



# 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](mailto:xujuan@hfut.edu.cn),

Mobile: 18055100485

- 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:  $(DST) \leftarrow (DST) \wedge (SRC)$

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:  $(DST) \leftarrow (DST) \vee (SRC)$

用途: 用于置位一个数的某些位。

异或指令: **XOR DST, SRC**

执行操作:  $(DST) \leftarrow (DST) \vee (SRC)$

用途: 将一个数的某些位取反。

测试指令: **TEST OPR1, OPR2**

执行操作:  $(OPR1) \wedge (OPR2)$

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**

# 逻辑运算指令



逻辑非指令: NOT OPR

\* OPR不能为立即数

执行操作:  $(\text{OPR}) \leftarrow \neg (\text{OPR})$

\* 不影响标志位

功能: 按位取反



## ❖ AND具有破坏性，TEST没有

- `AL = 0FFH`
- `AND AL, 0`
- `TEST AL, 0`

## ❖ 同SUB和CMP

**TEST指令设置CF = OF = 0，根据结果设置SF、ZF和PF状态，而对AF未定义**

# 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=0F=0, SF=0、ZF=0、PF=0

mov al, 45h

； 逻辑或 al=75h

or al, 31h

； CF=0F=0, SF=0、ZF=0、PF=0

mov al, 45h

； 逻辑异或 al=74h

xor al, 31h

； CF=0F=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

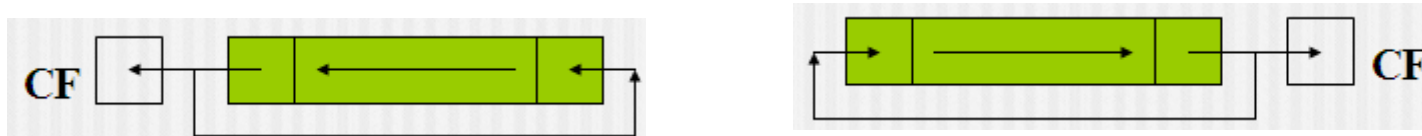
## SHL/SHR



## SAL/SAR



## 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

# 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

- **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。



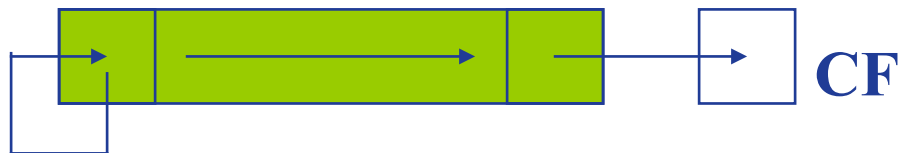
注意: 逻辑右移一位等价于将一个无符号数除以2(整除)

# 算术SAL/SAR

❖ 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**

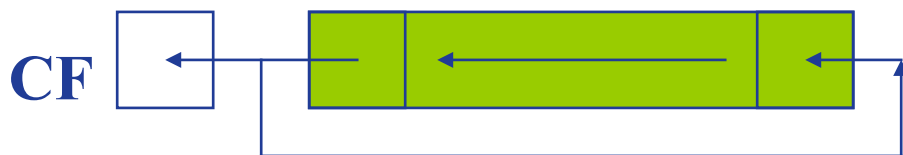


- 算术右移一位等价于将一个**带符号数除以2**（整除）
- 注意：当操作数为负数且最低位有1移出时，**SAR与IDIV**结果不同，如-1右移后为-1，不为0

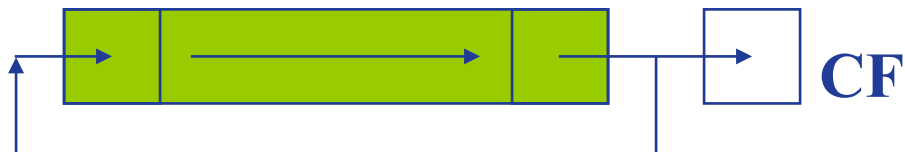
# 循环移位指令 (rotate)

