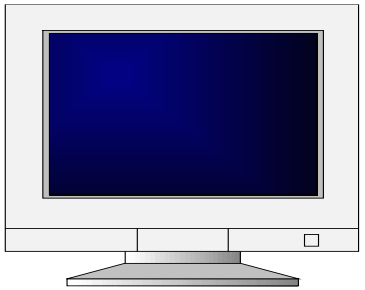
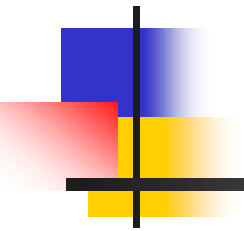
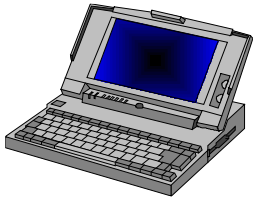
80X86指令系统(2)



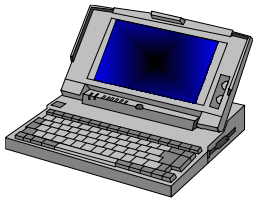


逻辑运算和移位指令



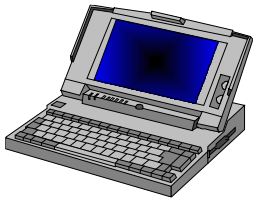
指令类型

- 逻辑运算
  - 与，或，非，异或
- 移位操作
  - 非循环移位，循环移位



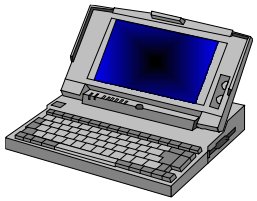
一、逻辑运算

- 逻辑运算指令对操作数的要求大多与MOV指令相同。
- “非”运算指令要求操作数不能是立即数；
- 除“非”运算指令 外，其余指令的执行都会使标志位OF=CF=0，标志位SF、ZF、PF根据结果设置。
- “非”运算指令不影响标志位



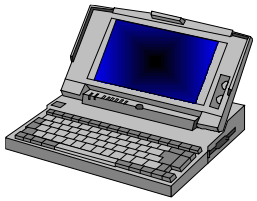
1.“与”指令：

- 格式：
  - AND OPRD1，OPRD2
- 操作：
  - 两操作数按位相“与”，结果送目标地址。



“与”指令的应用

- 实现两操作数按位相与的运算
  - AND BL，[ESI]
- 使目标操作数的某些位不变，某些位清零
  - AND AL，0FH
- 在操作数不变的 情况下使CF和OF清零
  - AND AX，AX

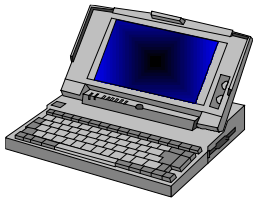


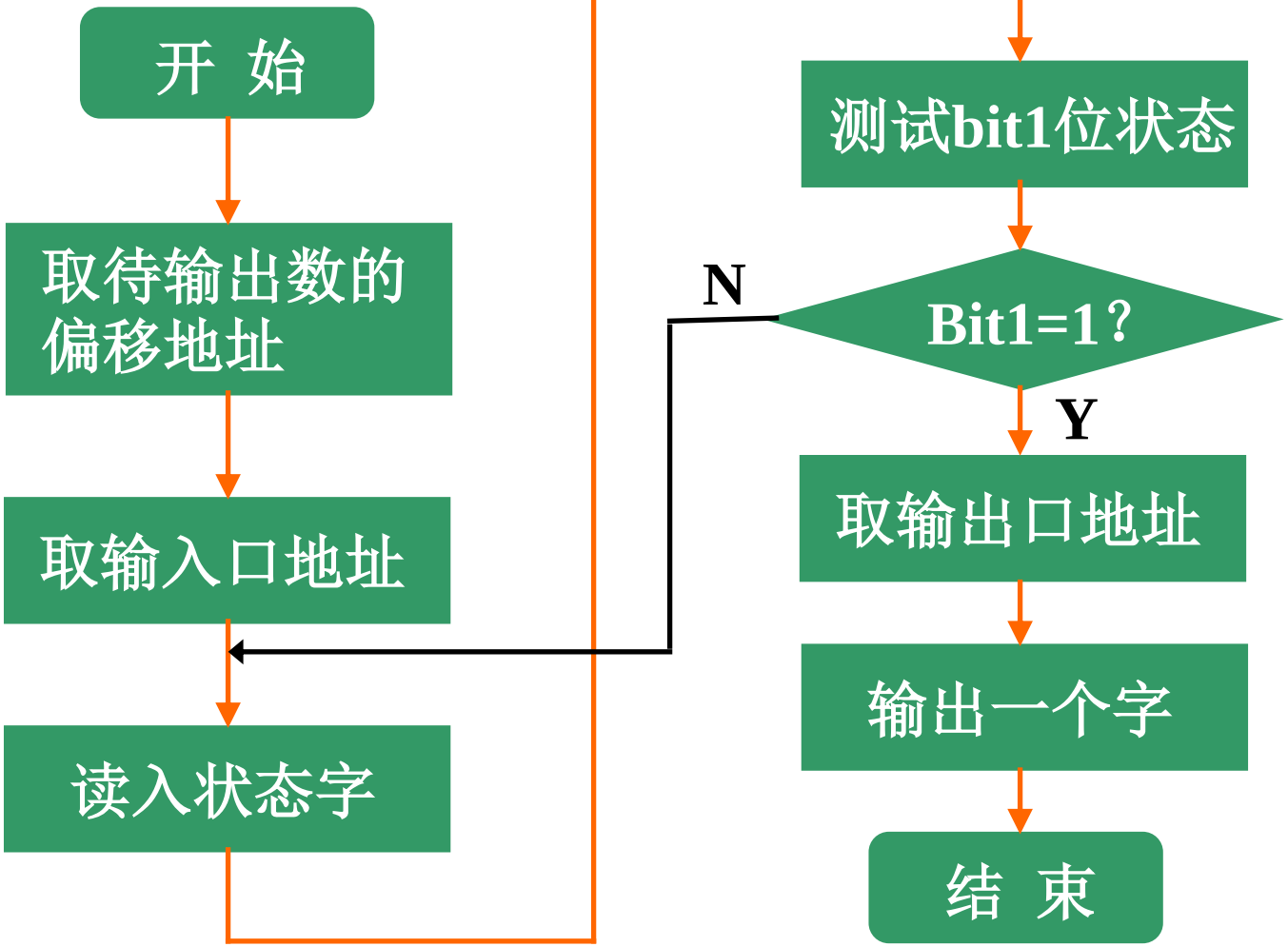
“与”指令应用例

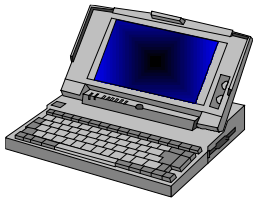
- 从地址为3F8H I/O端口中读入一个字节数，如果该数bit1位为1，则将DATA为首地址的一个字输出到38FH端口，否则就不能进行数据传送。

  编写实现该功能的程序段。

“与”指令应用例

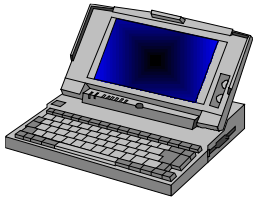



开 始
取待输出数的偏移地址
取输入口地址
读入状态字
测试bit1位状态
Bit1=1?
N
Y
取输出口地址
输出一个字
结 束



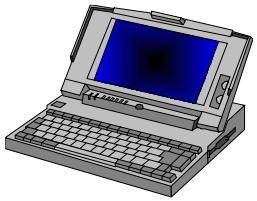
“与”指令应用例

```
        LEA  ESI, DATA
        MOV  DX, 3F8H
WAIT:   IN  AL, DX
        AND  AL, 02H
        JZ   WAIT              ; ZF=1转移
        MOV  DX, 38FH
        MOV  AX, [ESI]
        OUT  DX, AX
```



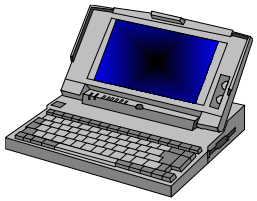
2.“或”运算指令

- 格式:
  - OR OPRD1，OPRD2
- 操作:
  - 两操作数按位相“或”，结果送目标地址



“或”指令的应用

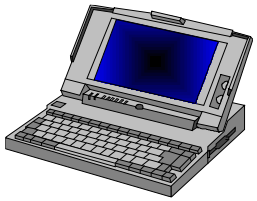
- 实现两操作数 相“或”的运算
  - OR AX，[EDI]
- 使某些位不变，某些位置“1”
  - OR CL，0FH
- 在不改变操作数的情况下使OF=CF=0
  - OR AX，AX



“或”指令的应用例

```
OR  AL，AL
JPE  GOON
OR  AL，80H
GOON：….
```

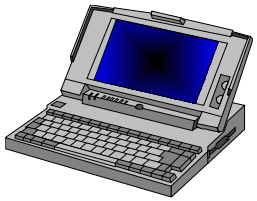
PF=1转移



“或”指令的应用

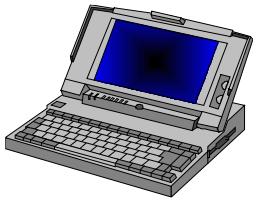
将一个8位二进制数9变为字符‘9’

如何实现？



3.“非”运算指令

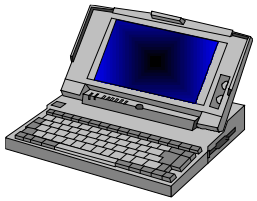
- **格式:**
  - NOT　OPRD
- **操作:**
  - **操作数按位取反再送回原地址**
- **注:**
  - **指令中的操作数不能是立即数**
  - **指令的执行对标志位无影响**
- **例:** NOT BYTE PTR[EBX]



4.“异或”运算指令

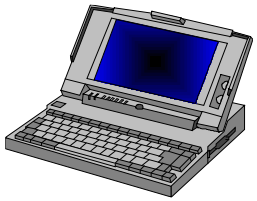
- 格式:
  - XOR OPRD1，OPRD2
- 操作:
  - 两操作数按位相“异或”，结果送目标地址
- 例:

```
XOR BL，80H ;将BL的最高位变反
XOR AX，AX
```



