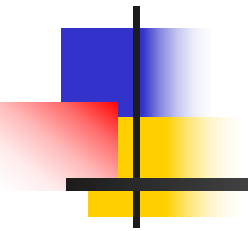
第3章 指令系统

主要内容：

- 基本概念
- 对操作数的寻址方式
- 六大类指令的操作原理：
  - 操作码的含义
  - 指令对操作数的要求
  - 指令执行的结果



3.1 概述

了解:

- 指令及指令系统
- 指令的格式
- 指令中的操作数
- 指令字长与机器字长

1. 指令与指令系统

指令：

控制计算机完成某种操作的命令

指令系统：

处理器所能识别的所有指令的集合

指令的兼容性：

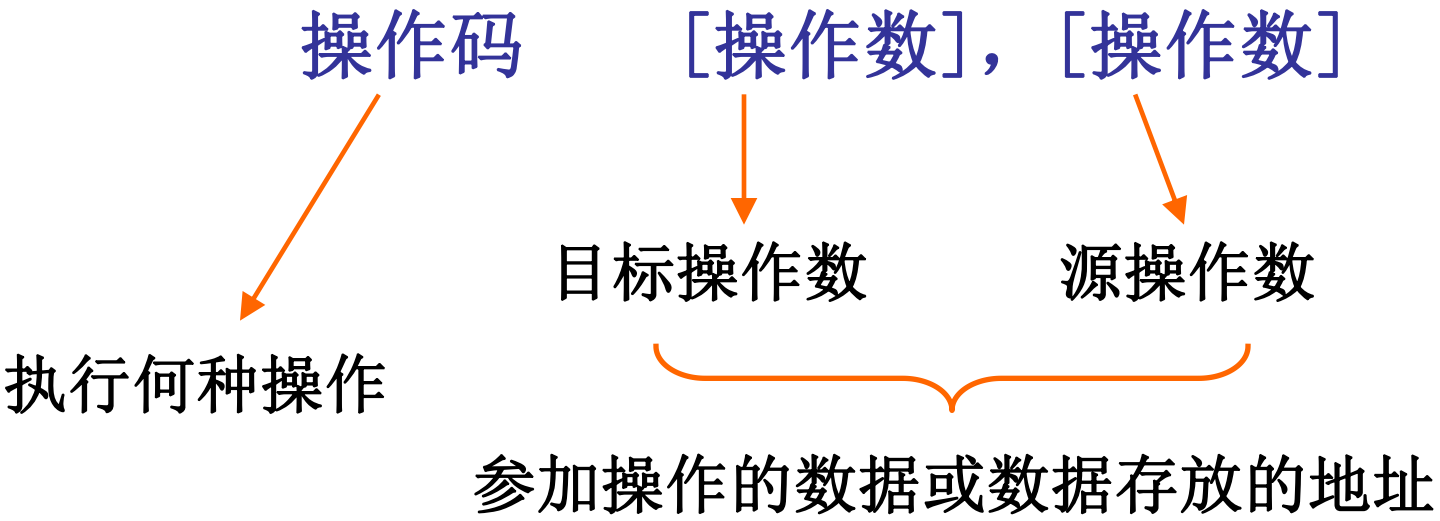
同一系列机的指令都是兼容的。

2. 指令格式

指令中应包含的信息：

- 运算数据的来源
- 运算结果的去向
- 执行的操作

指令格式



指令格式：

零操作数指令：　操作码

单操作数指令：　操作码　操作数

双操作数指令：　操作码　操作数，操作数

多操作数指令：　三操作数及以上

3. 指令中的操作数类型

- 立即数 → 表征参加操作的数据本身
- 寄存器、存储器 → 表征数据存放的地址

立即数操作数

- 立即数本身是参加操作的数据，可以是8位或16位，只能作为源操作数。
- 例：
  - MOV　AX，1234H
  - MOV　BL，22H

立即数无法作为目标操作数

立即数可以是无符号或带符号数，数值符合其取值范围

寄存器操作数

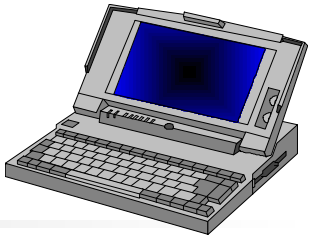
- 参加运算的数存放在指令给出的寄存器中，可以是16位或8位。
- 例：
  - MOV　AX，BX
  - MOV　DL，CH

存储器操作数

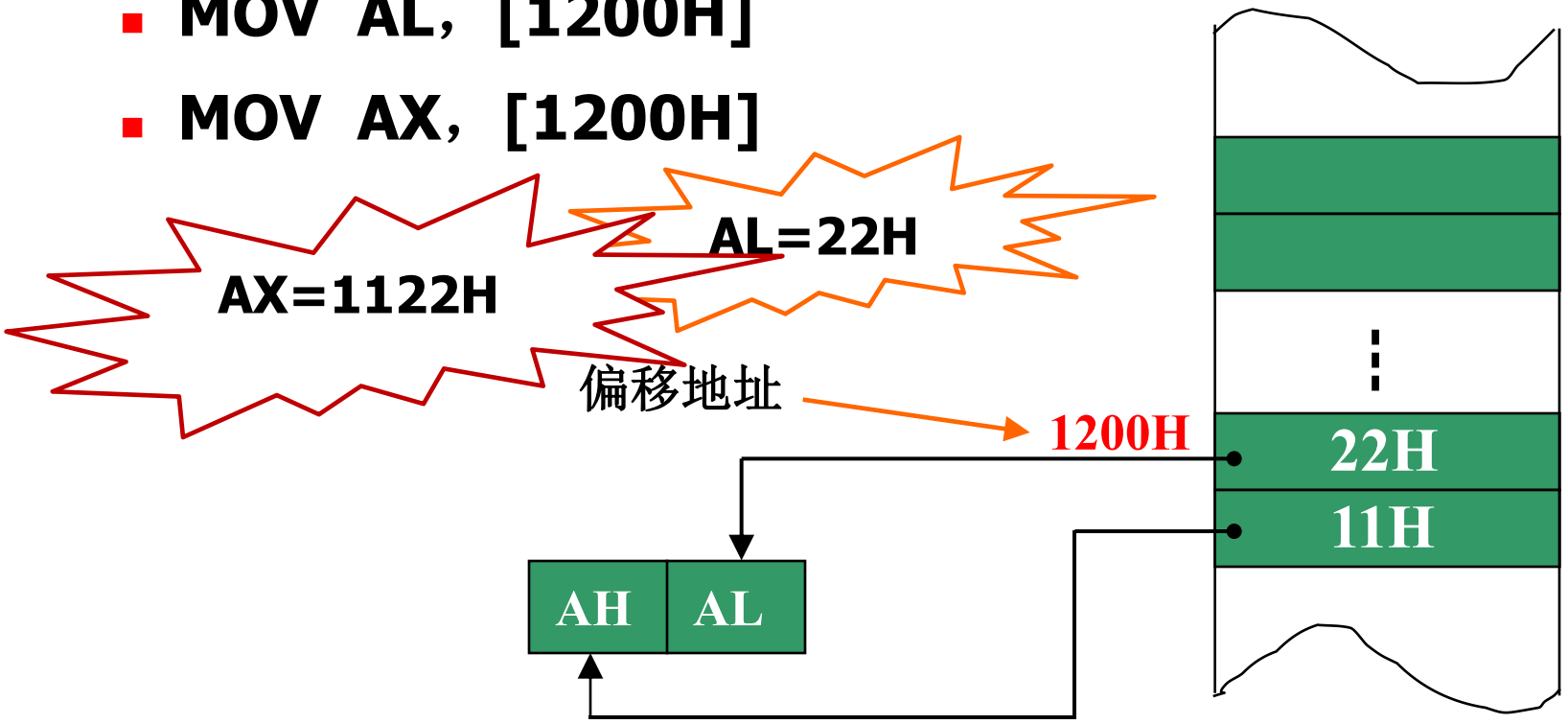
- 参加运算的数存放在存储器的某一个或某两个单元中
- 表现形式：[　　]

指令的操作数出现[　]，表示要寻址的数在内存中。

存储器操作数例

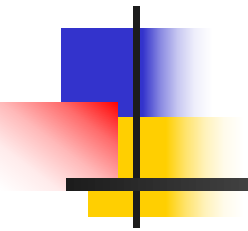


- 例：
  - MOV AL，[1200H]
  - MOV AX，[1200H]



三种类型操作数的比较

- 立即数：
  - 由指令直接给出
  - 无地址含义，只表示运算的数据
  - 立即数不能作为目标操作数
- 寄存器操作数
  - 表示运算的数据存放在寄存器中
  - 多数情况下，寄存器操作数指通用寄存器
  - 在三类操作数中所需运行时间最短
- 存储器操作数
  - 表示运算的数据存放在内存
  - 数据所在单元的地址是指令中“[ ]”里的值
  - 在三类操作数中所需运行时间最长。



