
执行操作：(OPR) ← ¬(OPR)
功能：按位取反

* OPR不能为立即数
* 不影响标志位

例子

例：屏蔽AL的0、1两位

```
AND  AL, 0FCH
```

```
      * * * * * * * *
AND   1 1 1 1 1 1 0 0
      * * * * * * 0 0
```

例：置AL的第5位为1

```
OR  AL, 20H
```

```
      * * * * * * * *
OR    0 0 1 0 0 0 0 0
      * * 1 * * * * *
```

例：使AL的0、1位变反

```
XOR  AL, 3
```

```
      * * * * * * * *
XOR   0 0 0 0 0 0 1 1
      * * * * * * * *
```

例：测试某些位是0是1

```
TEST  AL, 1
JZ  EVEN    ;最低位为0，转移到even，jump if zero
```

例题：逻辑运算

```
mov al,45h          ; 逻辑与 al=01h
and al,31h          ; CF=OF=0，SF=0、ZF=0、PF=0
-------------------------------------------------
mov al,45h          ; 逻辑或 al=75h
or al,31h           ; CF=OF=0，SF=0、ZF=0、PF=0
-------------------------------------------------
mov al,45h          ; 逻辑异或 al=74h
xor al,31h          ; CF=OF=0，SF=0、ZF=0、PF=1
-------------------------------------------------
mov al,45h          ; 逻辑非 al=0bah
not al              ; 标志不变
```

移位指令（shift）

分类：

- 逻辑SHL/SHR
- 算术SAL/SAR （Shift Arithmetic）
- 循环ROL/ROR （Rotate）
- 带进位循环RCL/RCR （Rotate carry）

共同特点

- 都是按位进行
- 当移动的位数为一位时，用立即数1；
- 当移动二位或二位以上时，要预先将移动的位数存放在CL中。
- SHL AL, 2 →MOV CL,2 ; SHL AL,CL;

移位指令对标志的影响

- 按照移入的位设置进位标志CF
- 根据移位后的结果影响SF、ZF、PF
- AF：没有定义
- 如果进行一位移动，则按照操作数的最高符号位是否改变，相应设置溢出标志OF：

  操作数最高位有变化，则OF = 1；否则OF = 0。

  当移位次数大于1时，OF不确定

逻辑SHL/SHR（Shift）

- **格式：SHL Dst,Src**

功能：将Dst的内容左移1～n位，右边添0。

说明：Dst—reg，mem；Src—1或CL

注意：逻辑左移1位等价于将一个无符号数乘以2。

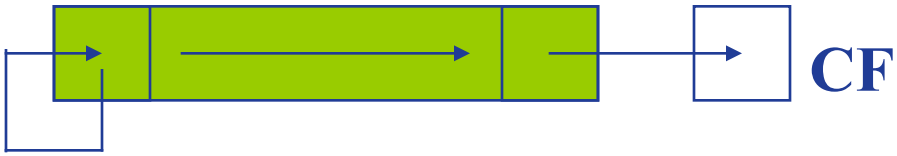
- **格式：SHR Dst,Src**

功能：将Dst的内容右移1～n位，左边添0。

注意：逻辑右移一位等价于将一个无符号数除以2(整除)

算术SAL/SAR （ shift arithmetic）

- 算术左移　SAL　（同逻辑左移SHL）
- 算术右移　格式：SAR Dst,Src
  - 功能：将Dst的内容右移1～n位，最高位不变

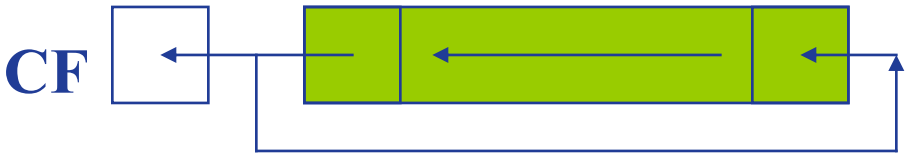


  - 算术右移一位等价于将一个带符号数除以2（整除）
  - 注意：当操作数为负数且最低位有1移出时，SAR与IDIV结果不同，如-1右移后为-1，不为0

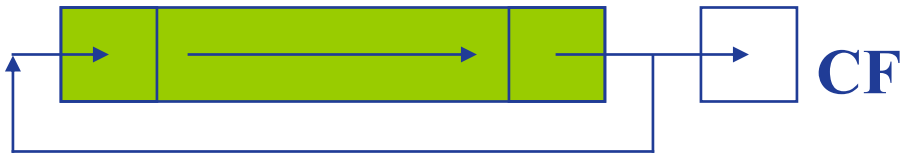
循环移位指令(rotate)