5. “测试”指令

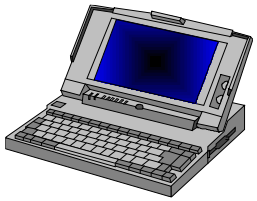
- 格式：
  - TEST OPRD1，OPRD2
- 操作：
  - 执行“与”运算，运算的结果影响标志位，但不送回目标地址。
- 应用：
  - 常用于测试某些位的状态



例：

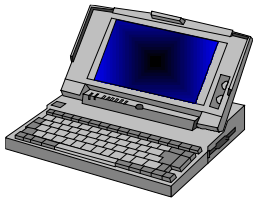
- 从地址为3F8H的端口中读入一个字节的状态数据，当该数的 bit1， bit3， bit5位同时为1时，则从38FH端口将DATA为首地址的一个字输出，否则就从端口重新输入状态数据。

  编写实现该功能的程序段。



源程序代码：

```
        LEA  ESI，DATA
        MOV  DX，3F8H
WAIT:   IN  AL，DX
        TEST  AL，02H
        JZ WAIT                    ；ZF=1转移
        TEST AL，08H
        JZ WAIT
        TEST AL，20H
        JZ WAIT
        MOV  DX，38FH
        MOV  AX，[ESI]
        OUT  DX，AX
```

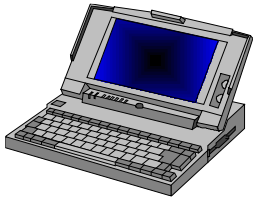


源程序代码：

```
        LEA  ESI，DATA
        MOV  DX，3F8H
WAIT:   IN  AL，DX

        AND  AL，2AH
        CMP AL，2AH
        JNZ   WAIT

        MOV  DX，38FH
        MOV  AX，[ESI]
        OUT  DX，AX
```

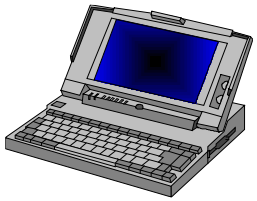


源程序代码：

```
        LEA  ESI，DATA
        MOV  DX，3F8H
WAIT:   IN  AL，DX

        AND AL，2AH
        XOR AL，2AH
        JNZ WAIT

        MOV  DX，38FH
        MOV  AX，[ESI]
        OUT  DX，AX
```