3.2 寻址方式

寻址方式

寻找操作数所在地址的方法

操作数在哪里？？

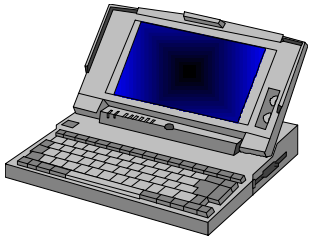
寻址方式

- 操作数可能的来源或运算结果可能的去处：
  - 由指令直接给出
  - 寄存器
  - 内存单元
- 寻找操作数所在地址的方法可以有三种大类型
  - 指令直接给出的方式
  - 存放于寄存器中的寻址方式
  - 存放于存储器中的寻址方式

寻址方式

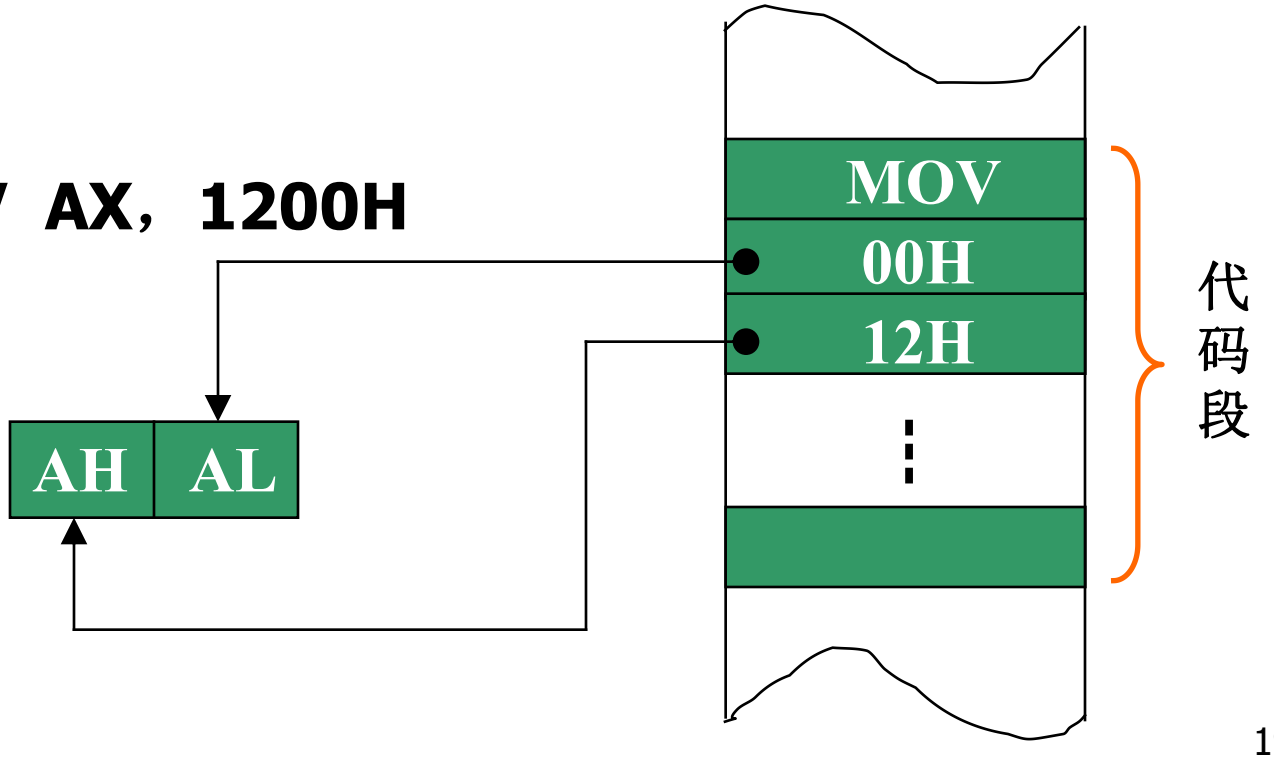
- 指令直接给出的方式
  - 运算对象由指令直接给出
- 存放于寄存器中的寻址方式
- 存放于存储器中的寻址方式
- 隐含给出方式

1，立即寻址



- 立即寻址仅适合于源操作数

- 例：
  - MOV AX，1200H



2，寄存器寻址

- 参加操作的操作数在CPU的通用寄存器中。
- 例：
  - **MOV AX，BX**

此种寻址方式中的寄存器主要是通用寄存器

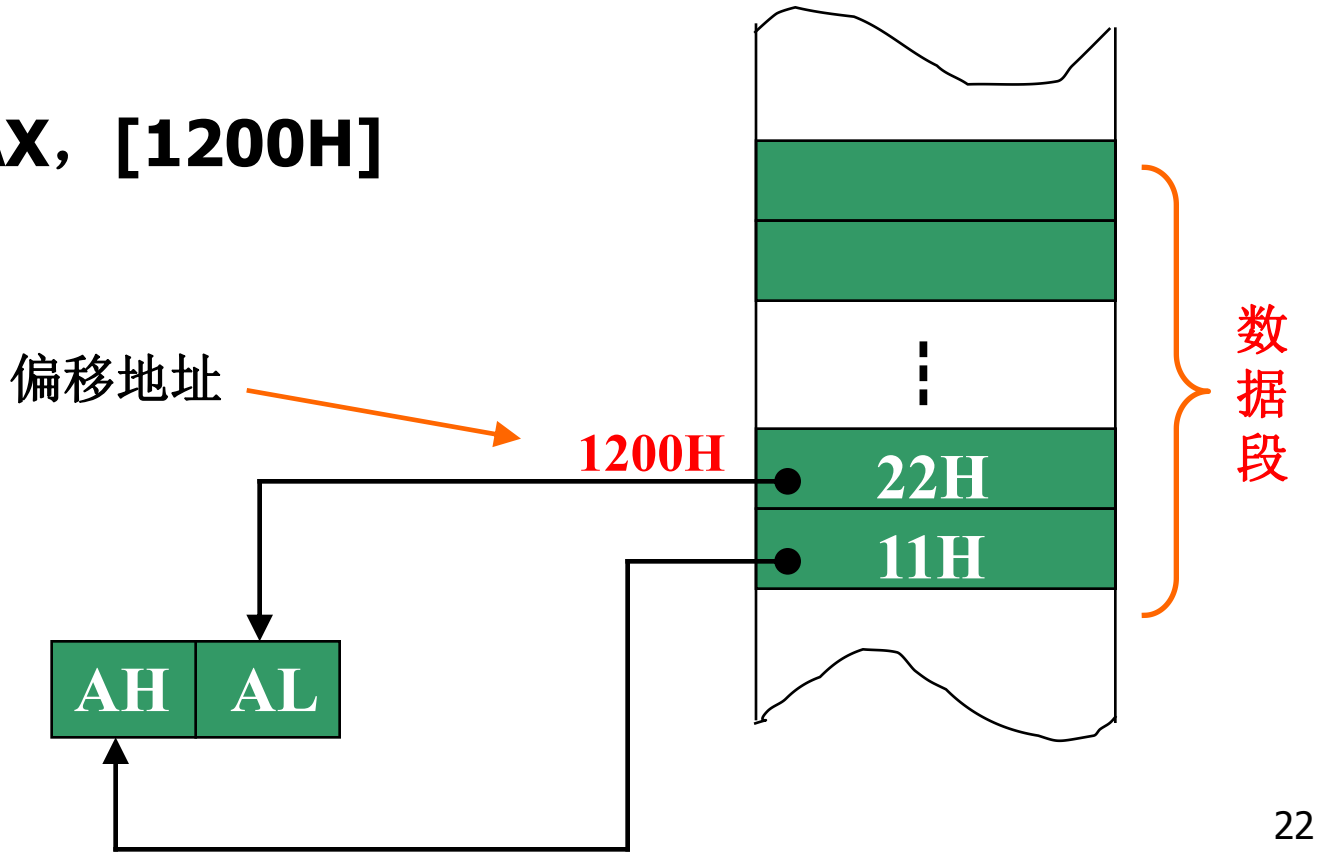
不含控制寄存器，段寄存器限制使用

存储器操作数的寻址方式

- 关注点：
  - 指令操作的对象在内存中，表现形式为：
    - [　　]
  - **操作数的字长取决于指令中的另一个寄存器操作数**，或通过其他方式指定字长（8bit/16bit）
  - 指令中给出运算对象在内存某个逻辑段中的偏移地址
    - [　偏移地址　　]
  - 逻辑段的段基地址通过默认或重设方式给出。

3，直接寻址

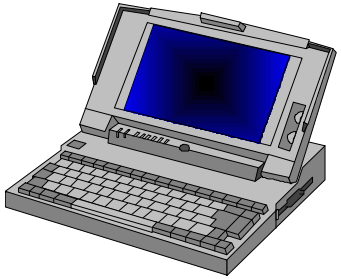
- 指令中直接给出操作数的偏移地址
- 例：
  - **MOV AX，[1200H]**