不带进位

- 循环左移　ROL　Dst,Src

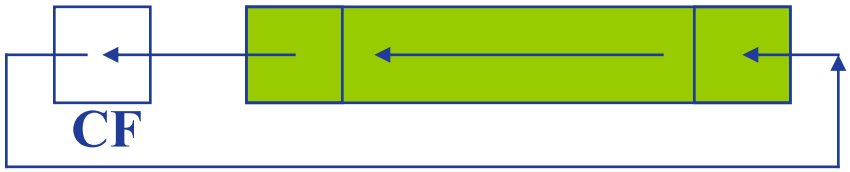


- 循环右移　ROR　Dst,Src

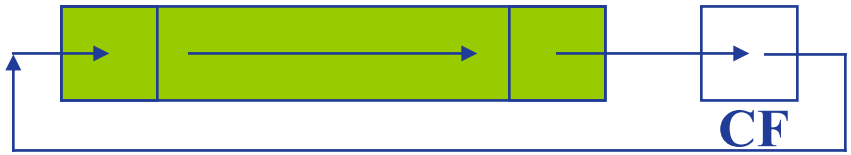


带进位循环RCL/RCR

- 带进位循环左移　RCL　Dst,Src



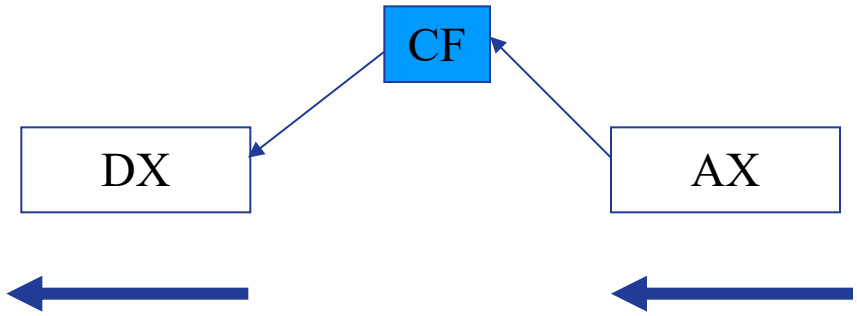
- 带进位循环右移　RCR　Dst,Src



例子（一）

- **RCL和RCR常用在多字节数的移位。**
  - 在DX和AX中存放着一个32位数据，试将其左移1位。
  - SHL AX,1
  - RCL DX,1

  - 右移如何处理？



例子（二）

$D_7/D_6/D_5/D_4/D_3/D_2/D_1/D_0$

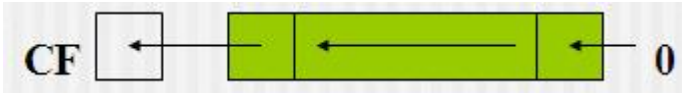
- 把(BL)中的8位数高低4位互换
  - MOV DL,BL　　；DL=BL
  - MOV CL,4　　；
  - SHR BL, CL　　；BL右移4位（0/0/0/0/ D7/D6/D5/D4）
  - SHL DL, CL　　；DL左移4位（D3/D2/D1/D0/ 0/0/0/0）
  - OR BL, DL

  - MOV CL,4
  - ROL/ROR BL, CL

移位指令

SHL/SHR

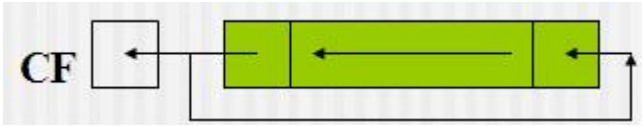
最高位补0，最低位进入CF



SAL/SAR

最高位不变，最低位进入CF

ROL/ROR



RCL/RCR

移位指令的用途：运算，测试某些位，遍历字节位，字节数据变形

IBM PC机的指令系统

- 80x86指令系统分成下列六大类：
  - 数据传送指令
  - 算术运算指令
  - 逻辑运算和移位指令
  - **控制转移指令**
  - 串操作指令
  - CPU控制指令

控制转移指令

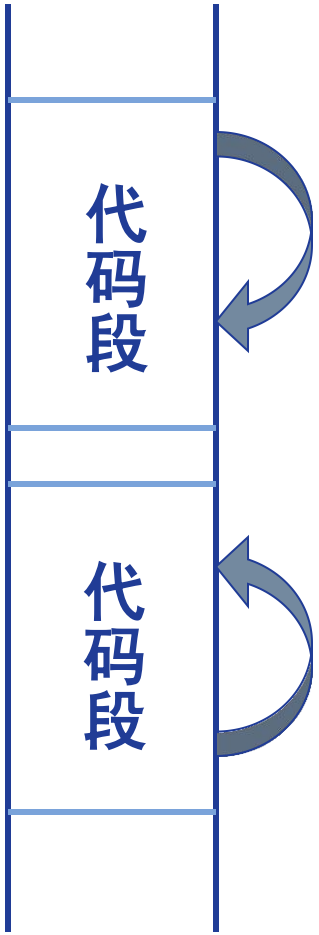
- 无条件转移指令
- 条件转移指令
- 循环指令
- 子程序调用和返回指令
- 中断指令

无条件转移指令(jump)

- **格式：JMP　地址表达式**
- **功能：使程序的流程无条件跳到转移地址所指的地方。**
  - 转移目的地址＝(CS)×16+(IP)
  - 段内转移：改变IP的内容，CS的内容不变。
  - 段间转移：IP、CS的内容都改变。