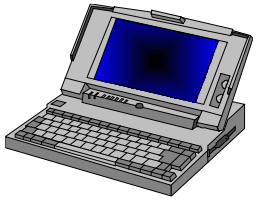


二、移位指令

- 非循环移位指令
- 循环移位指令

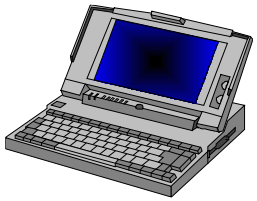
注：

- 移动次数用CL或8位常数来指定，即移位次数为0~255；
- 对于16位模式常数只能为1，即只有移位1次，才可以在指令中直接写常数“1”。



1. 非循环移位指令

- 逻辑左移
- 算术左移
- 逻辑右移
- 算术右移



算术左移和逻辑左移

- **算术左移指 令:**

  SAL OPRD，imm8
  SAL OPRD，CL

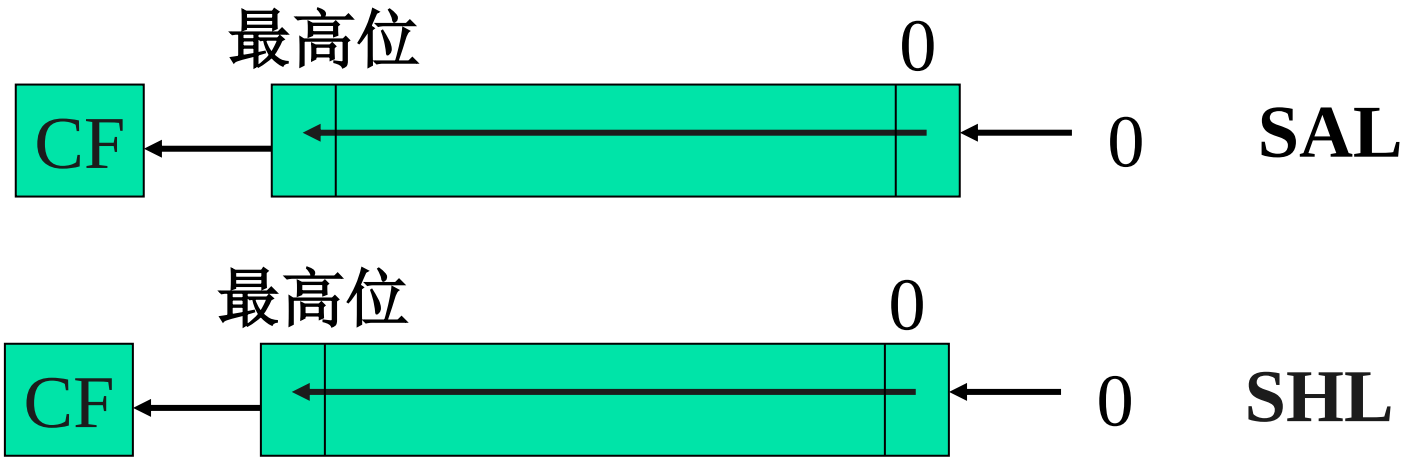
  } 有符号数

- **逻辑左移指 令:**

  SHL OPRD，imm8
  SHL OPRD，CL

  } 无符号数

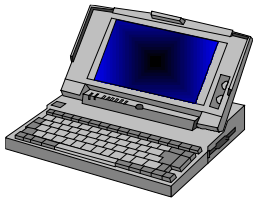
算术左移和逻辑左移



大家发现这两条指令有什么差别？

相同！

其实在指令系统中它们是同一条指令



算术右移和逻辑右移

- **算术右移指 令:**

  SAR OPRD，imm8

  SAR OPRD，CL

  } 有符号数

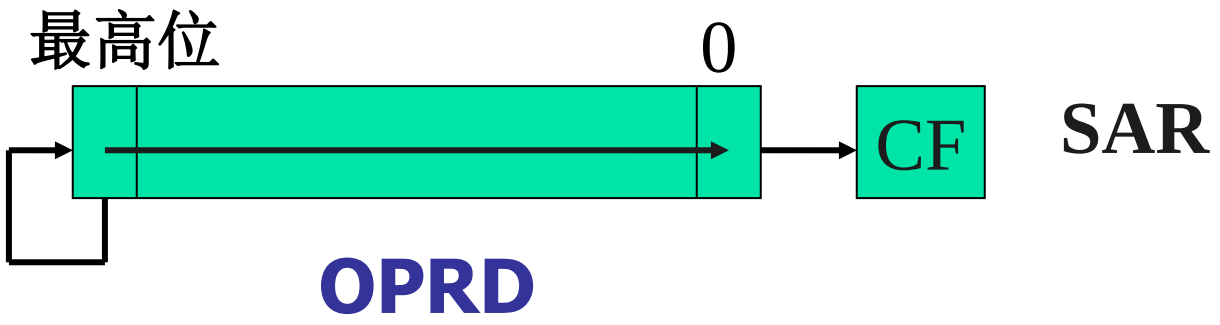
- **逻辑右移指 令:**

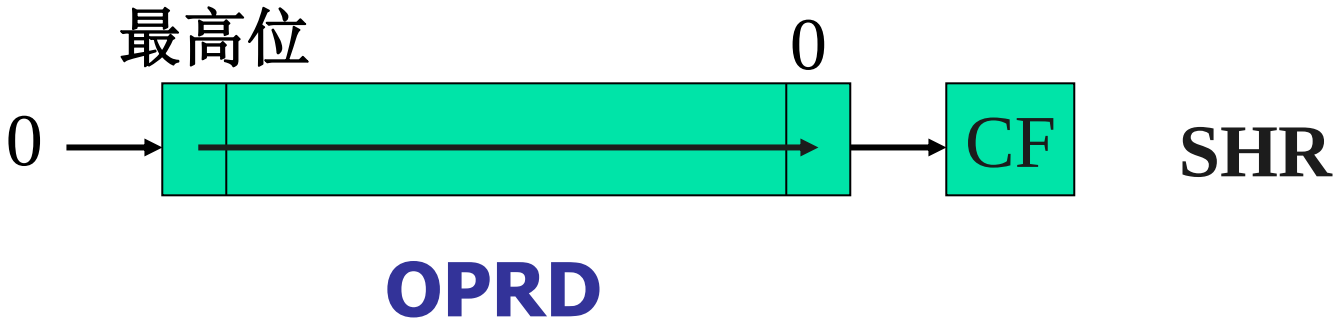
  SHR OPRD，imm8

  SHR OPRD，CL

  } 无符号数

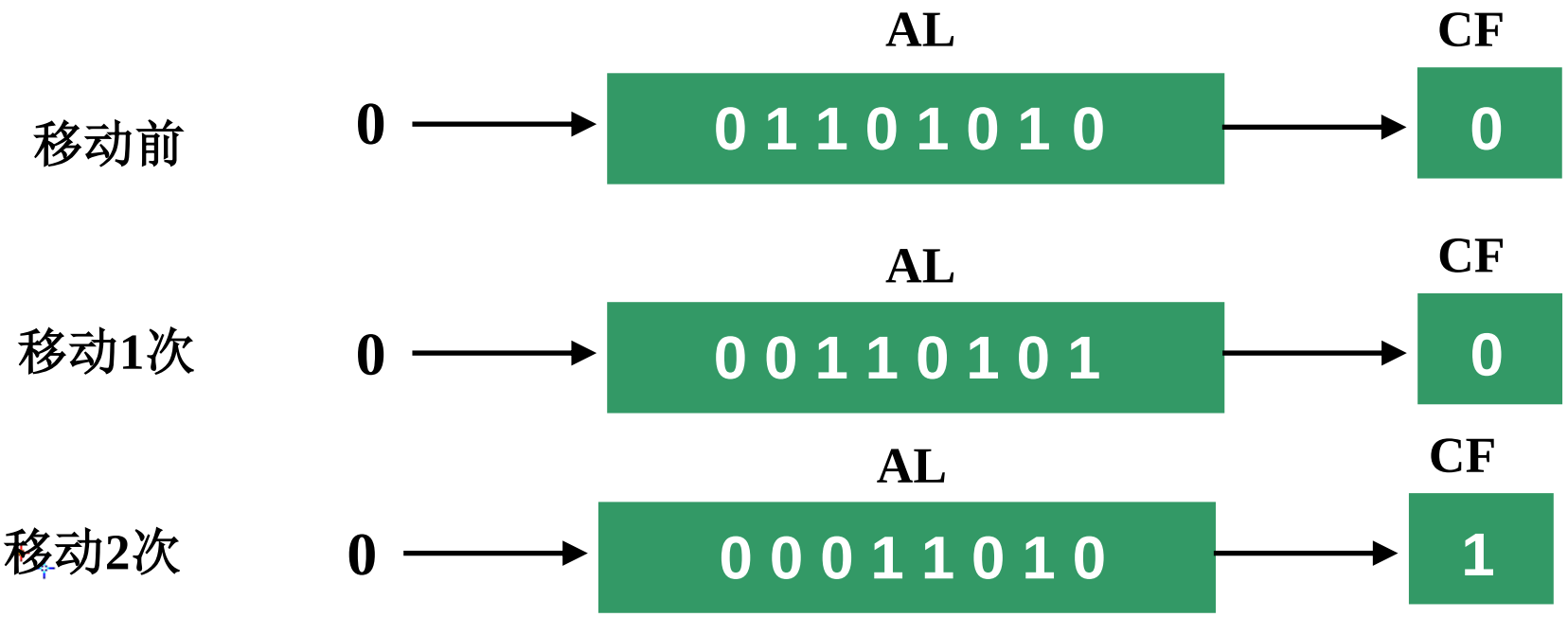
算术右移和逻辑右移

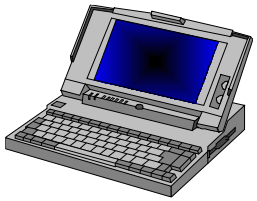




逻辑右移例：

- MOV AL，6AH
- SHR AL，2



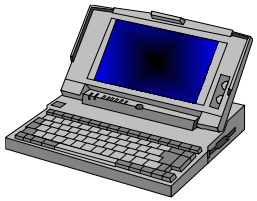


非循环移位指令的应用

- 左移可实现乘法运算,右移可实现除法运算
- 当移位次为n时，其作用分别相当于乘以2^n和除以2^n
- SAL和SAR将操作数视为带符号数，SHL和SHR将操作数视为无符号数。

例5 设AX中存放一个带符号数，若要实现(AX) ×5÷2，可由以下几条指令完成。

```
MOV    DX，AX
SAL    AX，2
ADD    AX，DX
SAR    AX，1
```



2. 循环移位指令

- **不带进位位的循环移位**
  - 左移 **ROL**
  - 右移 **ROR**
- **带进位位的循环移位**
  - 左移 **RCL**
  - 右移 **RCR**

指令格式、对操作数的要求与非循环移位指令相同

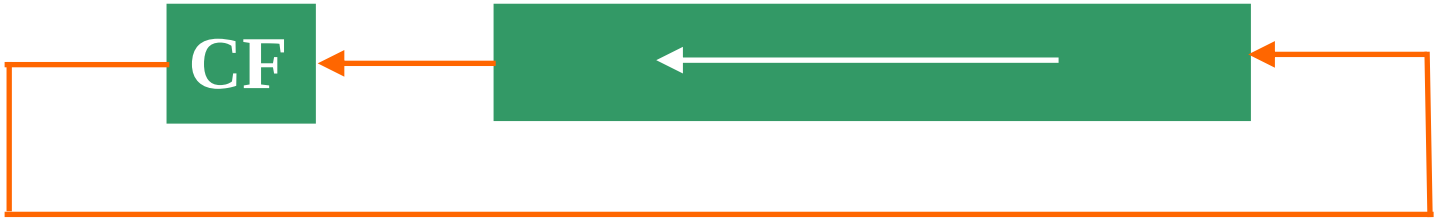
不带进位位的循环移位

ROL：

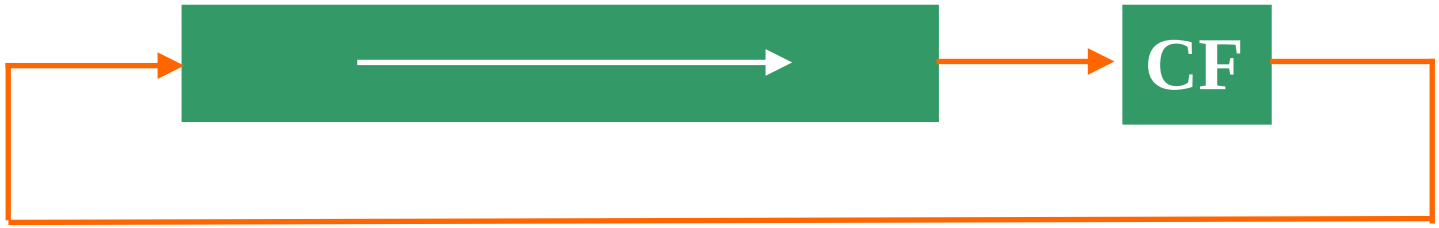
ROR：

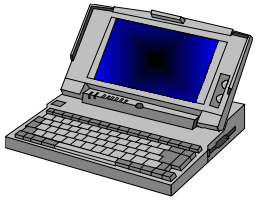
带进位位的循环移位

RCL：



RCR：





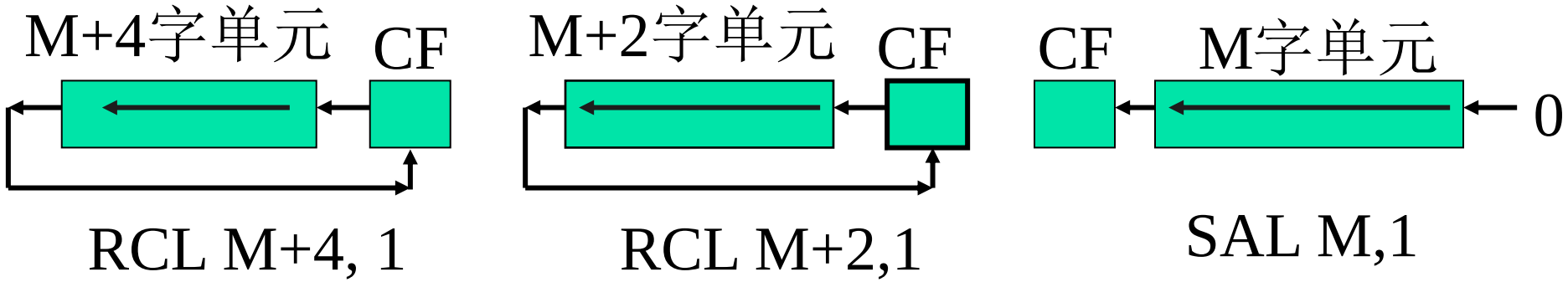
循环移位指令的应用

- 用于对某些位状态的测试；
- 高位部分和低位部分的交换；
- 与非循环移位指令一起组成32位或更长字长数的移位。

多字节单元数据联合移位例子

例　下面程序段对从存储单元M开始的三字数据执行左移一位。

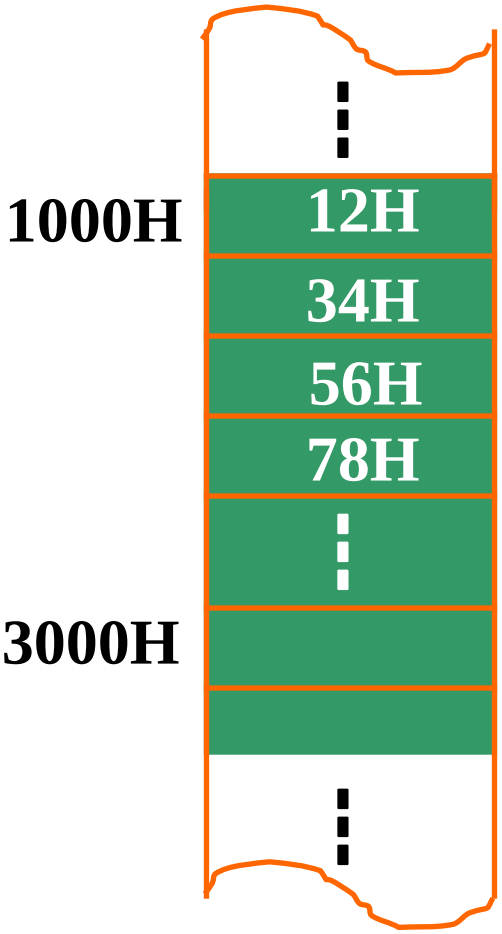
```
SAL　M，1
RCL　M+2，1
RCL　M+4，1
```

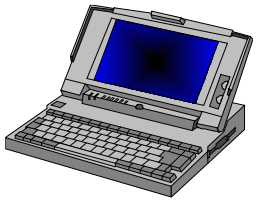


如果要联合移位2位或更多位，如何实现？

移位指令应用例子

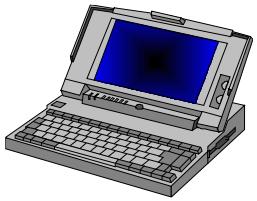
- 将1000H开始存放的4个压缩BCD码转换为ASCII码存放到3000H开始的单元中去。





实现程序

```
        MOV ESI,1000H
        MOV EDI,3000H
        MOV CX,4
Next:   MOV AL,[ESI]
        MOV BL,AL
        AND AL,0FH
        OR AL,30H
        MOV [EDI],AL
        INC EDI
        MOV AL,BL
        SHR  AL,4
        OR   AL,30H
        MOV  [EDI],AL
        INC  EDI
        INC ESI
        DEC CX
        JNZ  Next
```



3.双精度移位指令

- 32位模式下增加了双精度移位指令SHLD和SHRD
- 双精度左移指令：

  SHLD dest，source，count

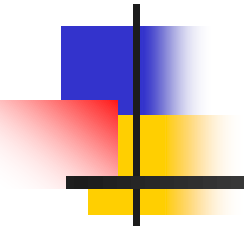
其中，count为CL或8位常数，指定移位次数。

dest可以是寄存器或存储器操作数，source只能是寄存器操作数。dest与source的长度必须一致。

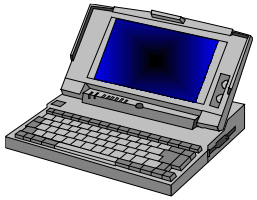
操作：将目的操作数dest向左移动指定位数，移动形成的空位由源操作数source的高位填充，指令执行后source保持不变。