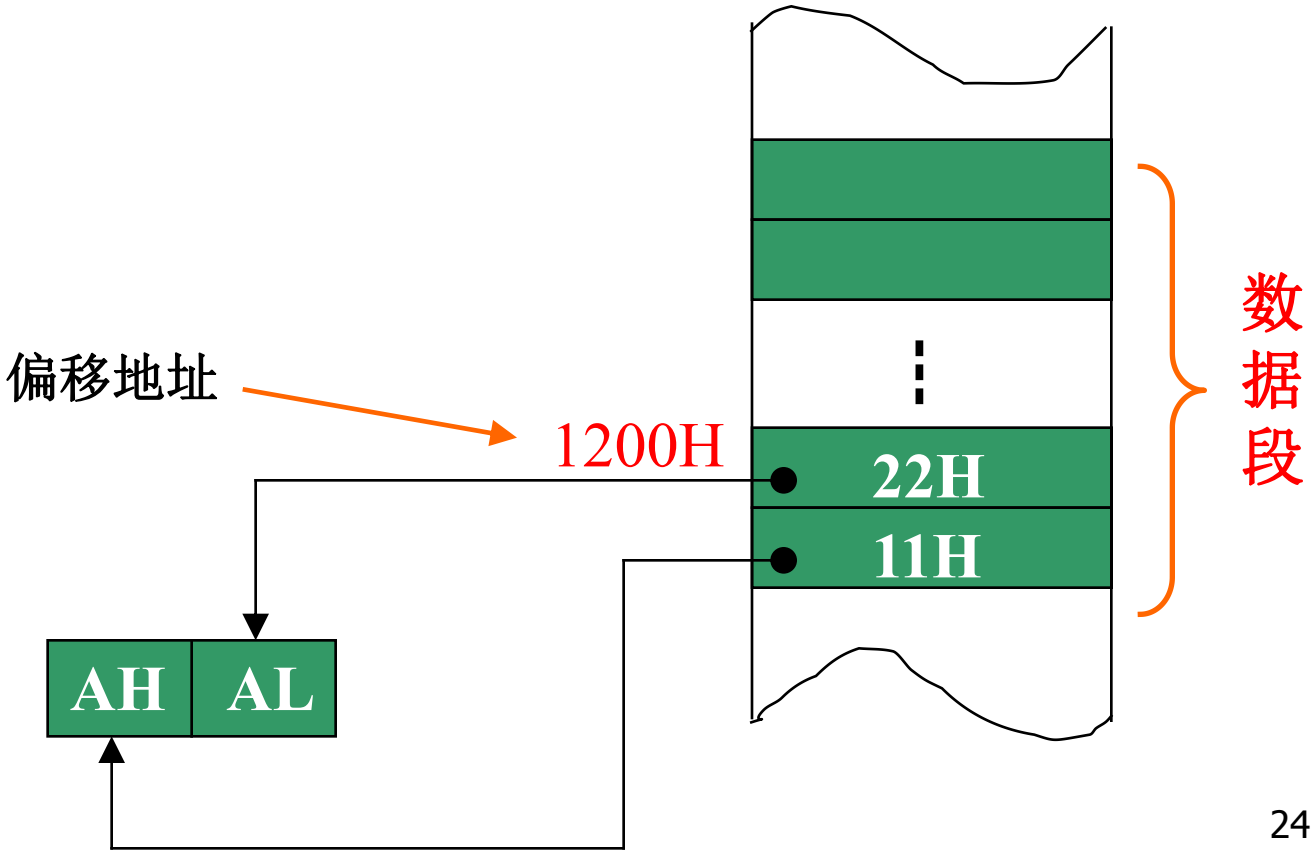
直接寻址

- 直接寻址方式下，**操作数默认为在数据段**，但允许段重设，即由指令给出所在逻辑段。
- 例：
  - **MOV AX，ES：[1200H]**

字节序



- 例：MOV AX，[1200H]



字节序

- 0x12345678
- Little Endian

低地址 高地址

---------->

| 78 | 56 | 34 | 12 |

Big Endian

低地址 高地址

---------->

| 12 | 34 | 56 | 78 |

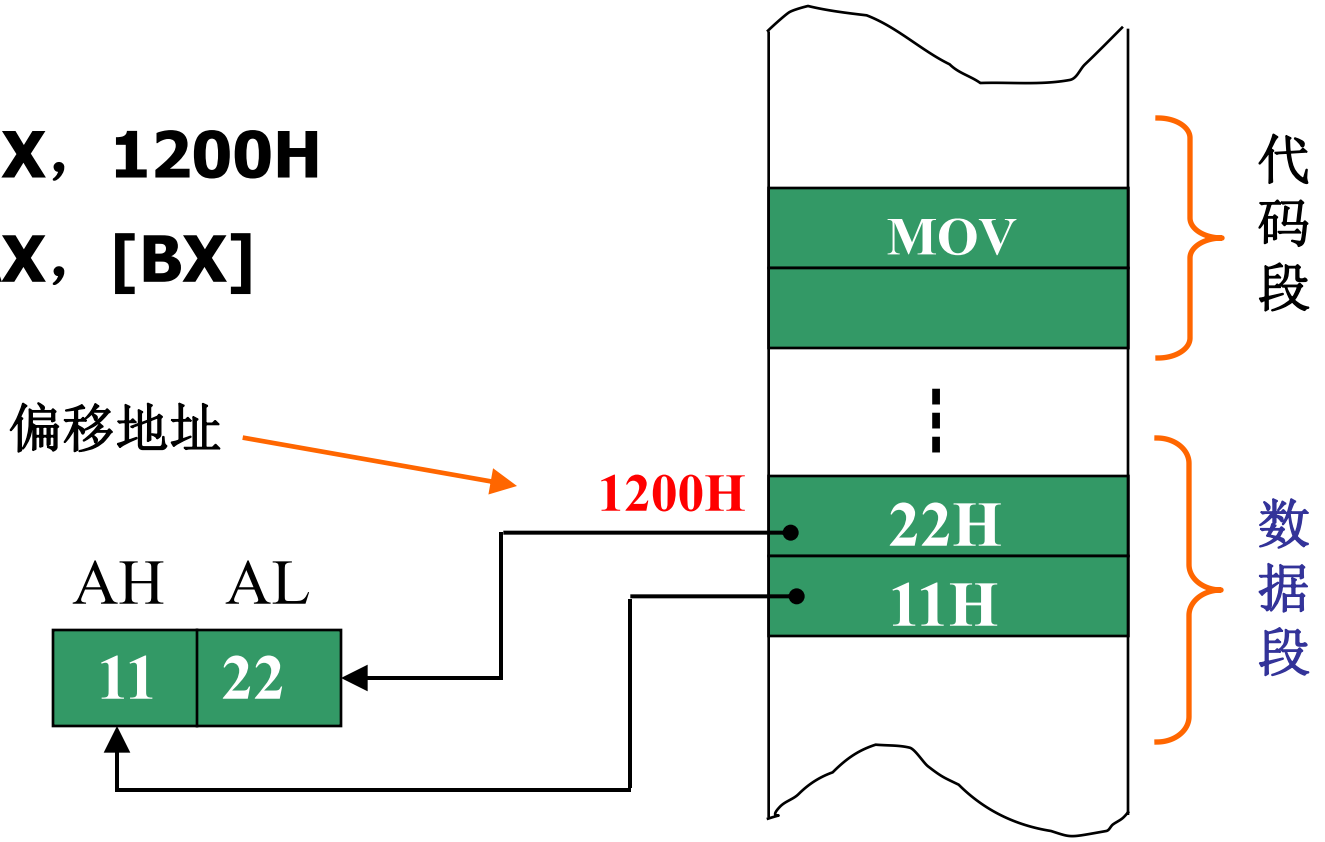
4，寄存器间接寻址

- 操作数存放在内存中，其偏移地址为方括号中通用寄存器的内容。
  - [间址寄存器]

BX，BP，SI，DI

寄存器间接寻址例

- 例：
  - MOV BX，1200H
  - MOV AX，[BX]



寄存器间接寻址

- 由寄存器间接给出操作数的偏移地址；
- 存放偏移地址的寄存器称为间址寄存器，它们是：**BX，BP，SI，DI**
- 操作数的段地址（数据处于哪个段）取决于选择哪一个间址寄存器：