目标地址的范围：段内

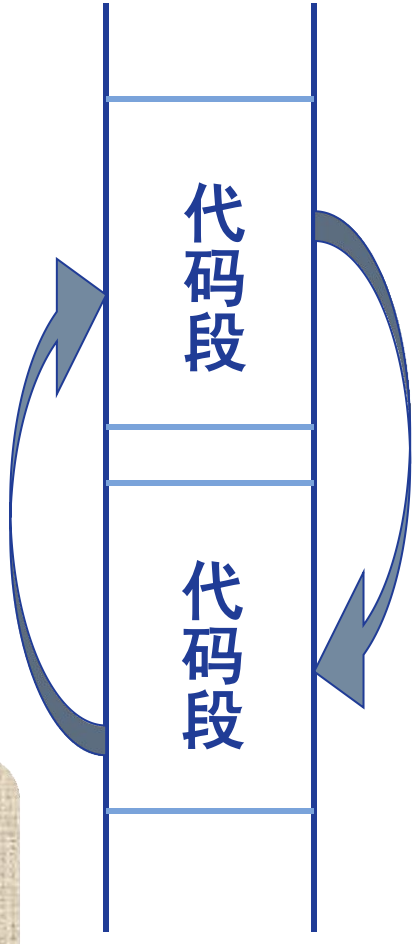
- **段内转移——近转移（near）**
  - 在当前代码段2^{16}=64KB范围内转移（ ±32KB范围）
  - 不需要更改CS段地址，只要改变IP偏移地址
- **段内转移——短转移（short）**
  - 转移范围可以用一个字节表达，在段内2^8=－128～＋127范围的转移



目标地址的范围：段间

❖ 段间转移——远转移（far）

- 从当前代码段跳转到另一个代码段，可以在1MB范围
- 更改CS段地址和IP偏移地址
- 目标地址必须用一个32位数表达，叫做32位远指针，它就是逻辑地址



动画

- 实际编程时，MASM汇编程序会根据目标地址的距离，自动处理成短转移、近转移或远转移
- 程序员可用操作符short、near ptr 或far ptr 强制

段内直接转移——相对寻址

用标号表达

JMP label ; IP←IP+位移量

- 位移量是紧接着JMP指令后的那条指令的偏移地址，到目标指令的偏移地址的地址位移
- 向地址增大方向转移时，位移量为正；向地址减小方向转移时，位移量为负

```
again:    dec cx              ; 标号again的指令
          ……
          jmp again           ; 转移到again处继续执行
          ……
          jmp output          ; 转向output
          ……
output:   mov result,al       ; 标号output的指令
```

段内间接转移——间接寻址

- **段内间接转移**(间接寻址)：指定某个**寄存器**的内容或**某个内存字单元的内容**作为转移地址的偏移地址。
  - 例如：JMP BX ; (BX) → IP
  - JMP WORD PTR[1000H] ; (DS:1000H) → IP
  - JMP WORD PTR[SI+2] ; (DS:SI+2) → IP
  - JMP TABLE[BX] ; (DS:TABLE+(BX)) → IP

段间转移

- **段间直接转移(直接寻址)：通过标号直接给出转移地址**
  - JMP far ptr NEXTP1　　；NEXTP1的段址→ CS，偏址→ IP
- **段间间接转移(间接寻址)：**

  指定一个4字节的单元内容作为转移地址，

  低二字节内容→IP，高二字节内容→CS。

```
mov word ptr [bx],0
mov word ptr [bx+2],1500h
JMP far ptr [bx]      ；转移到1500h:0
```

条件转移指令

- 标志位条件转移指令
- 二个无符号数比较转移指令
- 二个带符号数比较转移指令

标志位条件转移指令

- **JC　标号 ；当(CF)=1，则转移。**
  - JNC　标号；当(CF)=0，则转移。
- **JZ/JE　标号；当(ZF)=1，则转移。(Equal)**
  - JNZ/JNE 标号；当(ZF)=0，则转移。
- **JS　标号；当(SF)=1，则转移。**
  - JNS 标号；当(SF)=0，则转移。
- **JO　标号；当(OF)=1，则转移。**
  - JNO 标号；当(OF)=0，则转移。
- **JP 标号；当(PF)=1，则转移。**
  - JNP 标号；当(PF)=0，则转移。

二个无符号数比较转移指令

- 设A为被减数，B为减数。

  CMP A,B

- JA　标号；当A>B时转移；(above)
- JAE 标号；当A≥B时转移；(above or equal)
- JB　标号；当A<B时转移；(below)
- JBE 标号；当A≤B时转移。(below or equal)

  利用CF：高低　ZF：相等

二个带符号数比较转移指令

- JG　标号；当被减数大转移；(greate)
- JGE 标号；当被减数大于等于减数转移；
- JL　标号；当被减数小转移；(little)
- JLE 标号；当被减数小于等于减数转移

SF OF ZF

转移条件cc：两数大小关系

P55

JB/JNAE	CF=1
	Jump if Below/Not Above or Equal
JNB/JAE	CF=0
	Jump if Not Below/Above or Equal
JBE/JNA	CF=1或ZF=1
	Jump if Below/Not Above
JNBE/JA	CF=0且ZF=0
	Jump if Not Below or Equal/Above
JL/JNGE	SF≠OF
	Jump if Less/Not Greater or Equal
JNL/JGE	SF=OF
	Jump if Not Less/Greater or Equal
JLE/JNG	ZF≠OF或ZF=1
	Jump if Less or Equal/Not Greater
JNLE/JG	SF=OF且ZF=0
	Jump if Not Less or Equal/Greater