不带进位

❖ 循环左移 ROL Dst, Src

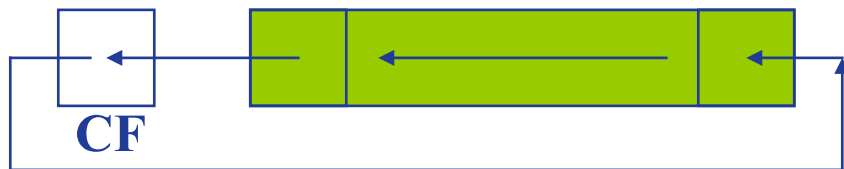


❖ 循环右移 ROR Dst, Src

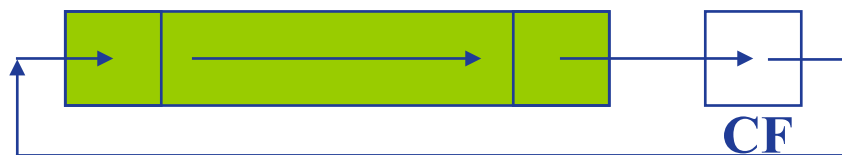


# 带进位循环RCL/RCR

❖ 带进位循环左移 RCL Dst, Src



❖ 带进位循环右移 RCR Dst, Src

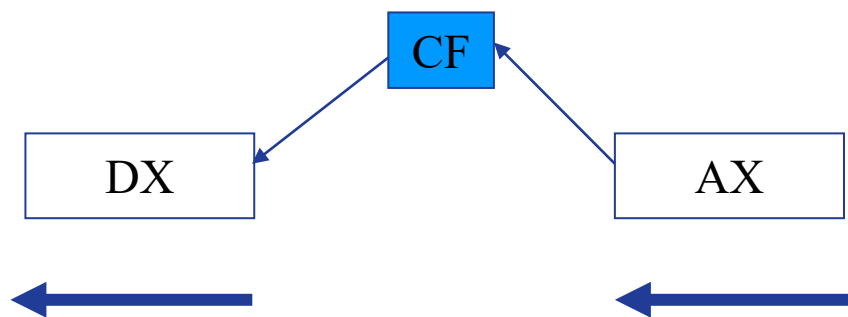




# 例子（一）

## ❖ 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/  $D_7/D_6/D_5/D_4$ )
- `SHL DL, CL` ; DL左移4位 ( $D_3/D_2/D_1/D_0$ / 0/0/0/0)
- `OR BL, DL`
  
- `MOV CL, 4`
- `ROL/ROR BL, CL`

## ❖ 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

- ❖ 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 =  $(CS) \times 16 + (IP)$
  - **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。

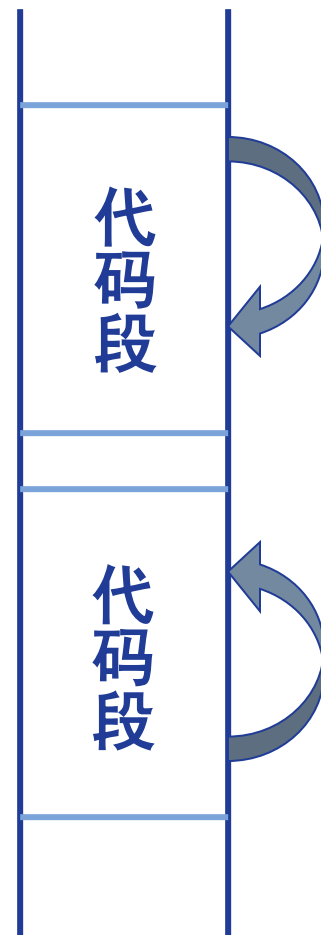
# 目标地址的范围：段内

## ❖ 段内转移——近转移（near）

- 在当前代码段 $2^{16}=64\text{KB}$ 范围内转移（ $\pm 32\text{KB}$ 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 $2^8=-128\sim+127$ 范围的转移