BX，SI，DI → 默认在数据段

BP → 默认在堆栈段

寄存器间接寻址

- 寄存器间接寻址
  - 基址寻址
    - 间址寄存器为基址寄存器BX，BP
  - 变址寻址
    - 间址寄存器为变址寄存器SI，DI

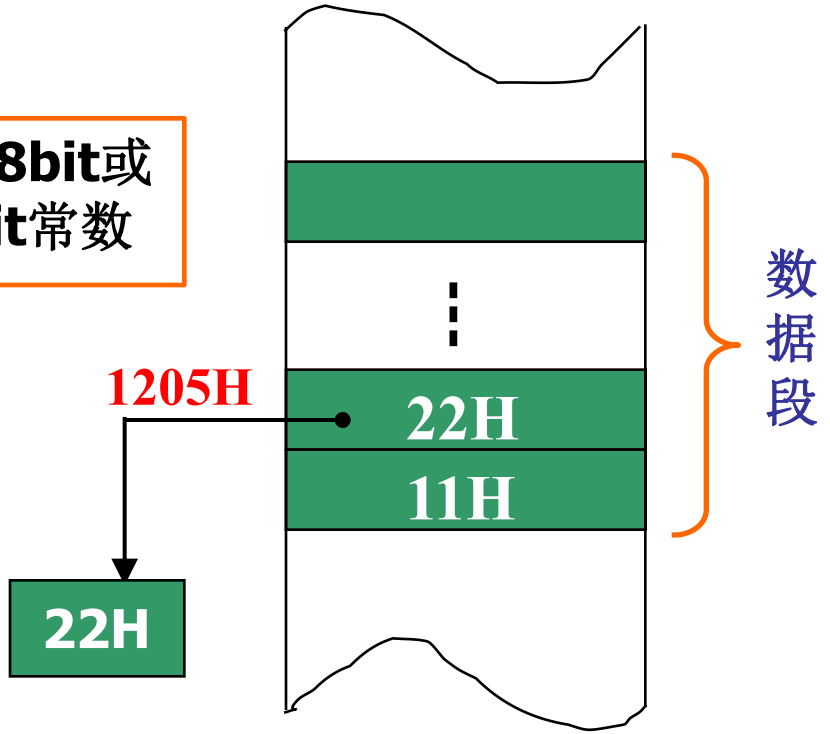
5，寄存器相对寻址

段地址由所选间址寄存器决定

- 操作数的偏移地址为寄存器的内容加上一个位移量。
- 如：
  - MOV AX，[BX+DATA]
- 例：
  - MOV AX，2000H
  - MOV DS，AX
  - MOV BX，1200H
  - MOV AL，[BX]5

任意8bit或16bit常数

相当于[BX+5]



相对寻址主要用于一维数组的操作

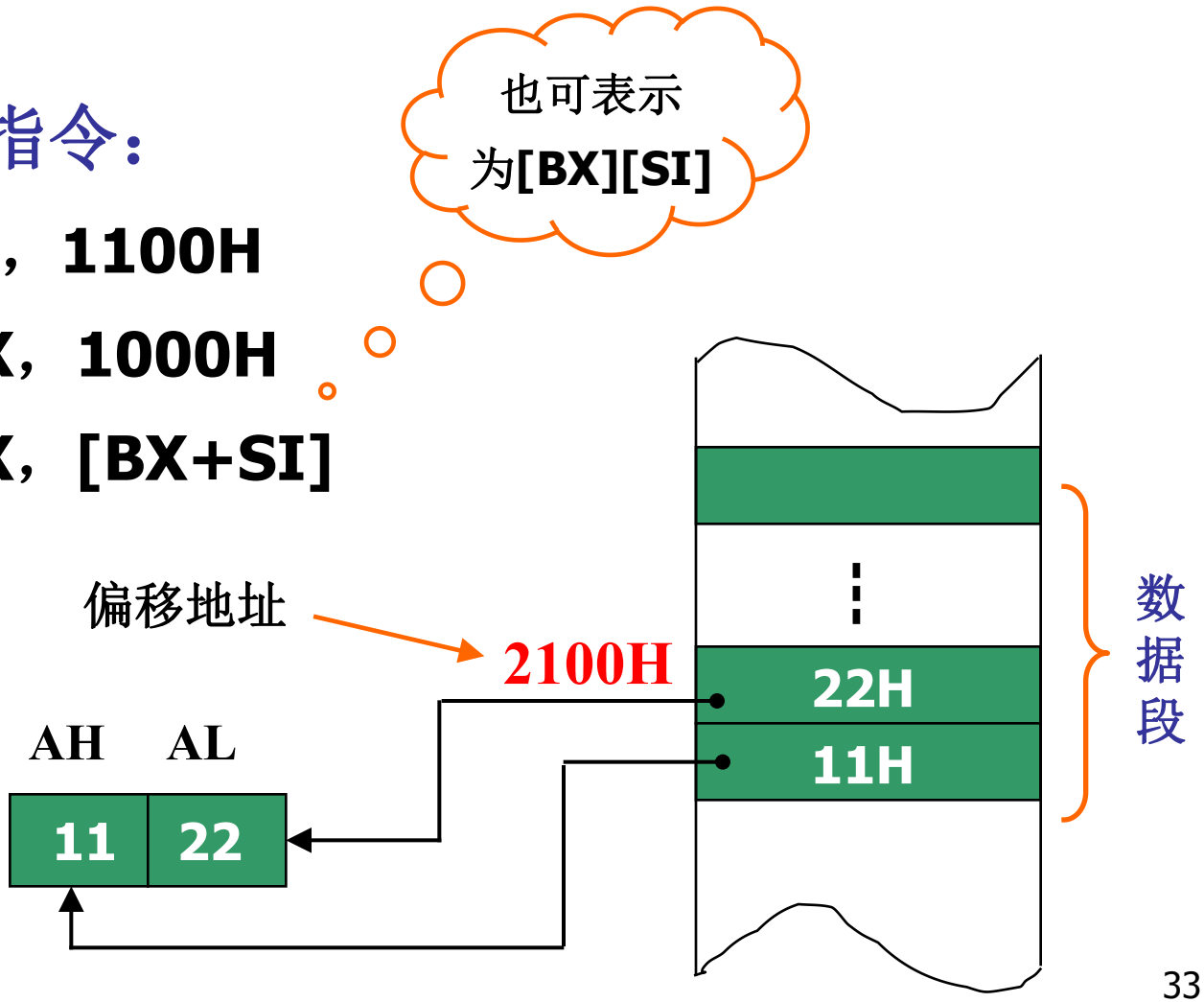
常将位移量作为“表头”地址，间址寄存器的值作为表内相对地址

6， 基址—变址寻址

- 操作数的偏移地址为
  - 一个基址寄存器的内容 + 一个变址寄存器的内容；
- 操作数的**段地址由选择的基址寄存器决定**
  - 基址寄存器为BX，默认在数据段
  - 基址寄存器为BP，默认在堆栈段
- 基址变址寻址方式与相对寻址方式一样，主要用于一维数组操作。

例：

- 执行下列指令：
  - MOV SI，1100H
  - MOV BX，1000H
  - MOV AX，[BX+SI]