记录BX中“1”的个数

```
        xor al,al          ; AL＝0，CF＝0
again:  cmp bx,0
        jz next
        shl bx,1           ; 也可使用 shr bx,1
        adc al,0
        jmp again
next:   ...                ; AL保存1的个数
```

例子1

- 完成分段函数

$$AH = \begin{cases} -1 & AL < 0 \\ 0 & AL = 0 \\ 1 & AL > 0 \end{cases}$$

2.4.3 循环指令（loop）

```
JCXZ label    ；CX＝0，转移到标号label
```

```
LOOP label    ；CX←CX－1，
    ；CX≠0，循环到标号label
```

```
LOOPZ label   ；CX←CX－1，
    ；CX≠0且ZF＝1，循环到标号label
```

```
LOOPNZ label ；CX←CX－1，
    ；CX≠0且ZF＝0，循环到标号label
```

- 循环指令默认利用CX计数器
- label操作数采用相对短转移寻址方式

Notice!

- 除无条件转移指令，其他指令只能使用标号；
- 只能是段内直接短转移，即偏移量为-128~127；
- LOOP指令三个要点：
  1. CX=循环次数
  2. LOOP标号放在前面；
  3. 循环程序段写在标号和loop之间
- 使用LOOP指令，注意初始值是否为0。

例2.45：记录空格个数

cmp cx,0
jz next

```
        mov cx,count ; 设置循环次数
        mov si,offset string
        xor bx,bx    ; bx=0，记录空格数
        jcxz done
        mov al,20h   ; ASCII码 20h=空格
again:  cmp al,es:[si]
        jnz next     ; ZF=0，非空格，转移
        inc bx       ; ZF=1是空格，个数加1
next:   inc si
        loop again
        ; 字符个数减1，不为0继续循环
done:    mov result,bx;
```

dec cx
jnz again

习题

记录BX中1的个数

```
        xor al,al          ; AL＝0，CF＝0
again:  test bx,0          ; 等价于 cmp bx,0
        je next            ; ZF=1
        shl bx,1
        jnc again          ;CF=0
        inc al
        jmp again
next:   ...                ; AL保存1的个数
```

记录BX中1的个数

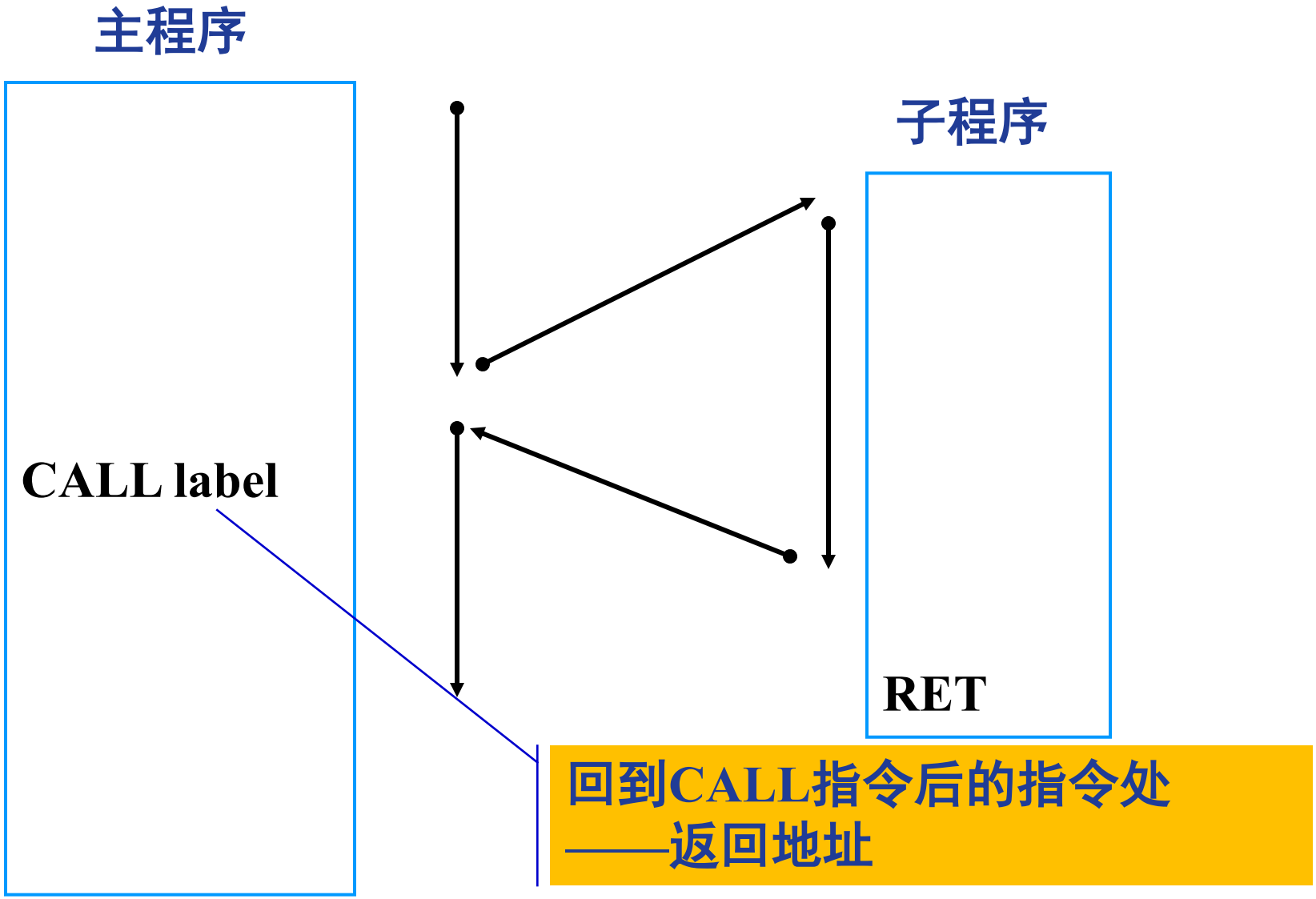
```
        xor al,al       ; AL＝0，CF＝0
again:  cmp bx,0
        jz next
        shl bx,1        ; 也可使用 shr bx,1
        adc al,0
        jmp again
next:   ...             ; AL保存1的个数
```