对标志位的影响：会影响SF、ZF、AF、PF、CF

- 双精度右移指令：SHRD dest，source，count

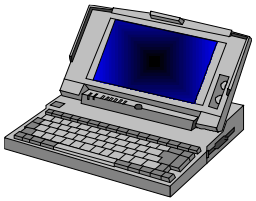


串操作指令



串操作指令说明

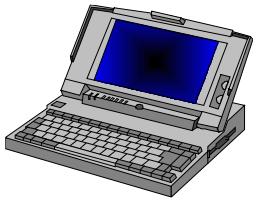
- 针对数据块或字符串的操作；
- 可实现存储器到存储器的数据传送；
- 待操作的数据串称为源串，目的地址称为目的串。



串操作指令的特点

- 源串地址由[ESI]提供，目的串由[EDI]提供。
- 在16位模式下源串的段基址由DS提供，目的串的段基址由ES提供，但32位模式下一般设为平坦模式，故不用考虑段寄存器。
- 每次只处理串中的一个单元(字节或字或双字)，并自动修改ESI和(或)EDI，使其指向下一个单元。
- 地址修改方向由DF标志位决定：

  DF=0 ⟶ 增地址方向； DF=1⟶ 减地址方向；
- 指令前面可加上自动重复前缀，实现自动重复执行串操作，重复执行次数由ECX指定。



重复前缀

重复执行串操作指令时，每执行一次则：ECX-1=>ECX

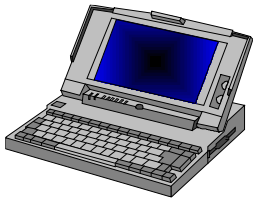
- 无条件重复
  - REP → 若ECX≠0 则重复
- 条件重复
  - REPE 相等重复
  - REPZ 为零重复

  } ECX≠0 ZF=1

  - REPNE 不相等重复
  - REPNZ 不为零重复

  } ECX≠0 ZF=0

- 注意：重复前缀本身是不改变标志位的

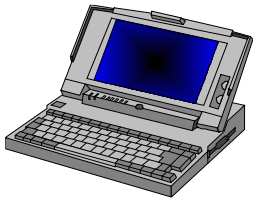


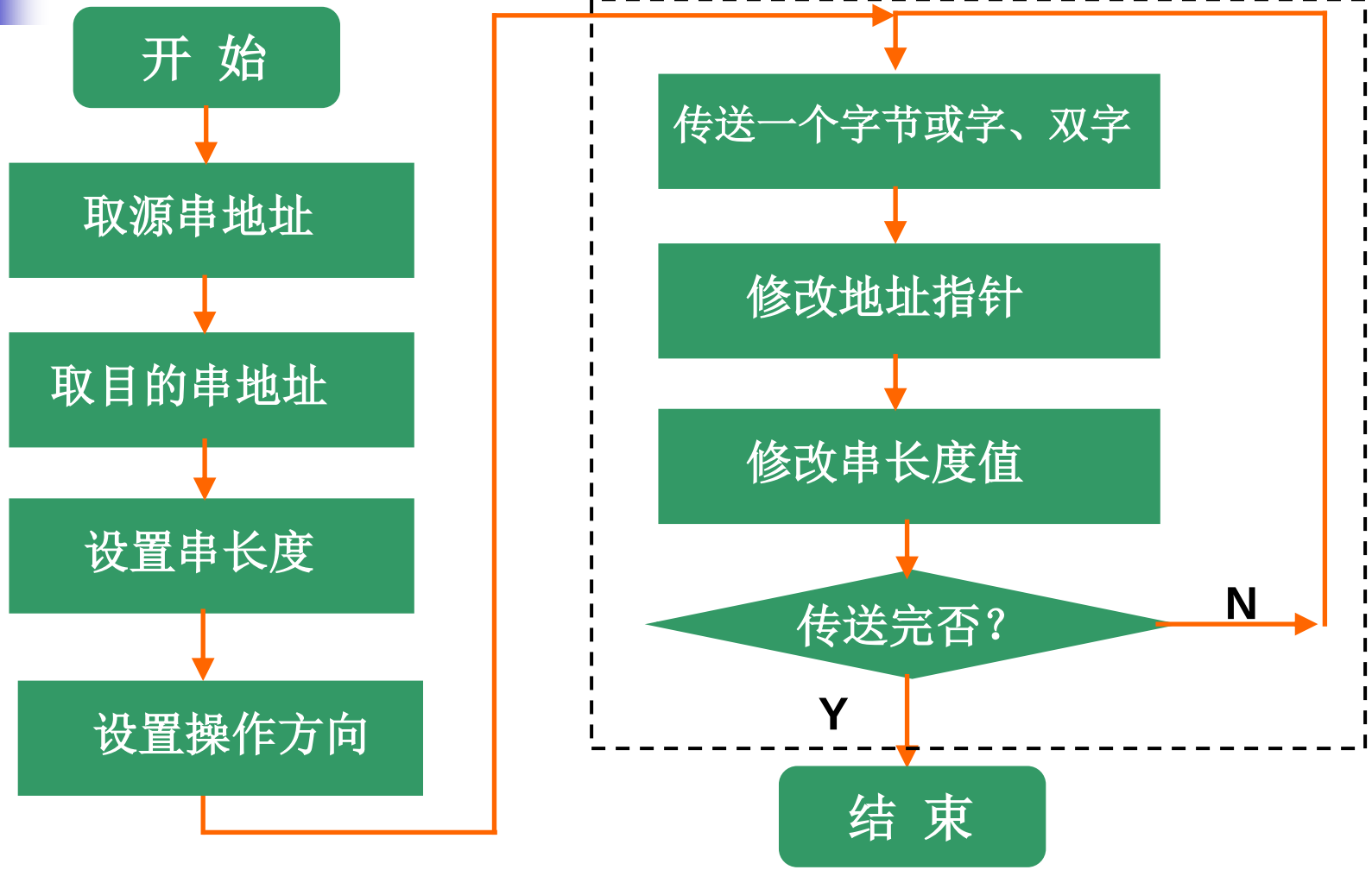
串操作指令

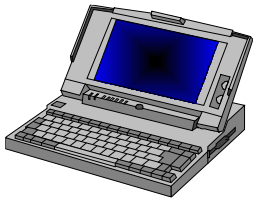
- 串传送 MOVS
- 串比较 CMPS
- 串扫描 SCAS
- 串装入 LODS
- 串送存 STOS

注意：串操作指令中可以出现两个存储器操作数。这是与其他指令不同的地方。

串操作指令使用流程(以传送操作为例)







1. 串传送指令

- 格式：

  目的串EDI　　源串ESI

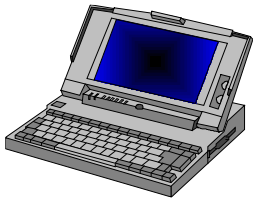
  MOVS　OPRD1，OPRD2

  MOVSB

  MOVSW

  MOVSD

- 串传送指令**不影响标志位**
- 串传送指令常与无条件重复前缀连用
- **能否与条件重复前缀连用?**



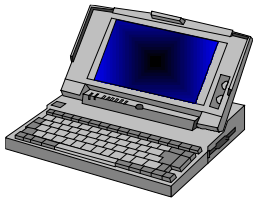
串传送指令的应用例子

- 对比用MOV指令和MOVS指令实现将200个字节数据从MEM1开始的一个内存区送到另一个从MEM2开始的区域的程序段。
- 用MOV指令实现：

```
LEA  ESI，MEM1
LEA  EDI，MEM2
MOV  ECX，200
NEXT: MOV AL,[ESI]
      MOV [EDI],AL
      INC ESI
      INC EDI
      DEC ECX
      JNZ  NEXT
```

- 用串指令实现：

```
LEA  ESI，MEM1
LEA  EDI，MEM2
MOV  ECX，200
CLD
REP  MOVSB
```



2. 串比较指令

- 格式：

源串　　目的串

CMPS　OPRD1，OPRD2

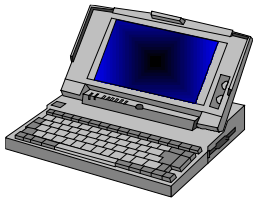
CMPSB

CMPSW

CMPSD

操作数位置与MOVS的区别！

- 执行OPRD1-OPRD2,不送结果。
- 串比较指令常与条件重复前缀连用，指令的执行不改变操作数，仅影响标志位。



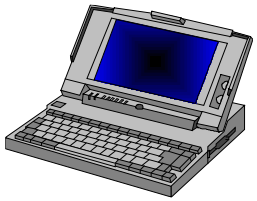
串比较指令使用例子

比较两组(200个字节)对应数据，找出第一个不同数据放入AL，其地址放入EBX

```
LEA  ESI，MEM1
LEA  EDI，MEM2
MOV  ECX，200
CLD
REPE  CMPSB
      JZ  STOP
      DEC  ESI
      MOV  AL，[ESI]
      MOV  EBX，ESI
STOP：HLT
```

为什么要减1？

指令执行结束后就可知道找到了一个不同的数据吗？为什么？



3. 串扫描指令

- 格式：

  SCAS　OPRD

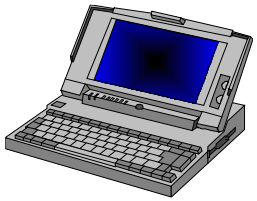
  SCASB

  SCASW

  SCASD

目的串用EDI指示

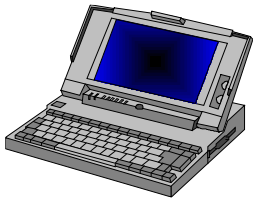
- 执行EAX/AX/AL-OPRD，结果不保存，只影响标志寄存器。



串扫描指令的应用

例：在给定一字符串“ABCDEFGH”中扫描一个匹配字符“F”，如果找到了该字母，EDI指向匹配字符串后面的一个字符；如果没有找到匹配字符，就执行JNZ指令退出。

```
ALPHA DB ‘ABCDEFGH’
COUNT EQU $ - ALPHA
…
MOV EDI , OFFSET ALPHA      ;EDI 指向字符串
MOV AL , ‘F’                ;查找字母F
MOV ECX , COUNT             ;设置查找计数器
CLD ； 方向 = 向前
REPNE SCASB                 ;不相等则重复
JNZ QUIT                    ;如果字符未找到则退出
DEC EDI                     ;找到了：EDI 回退
```