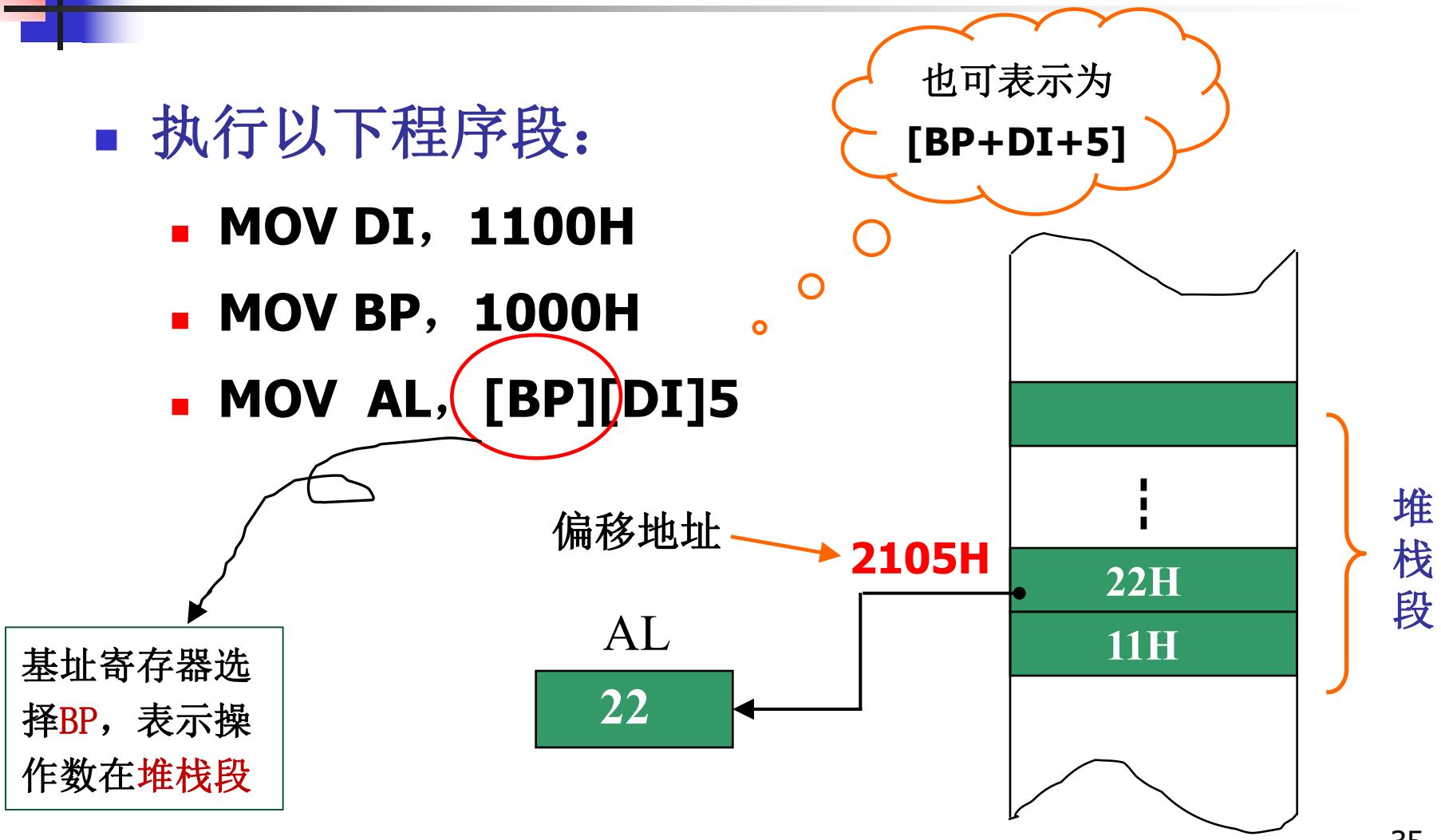


7，基址-变址-相对寻址

- 操作数的偏移地址为：
  - 基址寄存器内容+变址寄存器内容+位移量
- 操作数的段地址由选择的基址寄存器决定。
- 基址变址相对寻址方式主要用于二维表格操作。

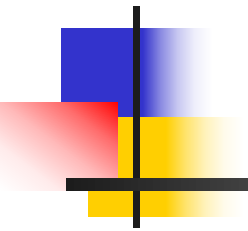
例：

- 执行以下程序段：
  - **MOV DI，1100H**
  - **MOV BP，1000H**
  - **MOV AL，[BP][DI]5**



8，隐含寻址

- 指令中隐含了一个或两个操作数的地址，即操作数在默认的地址中。
- 例：
  - **MUL BL**
- 指令执行：
  - **AL×BL** → **AX**



3.3 8086指令系统

掌握：

- 指令码的含义
- 指令对操作数的要求
- 指令的对标志位的影响
- 指令的功能

注：指令中的常用符号见教材P108

8086指令系统

从功能上包括六大类：

- 数据传送
- 算术运算
- 逻辑运算和移位
- 串操作
- 程序控制
- 处理器控制

3.3.1 数据传送指令

- 通用数据传送
- 输入输出
- 地址传送
- 标志位操作

1. 通用数据传送

- 一般数据传送指令
- 堆栈操作指令
- 交换指令
- 查表转换指令
- 字位扩展指令

该类指令的执行对标志位不产生影响

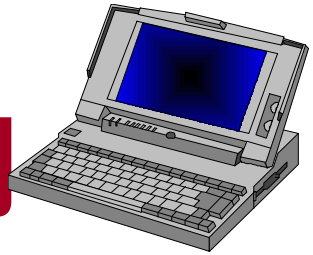
1）一般数据传送指令

- 一般数据传送指令 MOV
- 格式：
  - MOV dest，src
- 操作：
  - **src** → **dest**
- 例：
  - MOV AL，BL

一般数据传送指令

- 注意点：
  - 两操作数字长必须相同；
  - 两操作数不允许同时为存储器操作数；
  - 在源操作数是立即数时，目标操作数不能是段寄存器；
  - 两操作数不允许同时为段寄存器；
  - IP和CS不作为目标操作数
  - FLAGS一般也不作为操作数在指令中出现。

常见错误的数据传送指令例



- MOV 3, SI　　立即数不能作目标操作数
- MOV CH, 1234H　　CH是8位的，不可以传送16位
- MOV AX, CL　目标和源的位数不符
- MOV X, [100H]　　X不是寄存器
- MOV [100H], [DI] 8086不支持源操作数和目的操作数同时访问内存
- MOV DS, 1000H　段寄存器不能给立即数
- MOV CS, AX　代码段寄存器不能被赋值
- MOV [AX], BX 寄存器间接寻址只能使用BX, BP, SI, DI 。

注意

- 编程

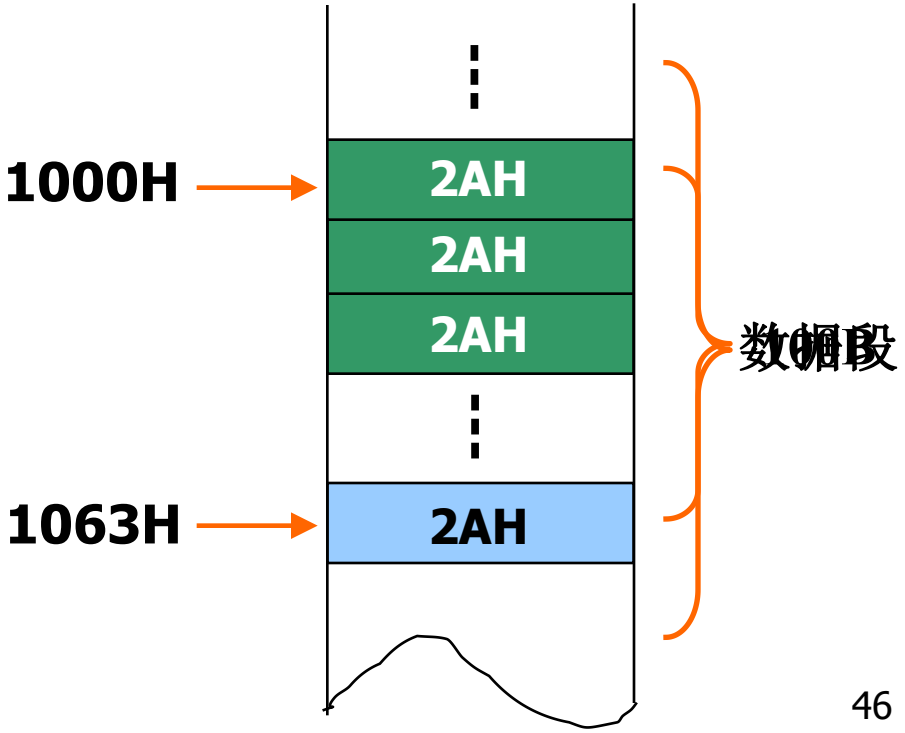
  加入BYTE　PTR或WORD　PTR定义

- 查错

  强制转换->寄存器->地址[寄存器] ->立即数

一般数据传送指令应用例

- 将(*)的ASCII码2AH送入内存数据段1000H开始的100个单元中。
- 题目分析：
  - 确定首地址
  - 确定数据长度
  - 写一次数据
  - 修改单元地址
  - 修改长度值
  - 判断写完否？
  - 未完继续写入，否则结束



一般数据传送指令应用例

程序段：

```
        MOV  DI，1000H
        MOV  CX，64H
        MOV  AL，2AH
AGAIN：MOV  [DI]，AL
        INC  DI              ；DI+1
        DEC  CX              ；CX-1
        JNZ  AGAIN           ；CX≠0则继续
        HLT
```

上段程序在代码段中的存放形式

- 設CS=109EH，IP=0100H，则各条指令在代码段中的存放地址如下：

CS ： IP	机器指令	汇编指令
109E：0100	B80010	MOV DI，1000H
109E：0103	.	MOV CX，64H
109E：0105	.	MOV AL，2AH
109E：0107	.	MOV [DI]，AL
109E：0109		INC DI
109E：010A		DEC CX
109E：010B		JNZ 0107H
109E：010D		HLT

数据段中的分布

- 送上2AH后数据段中相应存储单元的内容改变如下：

DS：1000　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A

DS：1010　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A

DS：1020　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A

DS：1030　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A

DS：1040　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A

DS：1050　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A

DS：1060　2A 2A 2A 2A 00 00 00 00 00 00 00 00 00 00 00 00

偏移地址[DI]