过程调用和返回指令

调用指令 CALL

- 格式：CALL 子程序/地址表达式
- 功能：
  - 保护断点——将当前断点压入堆栈；
  - 转入子程序——将子程序段的入口地址送入IP（/CS）；

主程序与子程序



子程序调用指令

- **CALL指令分成4种类型（类似JMP）**

```
CALL  label                ; 段内调用、相对寻址
CALL  r16/m16              ; 段内调用、间接寻址
CALL  far ptr  label       ; 段间调用、直接寻址
CALL  far ptr  mem         ; 段间调用、间接寻址
```

- **CALL指令需要保存返回地址：**
  - 段内调用——偏移地址IP入栈

    SP←SP－2，SS:[SP]←IP
  - 段间调用——偏移地址IP和段地址CS入栈

    SP←SP－2，SS:[SP]←CS

    SP←SP－2，SS:[SP]←IP

子程序返回指令

- 根据段内和段间、有无参数，分成4种类型

```
RET(RETN)        ；无参数段内返回
RET  i16         ；有参数段内返回
RET(RETF)        ；无参数段间返回
RET  i16         ；有参数段间返回
```

i16参数的作用

- 需要弹出CALL指令压入堆栈的返回地址
  - 段内返回——偏移地址IP出栈

    IP←SS:[SP]， SP←SP＋2
  - 段间返回——偏移地址IP和段地址CS出栈

    IP←SS:[SP]，SP←SP＋2

    CS←SS:[SP]，SP←SP＋2

10_依据位移进行转移的call指令_标清.flv

返回指令RET的参数

RET i16 ；有参数返回

- RET指令可以带有一个立即数i16，

  则堆栈指针SP将增加，即SP←SP+i16

- 这个特点使得程序可以方便地废除若干执行CALL指令以前入栈的参数

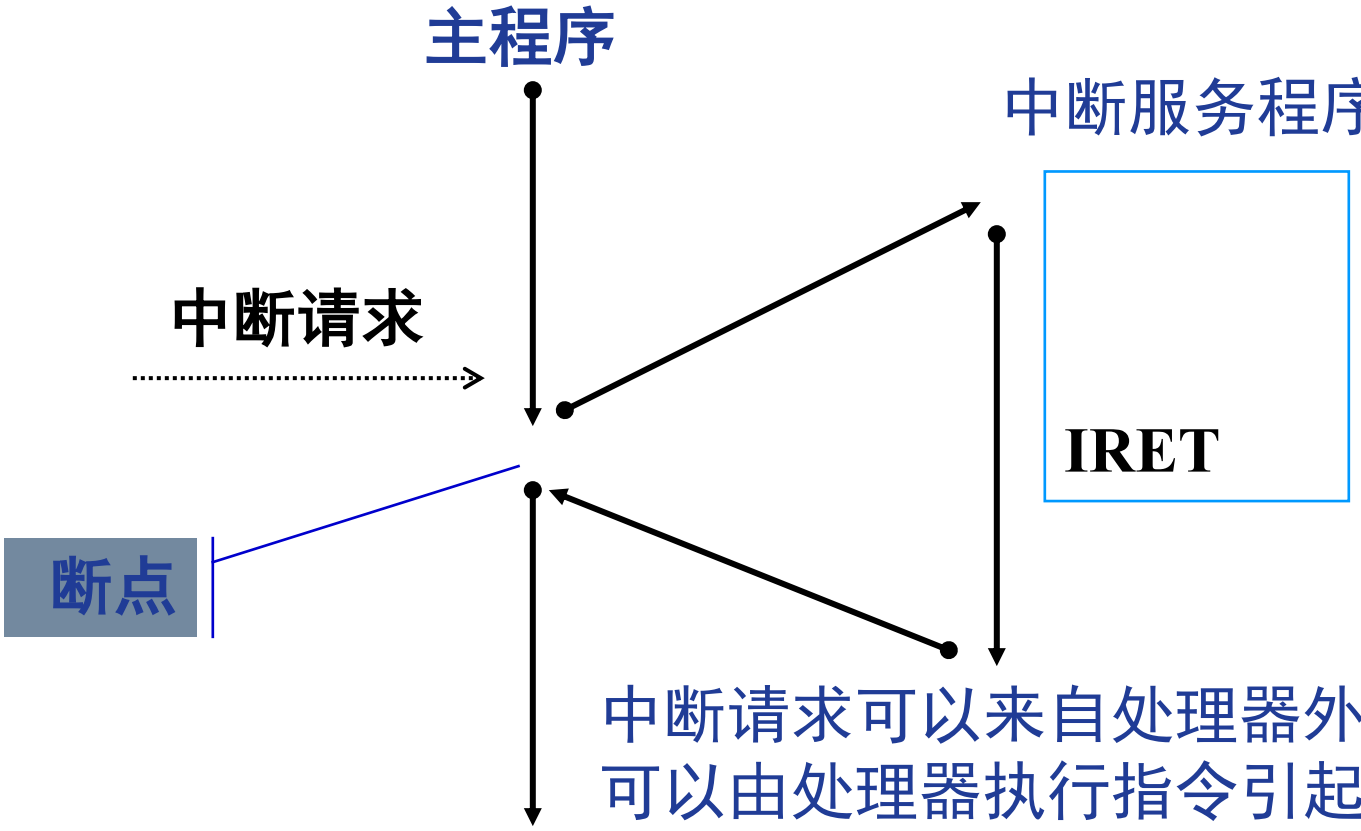
```
；主程序
        mov al,0fh          ；提供参数AL
        call htoasc         ；调用子程序
        ...
；子程序：将AL低4位的一位16进制数转换成ASCII码
htoasc: and al,0fh          ；只取al的低4位，0000 ****
        or al,30h           ；al高4位变成3，0011 ****
        cmp al,39h          ；是30～39，还是3Ah～3Fh
        jbe htoend
        add al,7            ；是3Ah～3Fh，加上7
htoend: ret                 ；子程序返回
```