4. 串装入指令

- 格式：

  LODS　OPRD

  LODSB

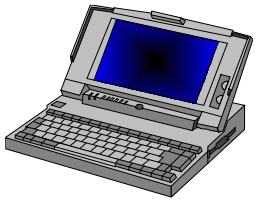
  LODSW

  LODSD

源串用[ESI]指示

- 操作：
  - 字节：　AL ← [ESI]
  - 字：　AX ← [ESI]
  - 双字：　EAX ← [ESI]
- 用于将内存某个区域的数据串依次装入累加器，以便进行处理（如显示或输出到接口）。
- LODS指令加重复前缀无意义。

为什么？



5. 串存储指令

- 格式:

  STOS　OPRD

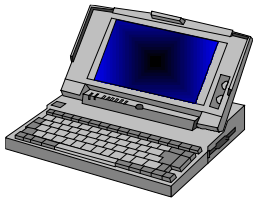
  STOSB

  STOSW

  STOSD

目的串用[EDI]指示

- 操作:
  - 字节:　AL → [EDI]
  - 字:　AX → [EDI]
  - 双字:　EAX → [EDI]

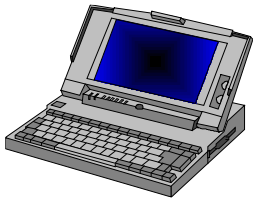


串存储指令的应用

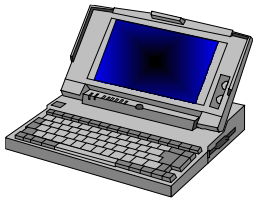
- 常用于将内存某个区域置同样的值
- 此时：
  - 将待送存的数据放入AL（字节数）或AX（字数据）或EAX（双字数据）；
  - 确定操作方向（增地址/减地址）和区域大小（串长度值）；
  - 使用串存储指令+无条件重复前缀，实现数据传送。

串存储指令例子：

例: 将STRING1开始100个字节都初始化为0FFH。



```
MOV  AL , 0FFH                    ;要存储的值
MOV  EDI , OFFSET  STRING1        ;EDI指向目标
MOV  ECX , COUNT                  ;字符计数
CLD                               ;方向 = 向前
REP  STOSB                        ;以AL中的值填充
```

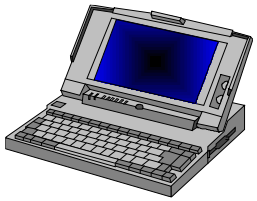


串操作指令应用注意事项

- 需要设置数据的操作方向
  - 确定DF的状态
- 源串和目的串指针必须分别为ESI和EDI，故可以使用隐含操作数的形式，如MOVSB、MOVSW、MOVSD。
- 若指令中给出了源串或目的串，与其形式无关，只用其表示数据类型（字节、字或双字）。
- 串长度值必须由ECX给出
- 注意重复前缀的使用方法
  - 传送类指令前加无条件重复前缀
  - 串比较类指令前加条件重复前缀，但前缀不影响ZF状态

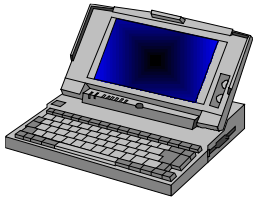
程序控制指令

- 转移指令
- 循环控制
- 过程调用
- 中断控制



程序的执行方向

- 程序控制类指令的本质：
  - 控制程序的执行方向
- 决定程序执行方向的因素：
  - 32位模式下EIP决定了从哪里取指令，也即程序执行顺序或方向。
  - 16位模式由CS和IP决定从哪里取指令。
- 控制程序执行方向的方法：
  - 32位模式下修改EIP
  - 16位模式修改IP或者CS 和IP



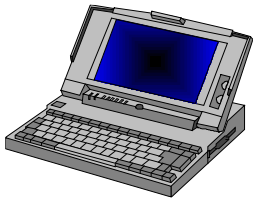
一、转移指令

无条件转移指令

无条件转移到目的地址，执行新的指令

有条件转移指令

在满足一定条件的情况下才转移到目的地址



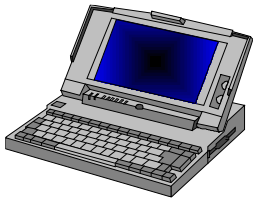
1. 无条件转移指令

- **格式：JMP OPRD**

OPRD：转移目的地。它可以是以下三种情况：

- **短转移—距离当前指令-128~+127字节范围**
- **近转移—与当前指令在同一代码段内**
- **远转移—转移到当前逻辑段之外的程序位置**

根据转移目的地OPRD的提供方式一般分为直接寻址和间接寻址两种形式。



直接寻址方式转移

- 转移的目的地OPRD为所在程序中的一个标号
  - 短转移：JMP指令与目的地OPRD之间的距离为-128~+127字节，指令长度为2个字节，第2个字节就是距离，称为位移量。
  - 近转移：距离超过-128~+127字节时，指令长度为5个字节，位移量为32位（4个字节），转移距离为-2^{31}~$+2^{31}-1$字节。
- JMP指令与目的地OPRD之间的距离计算是以JMP指令的下一条指令的起始地址为起点。
- 如果是相对本JMP指令的地址计算，则还要加上JMP指令的长度，即：

  短转移距离为-126~+129字节

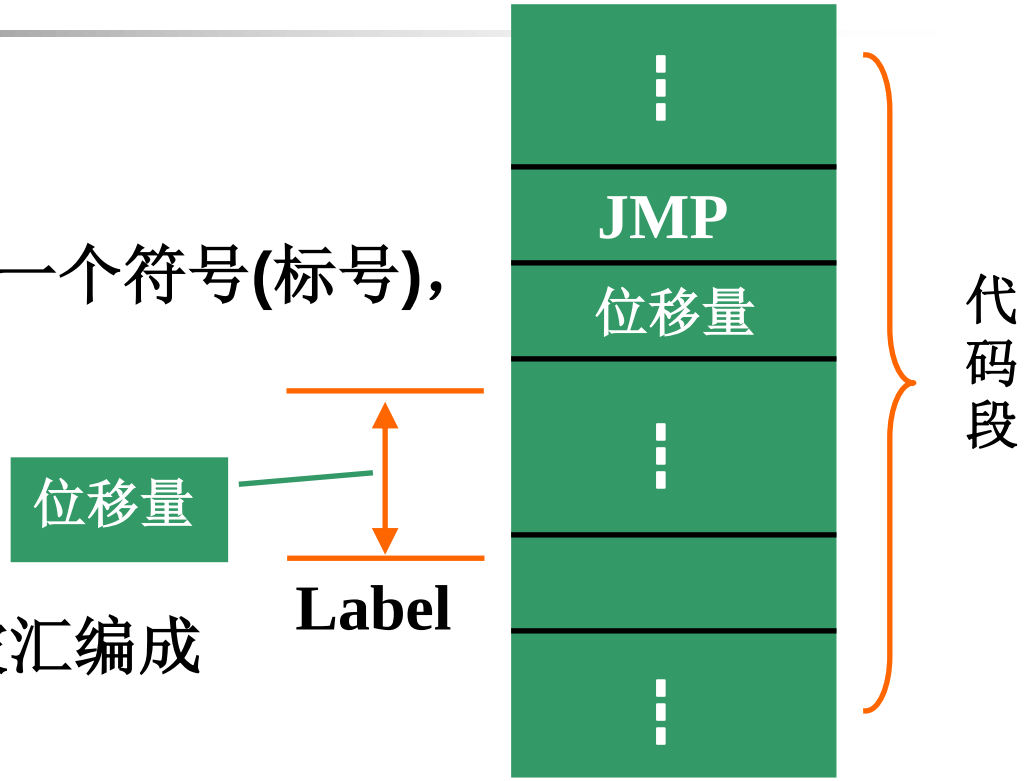
  近转移距离为$-2^{31}+5$~$+2^{31}+5$字节

执行JMP后 EIP=EIP当前值+位移量

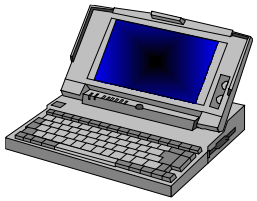
直接寻址方式转移示例

JMP Label

- **Label:** 汇编语言程序中是一个符号(标号)，表示一个存储器地址。



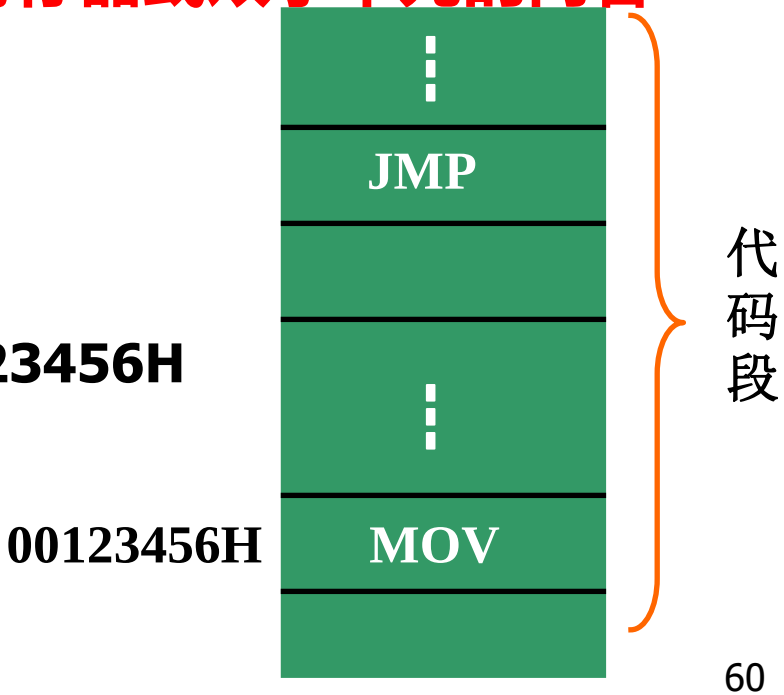
- 装入内存时，**Label**已经被汇编成一个**8**位或**32**位的位移量。
- 位移量表示**JMP**指令的下一条指令到**Label**地址之间的字节单元数。