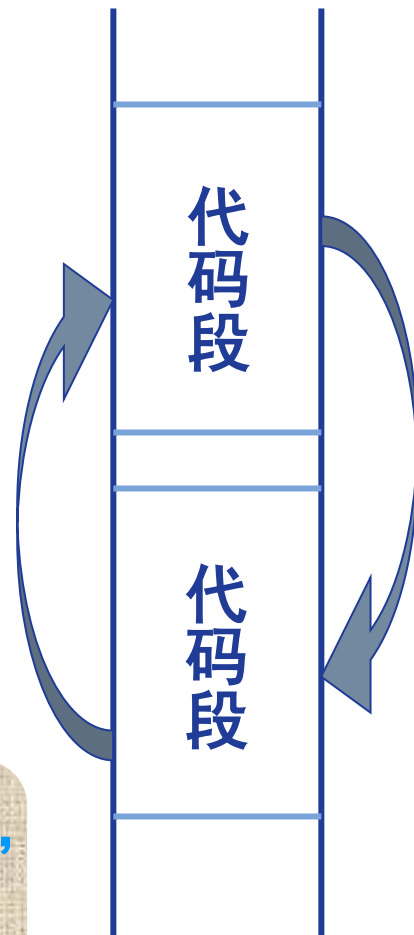


# 目标地址的范围：段间

## ❖ 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

- 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



动画

# 段内直接转移——相对寻址

## 用标号表达

JMP label ;  $IP \leftarrow IP + \text{位移量}$

❖ 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 $A \geq B$ 时转移；(above or equal)

❖ JB 标号；当 $A < B$ 时转移；(below)

❖ JBE 标号；当 $A \leq B$ 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

# 判断单个标志位状态

❖ 这组指令单独判断5个状态标志之一

- (1) JZ/JE和JNZ/JNE：利用零标志ZF，判断结果是否为零（或相等）
- (2) JS和JNS：利用符号标志SF，判断结果是正是负
- (3) JO和JNO：利用溢出标志OF，判断结果是否产生溢出
- (4) JP/JPE和JNP/JPO：利用奇偶标志PF，判断结果中“1”的个数是偶是奇
- (5) JC/JB/JNAE和JNC/JNB/JAE：利用进位标志CF，判断结果是否进位或借位

## 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
```

```
dec cx  
jnz again
```

```
; 字符个数减1, 不为0继续循环
```

```
done: mov result, bx;
```

# 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
```



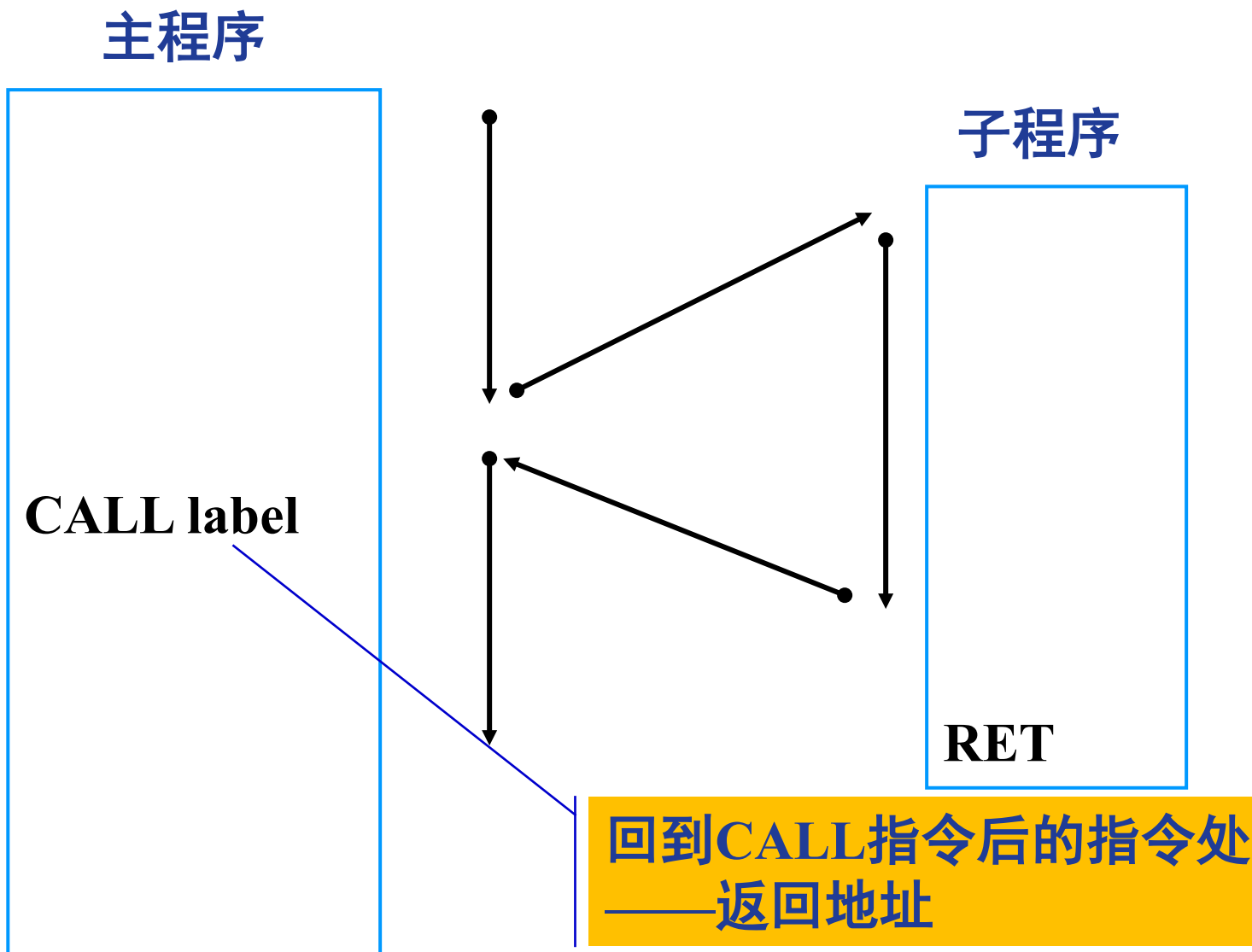
## 记录BX中1的个数

```
                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
```