2）堆栈操作指令

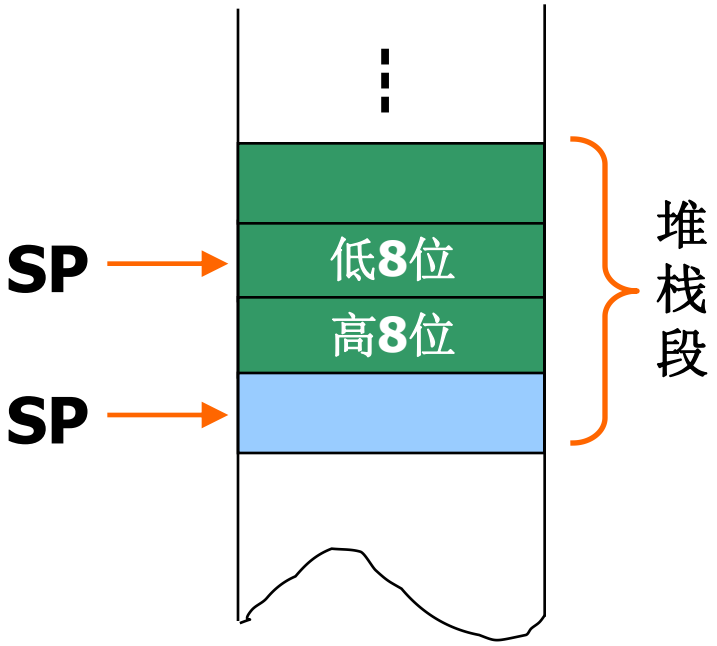
- 堆栈操作的原则
  - 先进后出
  - 以字为单位
- 堆栈操作指令：
  - 压栈指令
    - 格式：　PUSH　OPRD
  - 出栈指令
    - 格式：　POP　OPRD