间接寻址方式转移

- 间接寻址转移的目的地址存放在某个32位通用寄存器或存储器的某双字单元中。
- 执行JMP后 EIP=32位通用寄存器或双字单元的内容
- 例1：

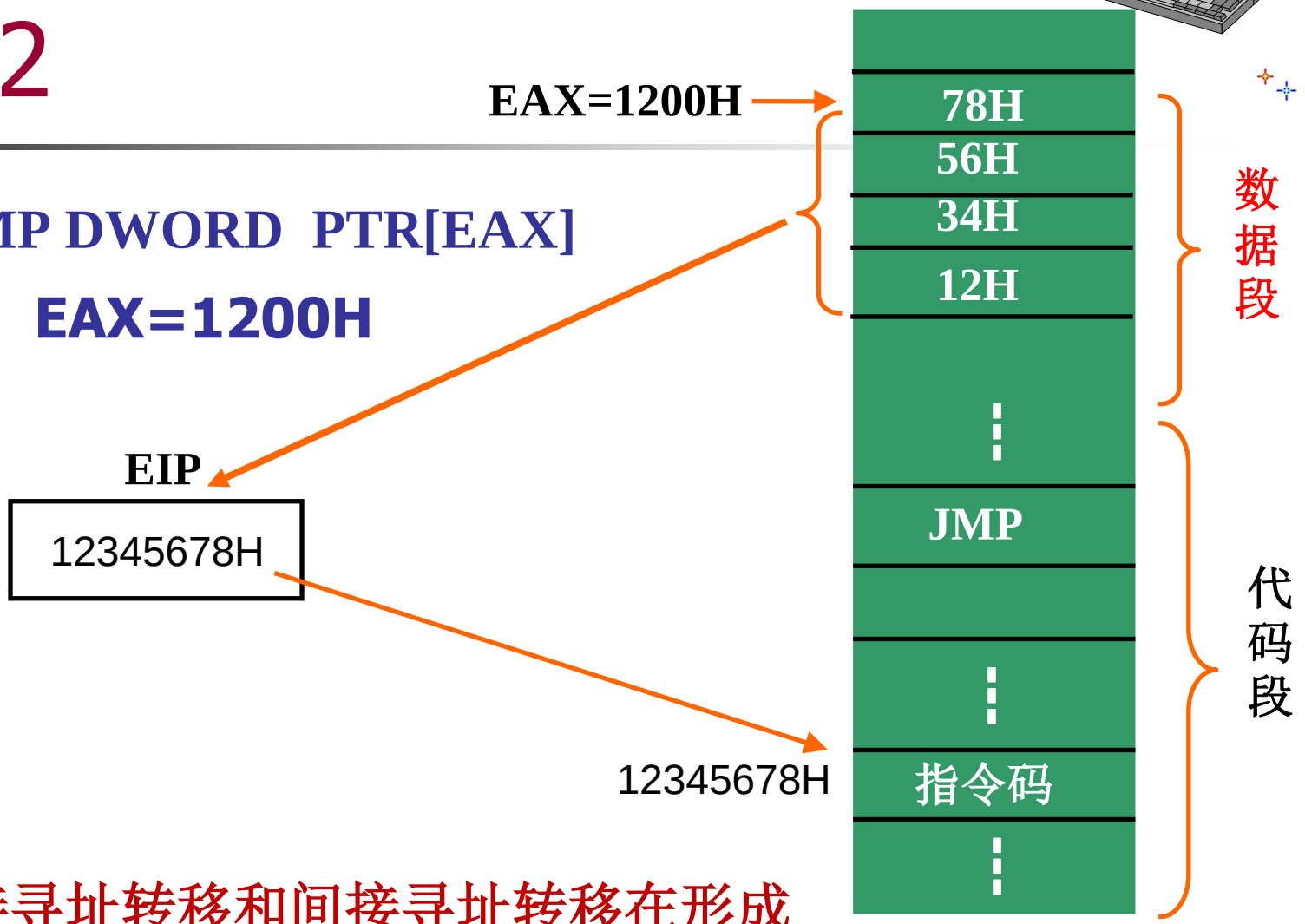
  JMP EBX
  - 若：EBX=00123456H
  - 则：转移的目的地址EIP=00123456H



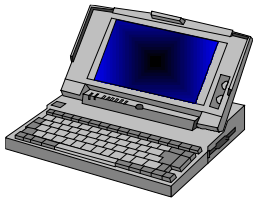
例2

- JMP DWORD PTR[EAX]

设：EAX=1200H



注意：直接寻址转移和间接寻址转移在形成EIP值的区别！



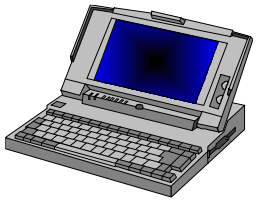
2. 条件转移指令

- 在满足一定条件下，程序转移到目标地址继续执行。
- 条件转移指令均为直接寻址的短转移，即转移的位移量为8位补码表示，范围为：

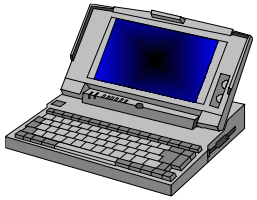
  -128~+127
- EIP=EIP+位移量

所有的条件转移指令见教材P102~103表3-2~表3-4

条件转移指令的应用



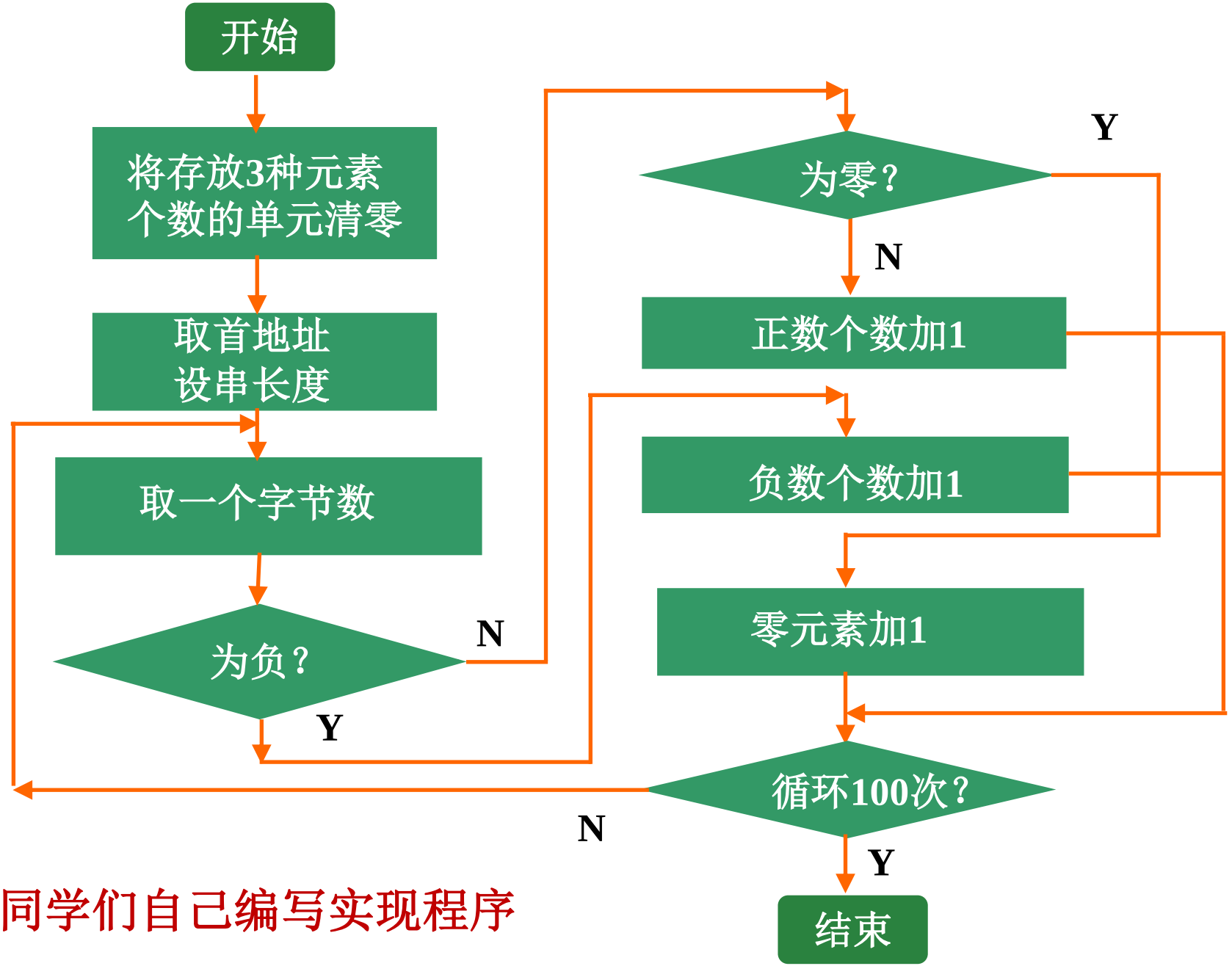
- 单个标志位的条件转移指令
  - JC/JNC
    - 判断CF的状态。常用于比大小
  - JZ/JNZ
    - 判断ZF的状态。常用于循环体的结束判断
  - JO/JNO
    - 判断OF的状态。常用于有符号数溢出的判断
  - JP/JNP
    - 判断PF的状态。用于判断运算结果低8位中1的个数是否为偶数
- 多标志位的条件转移指令
  - JA/JAE/JB/JBE
    - 判断CF或CF+ZF的状态。用于无符号数的大小比较
  - JG/JGE/JL/JLE
    - 判断SF、OF和ZF的状态。用于带符号数的大小比较
- 判断CX/ECX是否为0
  - JCXZ/JECXZ
    - CX=0(或ECX=0)转移