## ❖ 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  
 $SP \leftarrow SP - 2, SS: [SP] \leftarrow IP$
- **段间调用**——偏移地址IP和段地址CS入栈  
 $SP \leftarrow SP - 2, SS: [SP] \leftarrow CS$   
 $SP \leftarrow SP - 2, SS: [SP] \leftarrow IP$

# 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

$IP \leftarrow SS:[SP], \quad SP \leftarrow SP + 2$

- 段间返回——偏移地址IP和段地址CS出栈

$IP \leftarrow SS:[SP], \quad SP \leftarrow SP + 2$

$CS \leftarrow SS:[SP], \quad SP \leftarrow SP + 2$

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 $SP \leftarrow SP + i16$

❖ 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`

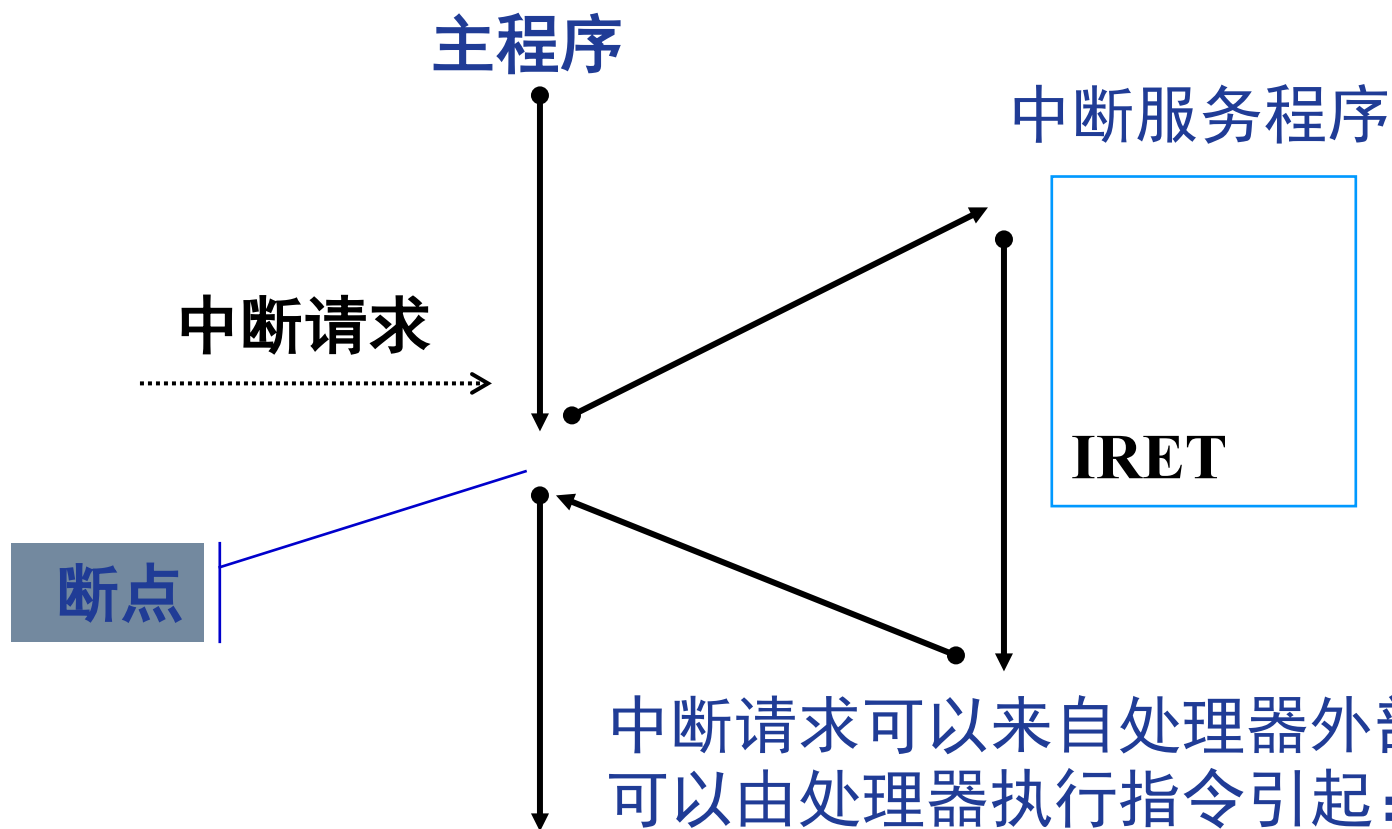
`add al,7`           ； 是3Ah~3Fh， 加上7

`htoend: ret`           ； 子程序返回

P7 表1.2