16位寄存器或
存储器两单元

压栈指令 PUSH

- 指令执行过程：
  - SP － 2 → SP
  - 操作数高字节 → SP+1
  - 操作数低字节 → SP

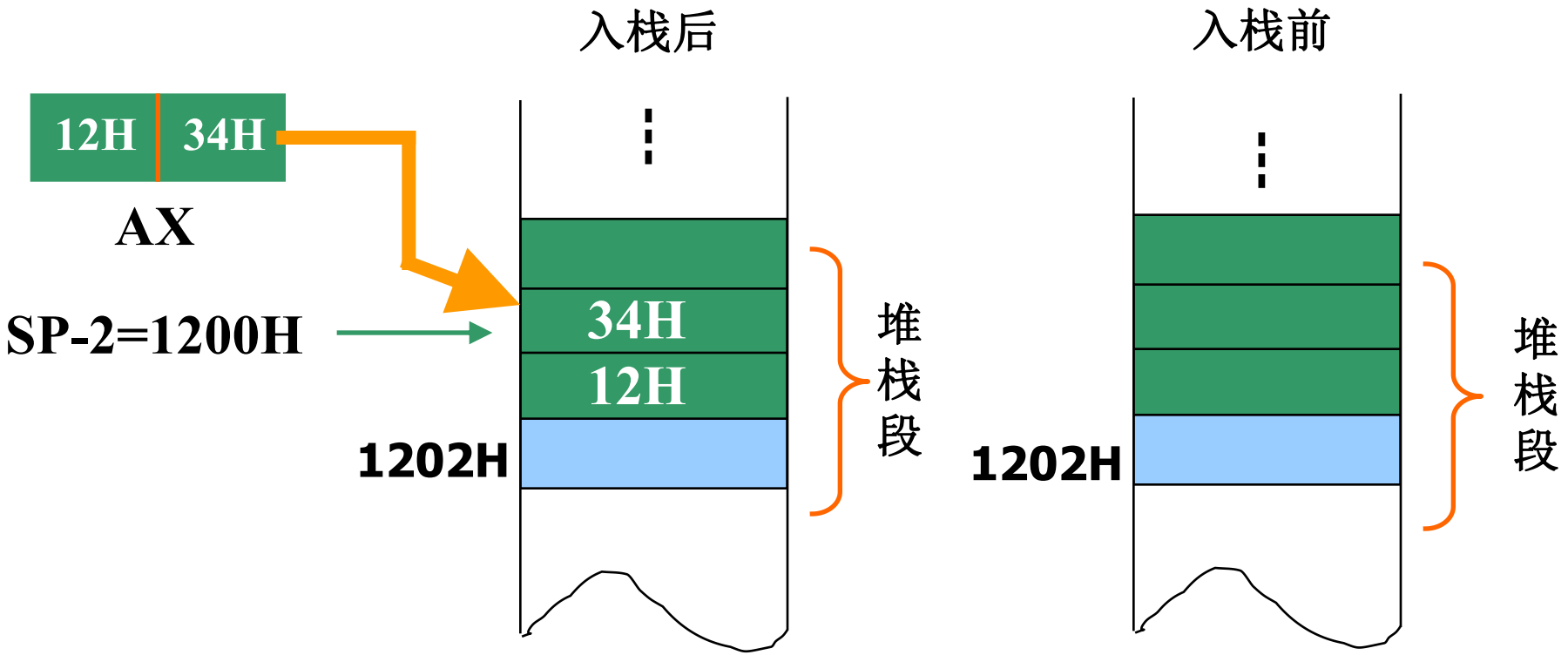
所谓：高字节放高地址，
低字节放低地址。



压栈指令的操作

设AX=1234H，SP=1202H

执行 PUSH AX 指令后堆栈区的状态：



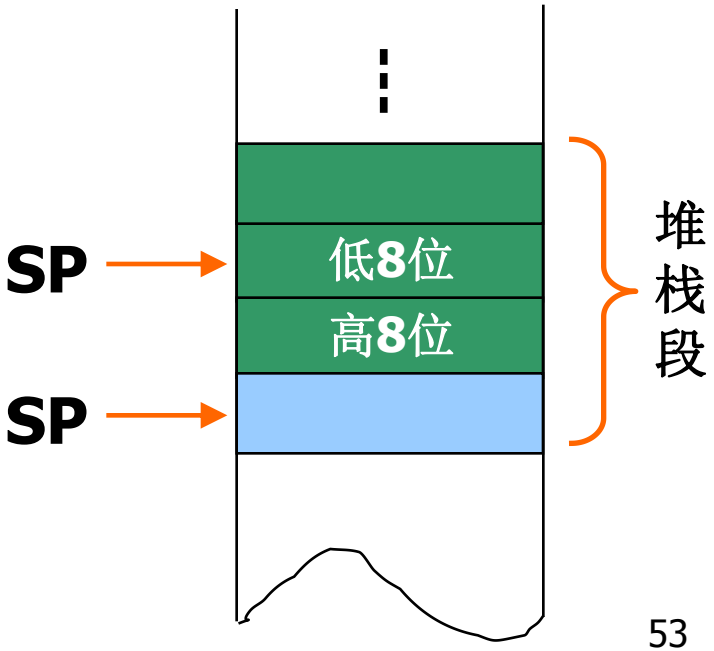
出栈指令POP

- 指令执行过程：

SP → 操作数低字节

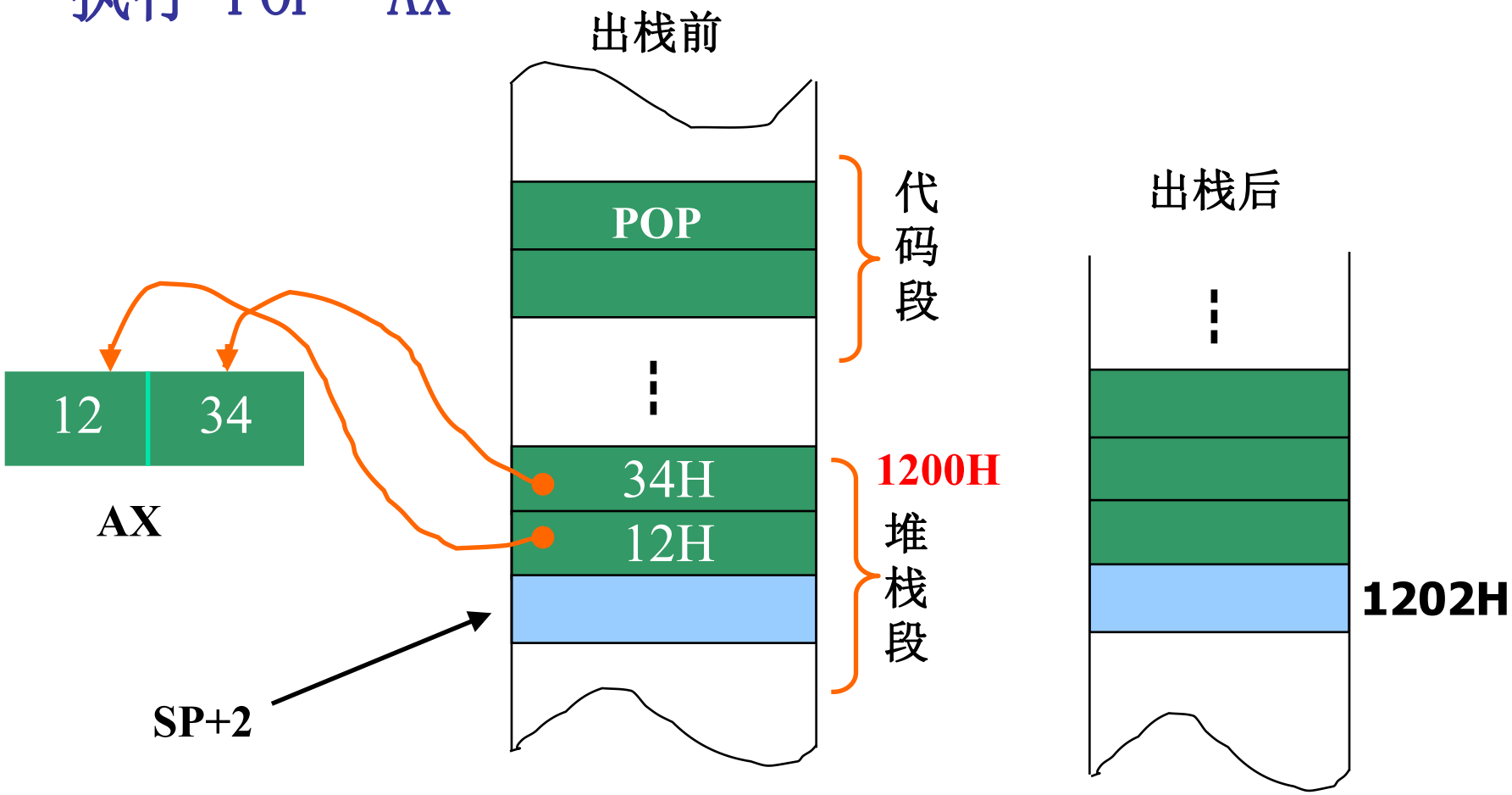
SP+1 → 操作数高字节

SP←SP+2



出栈指令的操作

执行 POP AX



堆栈操作指令说明

- 指令的操作数必须是16位；
- 操作数可以是寄存器或存储器两单元，但不能是立即数；
- 不能从栈顶弹出一个字给CS；
- PUSH和POP指令在程序中一般成对出现；
- PUSH指令的操作方向是从高地址向低地址，而POP指令的操作正好相反。

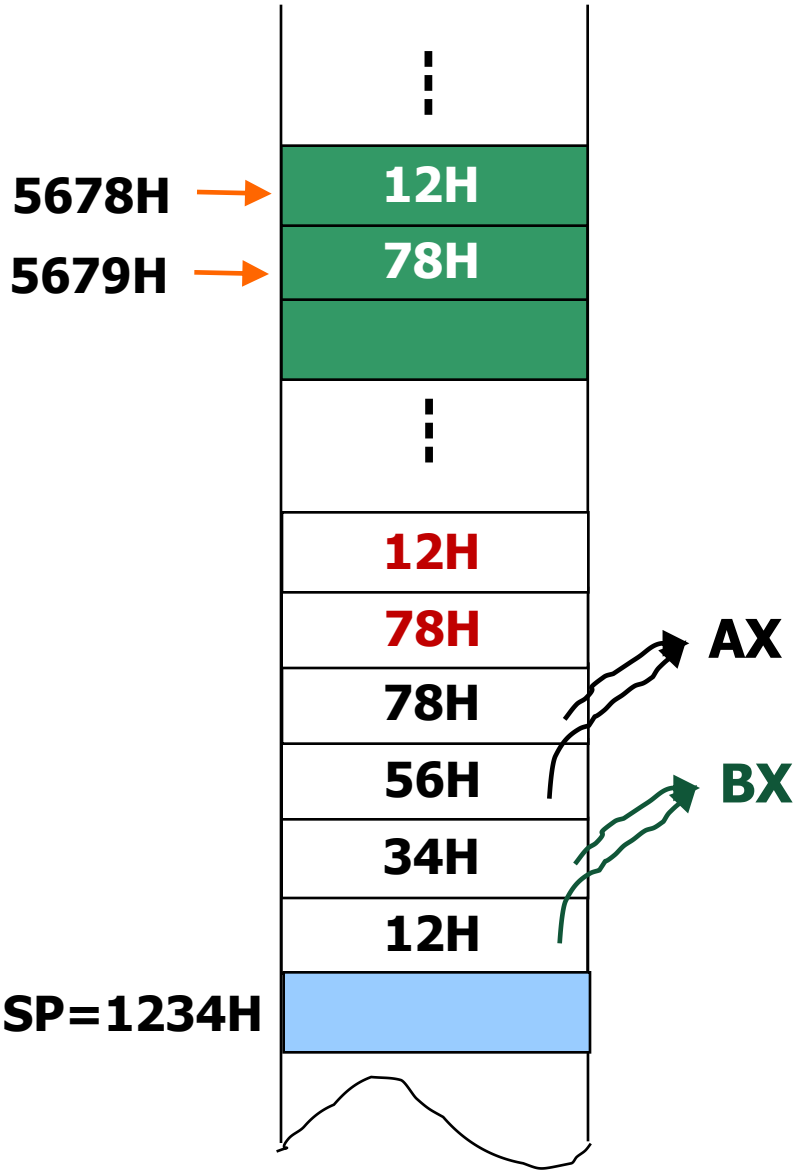
堆栈操作指令例

- MOV AX，1234H
- MOV SP，AX
- MOV BX，5678H
- MOV [BX]，AH
- MOV [BX+1]，BL
- PUSH AX
- PUSH BX
- PUSH WORD PTR[BX]

⋮

- POP WORD PTR[BX]
- POP AX
- POP BX

如此，会使AX和BX的内容互换



3) 交换指令

- 格式：
  - **XCHG OPRD1，OPRD2**
- 注：
  - 操作数可以是寄存器或存储器，不能为立即数。源操作数与目标操作数长度必须相等。
  - 两操作数不能同时为存储器。
  - 不允许使用段寄存器。
- 例：**XCHG AX，BX**
  - **XCHG [2000]，CL**

4) 查表指令XLAT

- 格式：
  - **XLAT** ；将偏移地址为**BX+AL**的所在单元的内
  - 容送到寄存器**AL**中。
- 注：
- 例： **MOV BX , 1000H**
- **MOV AL , 8**
  XLAT

字位扩展指令

- 将符号数的符号位扩展到高位；
- 指令为零操作数指令，采用隐含寻址，隐含的操作数为AX及AX，DX
- 无符号数的扩展规则为在高位补0

字节到字的扩展指令

- 格式：
  - CBW
- 操作：
  - 将AL内容扩展到AX
- 规则：
  - 若最高位=1，则执行后AH=FFH
  - 若最高位=0，则执行后AH=00H

字到双字的扩展指令

- 格式：
  - CWD
- 操作：
  - 将AX内容扩展到DX　AX
- 规则：
  - 若最高位=1，则执行后DX=FFFFH
  - 若最高位=0，则执行后DX=0000H

2. 输入输出指令

掌握：

- 指令的格式及操作
- 指令的两种寻址方式
- 指令对操作数的要求

输入输出指令

- 专门面向I/O端口操作的指令
- 指令格式：
  - 输入指令： IN acc，PORT
  - IN acc, DX
  - 输出指令 ：OUT PORT，acc
  - OUT DX, acc

指令寻址方式

- 根据端口地址码的长度，指令具有两种不同的端口地址表现形式。
- 直接寻址
  - 端口地址为8位时，指令中直接给出8位端口地址；
  - 寻址256个端口。
- 间接寻址
  - 端口地址为16位时，指令中的端口地址必须由DX指定；
  - 可寻址64K个端口。

I/O指令例

- IN AX，80H ；从80H端口读入16bit数据到AX
- MOV DX，2400H
- IN AL，DX ；从2400H端口读入8bit数据到AL
- OUT 35H，AX ；将AX的值写入到35H端口中
- OUT DX，AL

小结

- 例：

① MOV　SI，100

② MOV　DX，03F8H

③ IN　AL，DX

④ 如果AL的最高位=0，则转向③，否则继续下一步

⑤ MOV　AX，[SI]