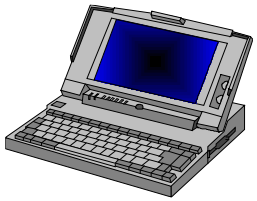
转移指令例

- 统计内存数据段中以TABLE为首地址的100个8位有符号数中正数、负数和零的个数。

实现程序流程



请同学们自己编写实现程序

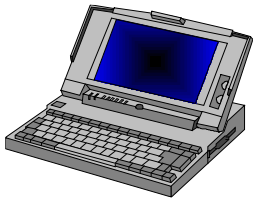


二、循环控制指令

- 循环范围：
  - 以当前EIP为中心的-128～+127范围内循环。
- 循环次数由ECX寄存器指定。
- 循环指令：

LOOP → 无条件循环指令

*LOOPZ
*LOOPNZ
} 条件循环指令(自学)



无条件循环指令

- 格式:

  LOOP LABEL

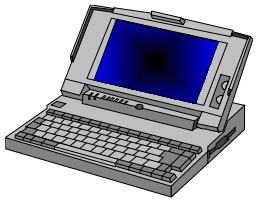
- 循环条件:

  ECX ≠ 0

- 操作:

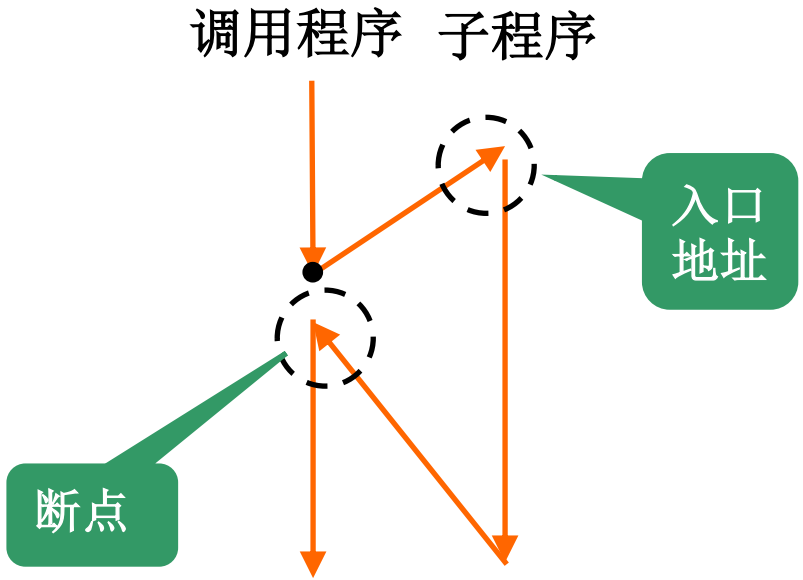
  (1) ECX-1 → ECX

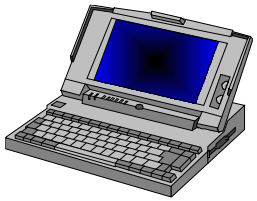
  (2) ECX ≠ 0则转LABEL，否则执行下条指令，即推出循环。



三、过程调用和返回

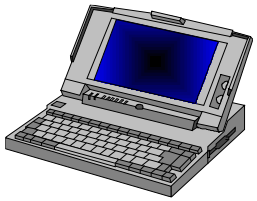
- 用于调用一个子过程；
- 子过程由程序员预先设计并装入内存
- 子过程执行结束后要返回原调用处
- 有两个重要的概念：**入口地址**和**断点**。





调用与返回的执行过程

- 保护断点；
  - 将调用指令的下一条指令的地址（断点）压入堆栈
- 获取子过程的入口地址；
  - 子过程第1条指令的地址
- 执行子过程，含相应参数的保存及恢复；
- 恢复断点，返回原程序。
  - 将断点地址由堆栈弹出



过程调用

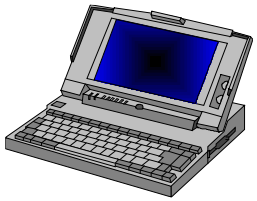
16位模式

段内调用

- 被调用程序与调用程序在同一代码段
  - 调用时只需保护断点的偏移地址IP
  - 执行过程:(1)将IP压入堆栈 (2)将过程入口地址→IP

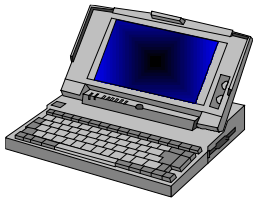
段间调用

- 被调用过程与调用程序不在同一代码段
  - 调用时需要保护断点的段基址CS和偏移地址IP
  - 执行过程:(1)将IP和CS压栈 (2)将过程入口地址→IP和CS



32位模式

- 在32位模式下一般采用flat平坦存储模式，不涉及段地址的保护与返回。
- 过程调用
  - (1) 32位的EIP压入堆栈
  - (2) 过程的入口偏移地址→EIP



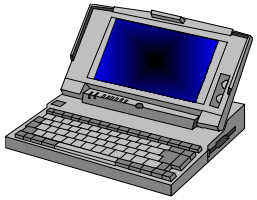
3. 返回指令

- 功能：
  - 从堆栈中弹出断点地址，返回原程序
- 格式：

  RET

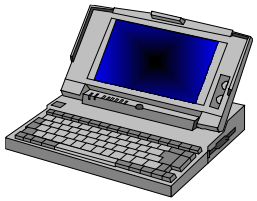
  或 RET n

- RET指令一般位于子程序的最后。
- 返回指令在格式上不区分16位模式（段内或段间）或32位模式。但不同的模式从堆栈中弹出的内容不一样。



四、中断指令

- 中断
- 中断源
- 中断的类型
- 中断指令
  - 引起CPU产生一次中断的指令,其响应过程按照一般的中断来处理。



中断与过程调用的区别

- 中断是随机事件或异常事件引起，调用则是事先已在程序中安排好；
- 响应中断请求不仅要保护断点地址，还要保护FLAGS(或EFLAGS)内容；
- 过程调用指令在指令中直接给出子程序入口地址；中断指令只给出中断向量码，入口地址则在中断向量码指向的中断向量表（内存单元）中。

1. 中断指令(也叫软中断指令)

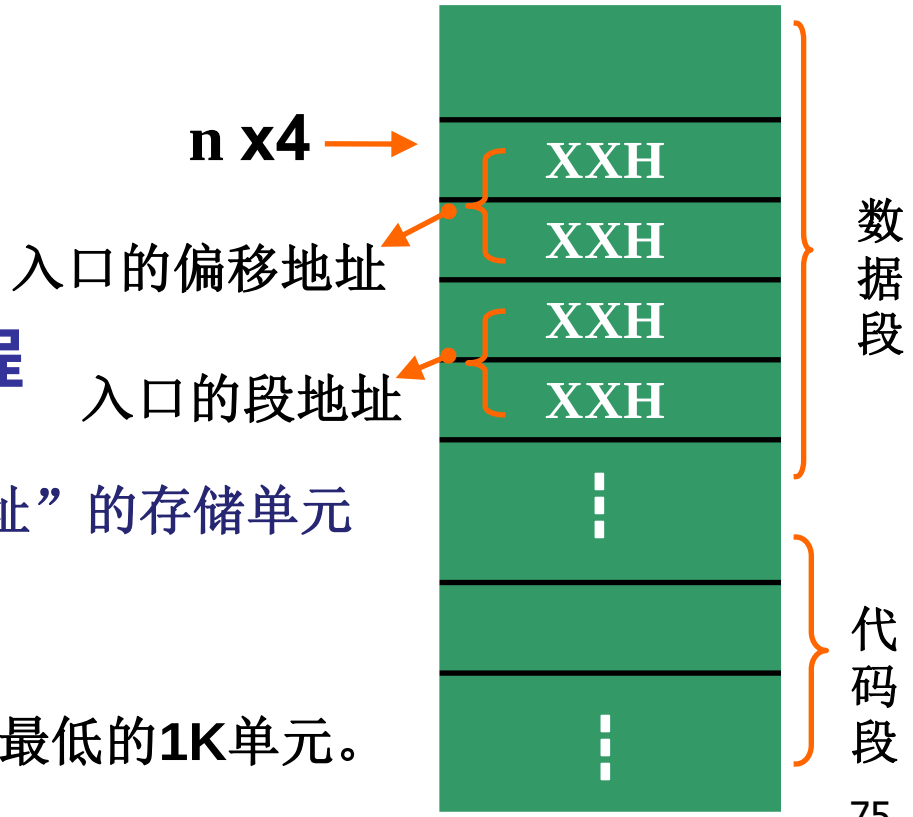
- **格式：**

  INT n

中断类型号
n=0 ~ 255

- **说明：n用来形成中断服务程序入口地址**

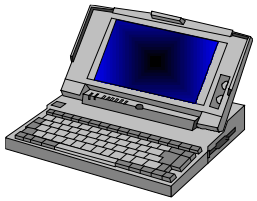
下面以16位模式为例说明其过程



(n×4)为存放“中断服务程序入口地址”的存储单元的偏移地址;