P12 表1-4

中断相关概念

中断（Interrupt ）是又一种改变程序执行顺序的方法

计算机暂停现行程序的运行，转去执行另一程序以处理发生的事件，处理完毕后又自动返回原来的程序继续运行



中断请求可以来自处理器外部的中断源，也可以由处理器执行指令引起：如INT i8指令。

中断相关概念

- 中断：数据传输方式；软中断和硬中断
- 8086可以管理256个中断，各种中断用一个向量编号来区别
- 中断服务程序：处理中断的特殊的子程序，放在内存中；
- 中断向量表：存放中断子程序的入口地址，4字节对应一个中断，低位→中断服务程序IP，高位→中断服务程序CS

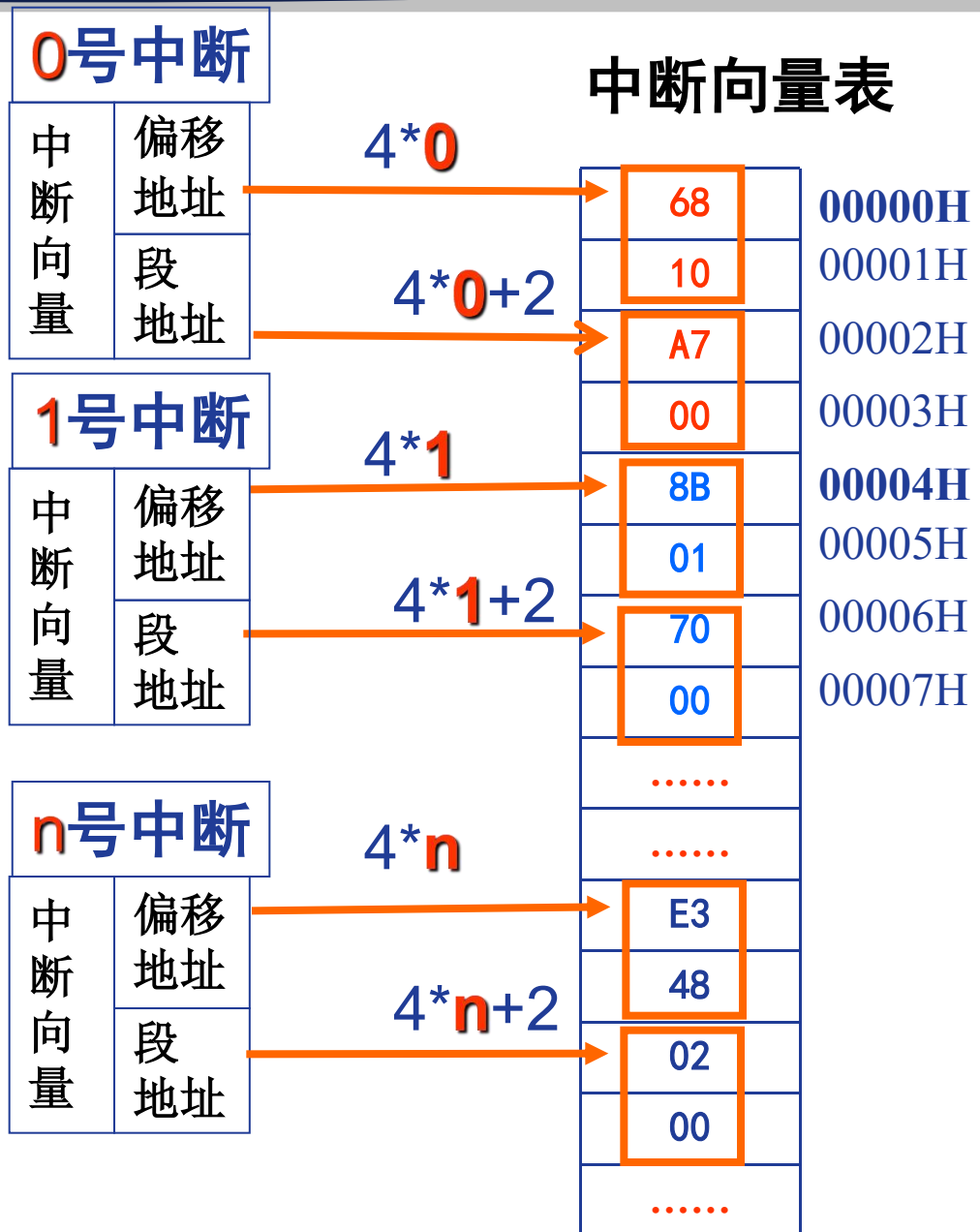
  256个，00000H-003FFH，1KB
- 中断类型码：给中断向量的一个编号
- （中断向量表查看：debug）