# 中断相关概念

中断（Interrupt）是又一种改变程序执行顺序的方法  
计算机暂停现行政程序的运行，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
- ❖ 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

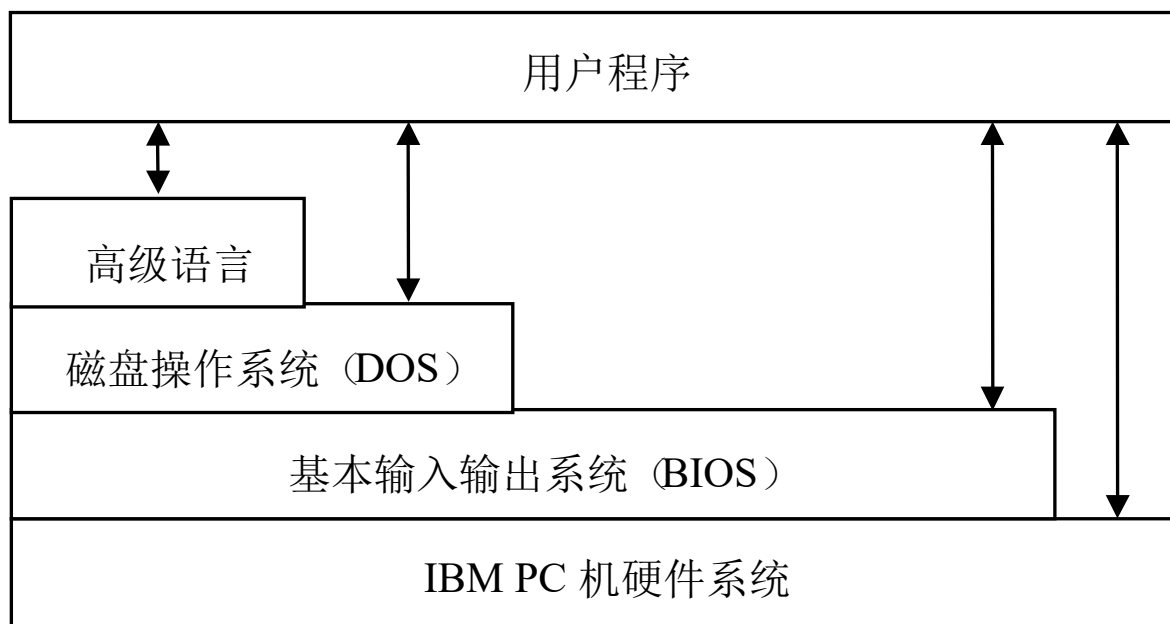
## ❖ 中断返回指令 IRET

- 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

P67

## ❖ 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
```

```
mov dx, offset string
```

```
int 21h
```

; 设置功能号: ah←09h

; 入口参数: dx←字符串的偏移地址

; DOS功能调用: 显示

## ❖ 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

## ❖ 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**



❖ 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；



## ❖ 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

## ❖ 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)**