该单元在数据段，段地址=DS

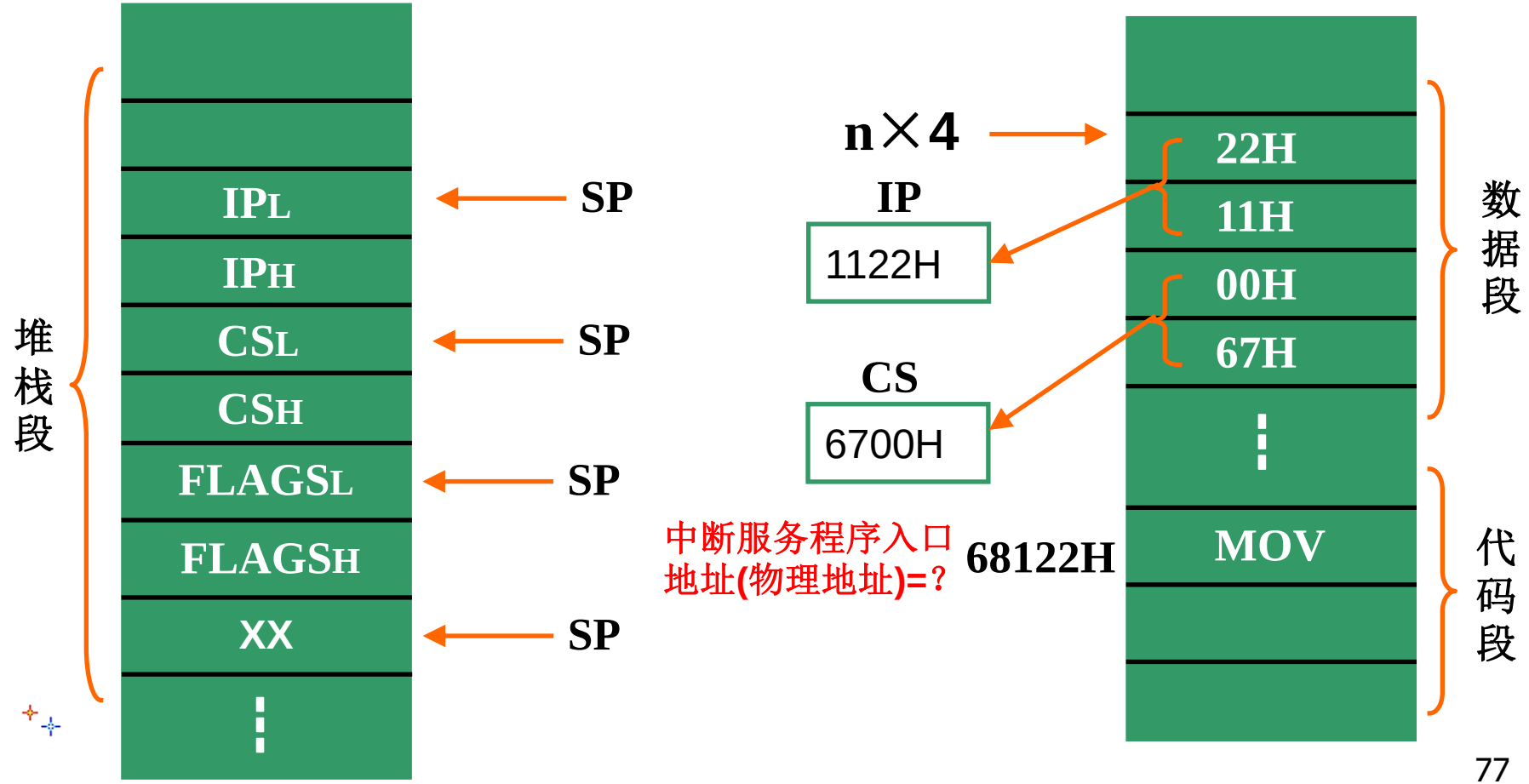
说明：8086的中断向量表存放在内存最低的1K单元。



中断指令的执行过程

- 将FLAGS压入堆栈；
- 将INT指令的下一条指令的CS、IP压栈；
- 由n×4得到存放中断向量的地址；
- 将中断向量（中断服务程序入口地址）送CS和IP寄存器；
- 转入中断服务程序。

中断指令的执行过程(续)


堆栈段
IPL
IPH
CSL
CSH
FLAGSL
FLAGSH
XX
SP
SP
SP
SP
n×4
IP
1122H
CS
6700H
中断服务程序入口地址(物理地址)=?
68122H
22H
11H
00H
67H
MOV
数据段
代码段

中断指令例

执行程序段：

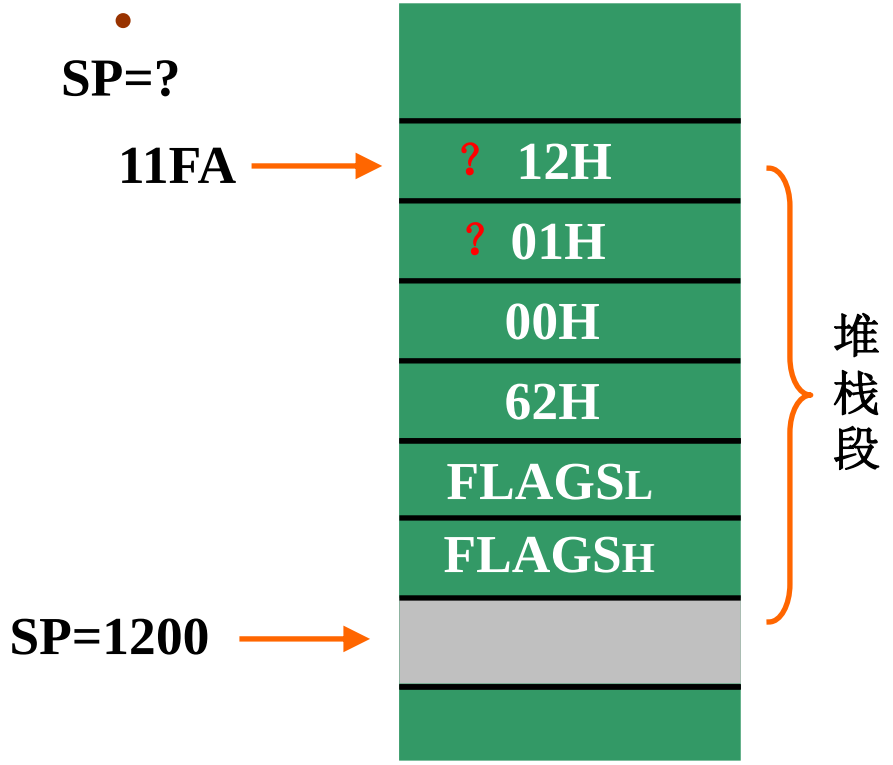
CS　　IP

6200H:0110H　INT 21H

6200H:0112H　MOV AX，BX

执行INT指令后

SP=?

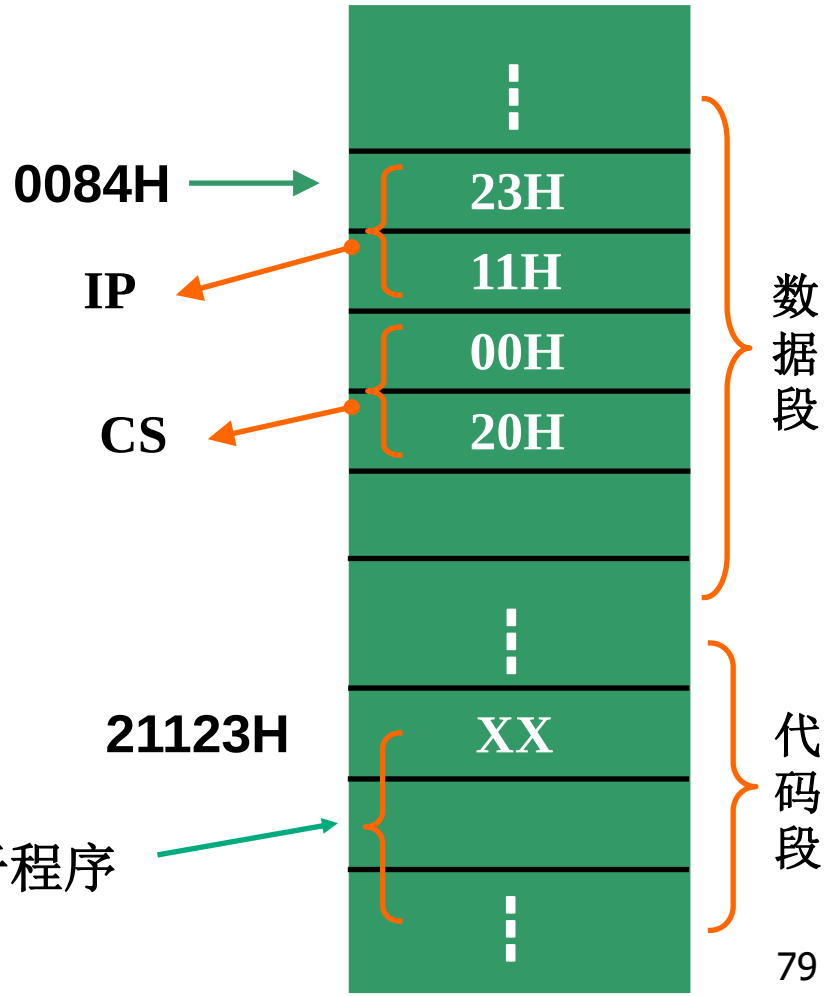


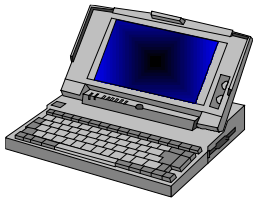
中断指令例

- 执行INT 21H指令后

  IP<=[21Hx4]

  CS<=[（21Hx4）+2]



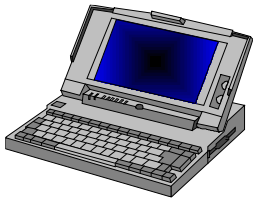


2. 溢出中断指令

- 格式：

  INTO

- 指令执行时检查OF标志：
  - 若OF=1, 则启动一个类型为4的中断过程, 即相当于执行指令： INT 4
  - 若OF=0, 不做任何操作执行下一条指令。
- INTO指令通常安排在带符号数加减运算指令之后判断是否发生溢出。



3. 中断返回指令

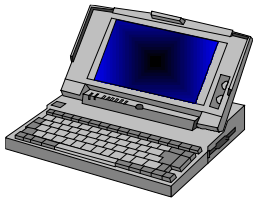
- 格式：

  IRET

- 中断服务程序的最后一条指令，执行：

  恢复断点

  恢复标志寄存器内容



六、处理器控制指令

- 对标志位的操作
- 与外部设备的同步

具体介绍见讲义108页

对标志位操作都是无操作数指令，操作数隐含为FLAGS的某个标志位。可操作的标志位有CF、IF和DF。

（1）清除进位标志 CLC

（2）置1进位标志STC

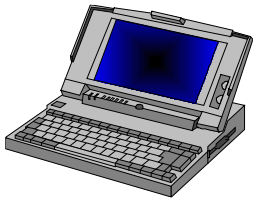
（3）进位标志取反CMC

（4）清除方向标志CLD

（5）置1方向标志STD

（6）清除中断标志CLI

（7）置1中断标志STI



作业

- 习题三 3.8 (1)(2)(4)(6)(8)(9)(12), 3.9 3.12, 3.13, 3.14(3)(5)(8), 3.17, 3.19