8086的中断

- 外部中断——来自CPU之外的原因引起的中断，分成
  - 可屏蔽中断：可由CPU的中断允许标志IF控制
  - 非屏蔽中断：不受CPU的中断允许标志IF控制

- 内部中断——CPU内部执行程序引起的中断，分成：
  - 除法错中断：执行除法指令，结果溢出产生的 0 号中断
  - 指令中断：执行中断调用指令INT n
  - 断点中断：用于断点调试（INT 3）的 3 号中断
  - 溢出中断：执行溢出中断指令，OF＝1产生的 4 号中断
  - 单步中断：TF＝1在每条指令执行后产生的 1 号中断

中断相关概念


0号中断
中断向量
偏移地址
段地址
1号中断
中断向量
偏移地址
段地址
n号中断
中断向量
偏移地址
段地址
中断向量表
4*0
4*0+2
4*1
4*1+2
4*n
4*n+2
68
10
A7
00
8B
01
70
00
.....
.....
E3
48
02
00
.....
00000H
00001H
00002H
00003H
00004H
00005H
00006H
00007H

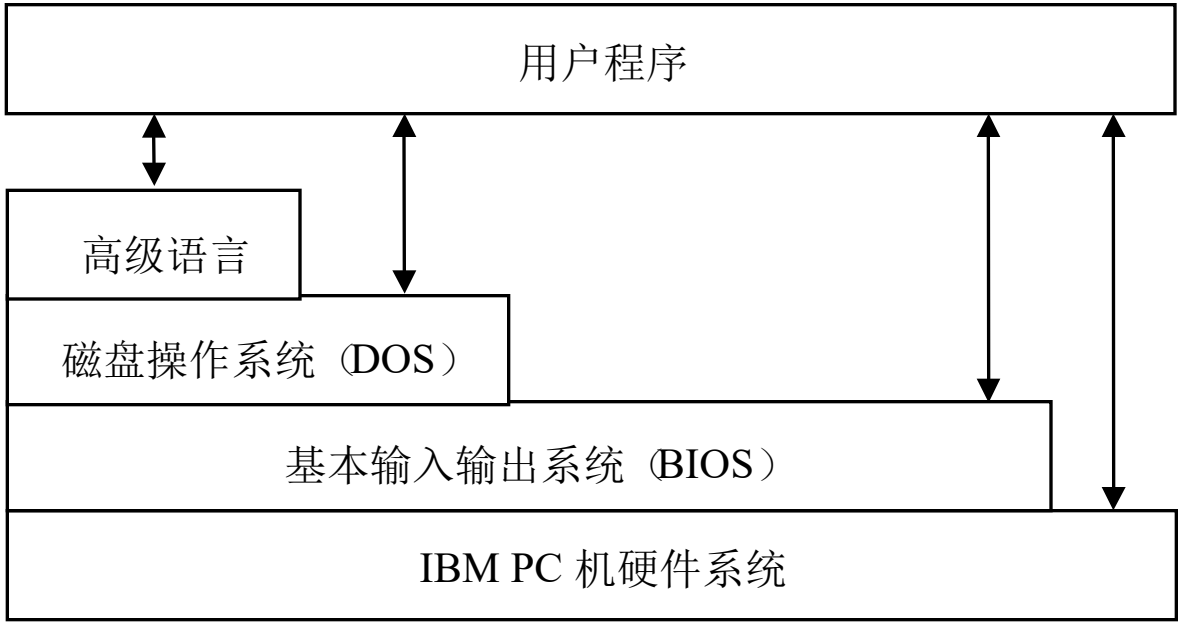
中断相关概念

```
C:\>debug
-d 0000:0000
0000:0000  68 10 A7 00 8B 01 70 00-16 00 A9 03 8B 01 70 00
0000:0010  8B 01 70 00 B9 06 12 02-40 07 12 02 FF 03 12 02
0000:0020  46 07 12 02 0A 04 12 02-3A 00 A9 03 54 00 A9 03
0000:0030  6E 00 A9 03 88 00 A9 03-A2 00 A9 03 FF 03 12 02
0000:0040  A9 08 12 02 A4 09 12 02-AA 09 12 02 5D 04 12 02
0000:0050  B0 09 12 02 0D 02 E1 02-C4 09 12 02 8B 05 12 02
0000:0060  0E 0C 12 02 14 0C 12 02-1F 0C 12 02 AD 06 12 02
0000:0070  AD 06 12 02 A4 F0 00 F0-37 05 12 02 18 3B 00 C0
0000:0080  72 10 A7 00 7C 10 A7 00-4F 03 E2 05 8A 03 E2 05
0000:0090  17 03 E2 05 86 10 A7 00-90 10 A7 00 9A 10 A7 00
0000:00A0  B8 10 A7 00 54 02 70 00-F2 04 47 D7 B8 10 A7 00
0000:00B0  B8 10 A7 00 B8 10 A7 00-40 01 27 04 50 09 7E DF
0000:00C0  EA AE 10 A7 00 E8 00 F0-B8 10 A7 00 A6 24 02 D4
0000:00D0  B8 10 A7 00 B8 10 A7 00-B8 10 A7 00 B8 10 A7 00
.........
```

- 中断调用指令 INT
  - 格式：INT n
  - 功能：调用n号中断子程序
  - 操作：
    - PUSHF； PUSH CS； PUSH IP
    - 取得中断向量，转入
- 中断返回指令 IRET
  - 格式：IRET
  - 操作：POP IP； POP CS； POPF

DOS 中断调用

- MS-DOS “API” & System “API”
- AH为功能号
- DOS INT部分使用AL/AX作为返回值— 0-成功；1-失败



DOS系统功能调用

- 21H号中断是DOS提供给用户的用于调用系统功能的中断，它有近百个功能供用户选择使用，主要包括设备管理、目录管理和文件管理三个方面的功能
- ROM-BIOS也以中断服务程序的形式，向程序员提供系统的基本输入输出程序
- 汇编语言程序设计需要采用系统的各种功能程序

功能调用的步骤

通常按照如下4个步骤进行：

(1) 在AH寄存器中设置系统功能调用号

(2) 在指定寄存器中设置入口参数

(3) 执行指令INT 21H（或ROM-BIOS的中断向量号）实现中断服务程序的功能调用

(4) 根据出口参数分析功能调用执行情况

P82

DOS 21H号中断调用

- **1. 从键盘读入一个字符**
  - MOV AH, 1/8H
    [回显/不回显]
  - INT 21H ;
  - 键入字符的ASCII存入AL中
- **2. 显示一个字符到屏幕**
  - MOV AH, 2H
  - MOV DL, ASCII
  - INT 21H ;

DOS 21H号中断调用

3. 显示一个字符串到屏幕

- MOV AH, 9H
- LEA DX, STRING
- INT 21H ;

;字符串要求以”$”结束,
;输出回车（0DH）和换行（0AH）字符产生回车和换行

```
string db 'Hello,Everybody !',0dh,0ah,'$ '
```

; 在数据段定义要显示的字符串

```
      ...
mov ah,09h                  ; 设置功能号：ah←09h
mov dx,offset string        ; 入口参数：dx←字符串的偏移地址
int 21h                     ; DOS功能调用：显示
```

DOS 21H号中断调用

4. 从键盘读入一个字符串到屏幕

MOV AH，0AH

LEA DX，STRING

INT 21H

STRING第一个字节为最多欲接收的字符长度；第二个为实际输入的长度；第三个是输入的字符串。

可执行全部标准键盘编辑命令；用户按回车键结束输入，如按Ctrl＋Break或Ctrl＋C则中止

5. 返回DOS

- MOV AH,4CH
- INT 21H

IBM PC机的指令系统

- 80x86指令系统分成下列六大类：
  - 数据传送指令
  - 算术运算指令
  - 逻辑运算和移位指令
  - 控制转移指令
  - 串操作指令
  - CPU控制指令

NOP

- 空操作 (机器码: 90H)
- 与XCHG AX,AX相同
- 用途:
  - Timer
    - 1个时钟周期; DSP, C51
  - Place Holder
    - 一个字节;

HLT

- 暂停指令
- 功能：
  - 使CPU进入暂停状态，直到系统复位或发生外部中断
  - 应用程序一般不使用

LOCK

- 封锁前缀
- 用途：
  - 用于多处理器系统，使当前处理器锁住总线，以保证当前指令为原子操作；
  - 当目的操作数为内存操作数时，为了完成“读-修改-写内存” 的操作不被打断；
- 示例：Lock add [bx], ax

第2章　总结

- 介绍了8086的16位指令系统的每条指令
- 希望大家进行一下整理（总结）：
  - 寻址方式
  - 指令支持的操作数形式
  - 指令对标志的影响
  - 常见编程问题
- 掌握常用指令
- 习题：2.1　2.6　2.8　2.10　　**2.20　2.22　2.24 (3) (6) (7)**