⑥ OUT　58H，AX

实际程序设计中，通常不直接给出偏移地址，而是采用符号地址。

程序功能？

3. 地址传送指令

- 取偏移地址指令LEA
- *LDS指令
- *LES指令

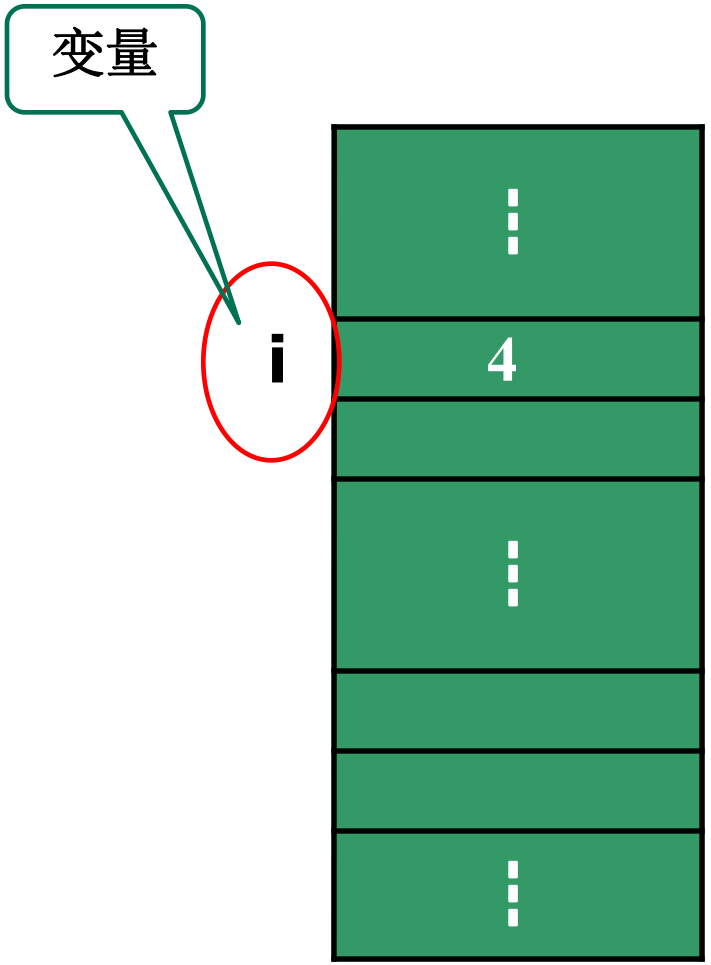
取偏移地址指令LEA

- 操作：
  - 将变量的16位偏移地址取出送目标寄存器
- 当程序中用符号表示内存偏移地址时，须使用该指令。
- 格式：

  LEA REG，MEM
- 指令要求：
  - 源操作数必须是一个存储器操作数，目标操作数通常是间址寄存器。

LEA指令与MOV指令执行结果对比

- MOV指令的执行结果：
  - 内存某单元中的内容
  - 如：
    - MOV　AL，i
    - 结果：AL=4
- LEA指令的执行结果：
  - 内存某单元的偏移地址
    - 获得i值本身
  - 如：
    - LEA　BX，i
    - 结果：BX=i



LEA指令与MOV指令执行结果对比

- 比较下列指令：

符号
地址

```
MOV   SI，DATA1
```

执行结果：

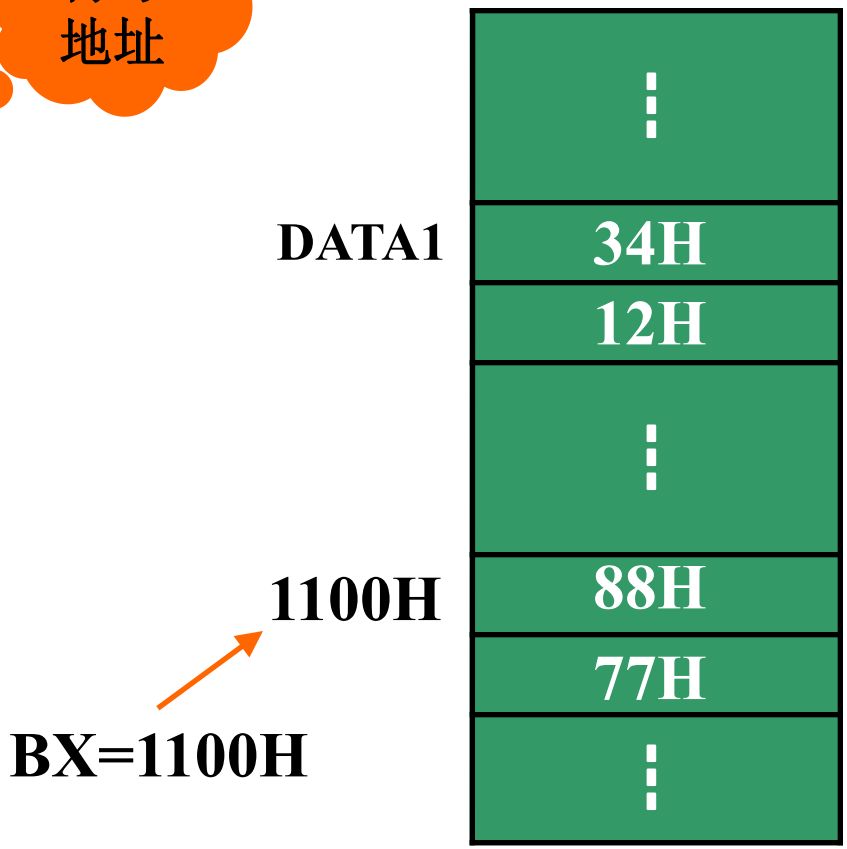
执行结果：

SI=DATA1

执行结果：

执行结果：

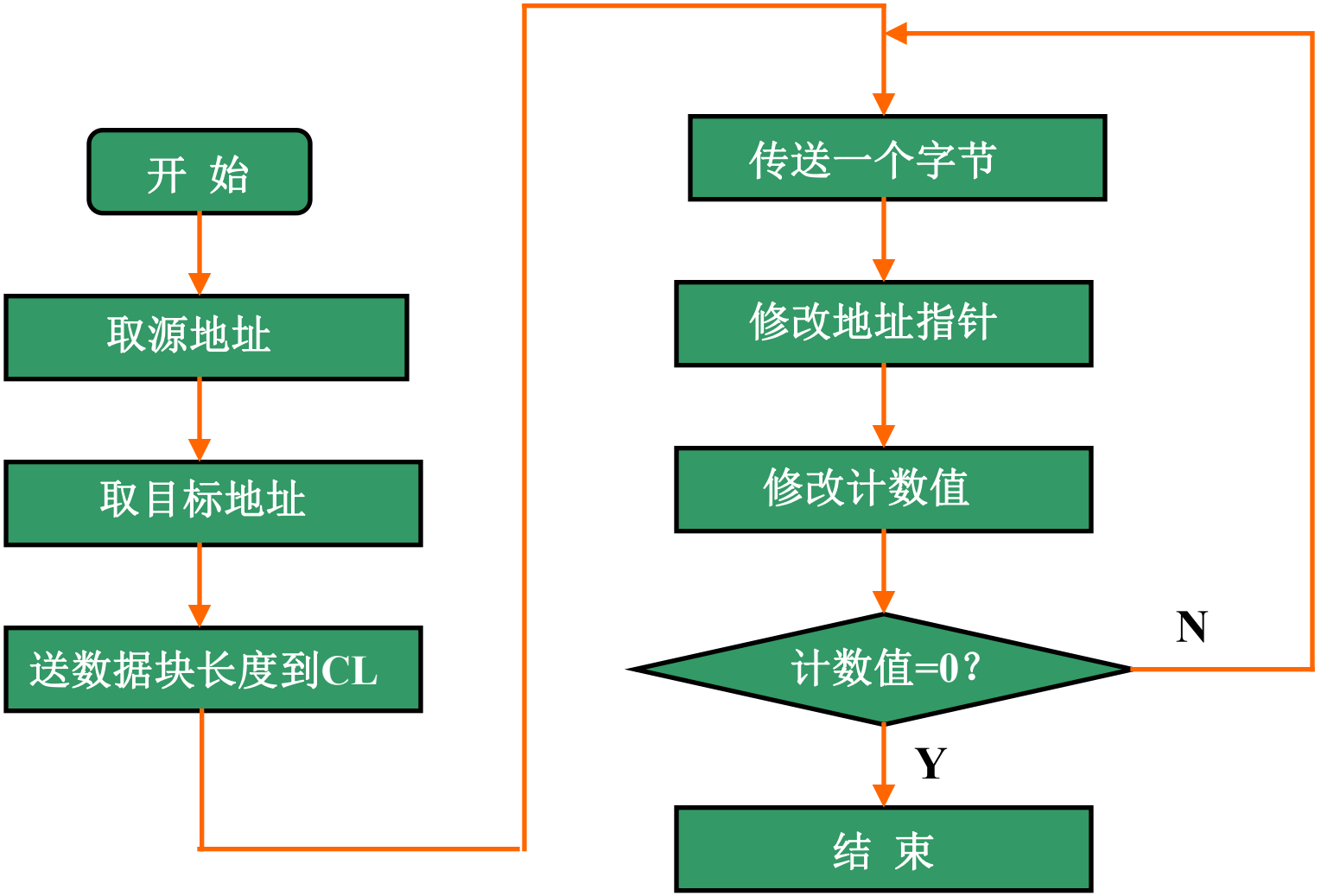
BX=1100H



LEA指令在程序中的应用

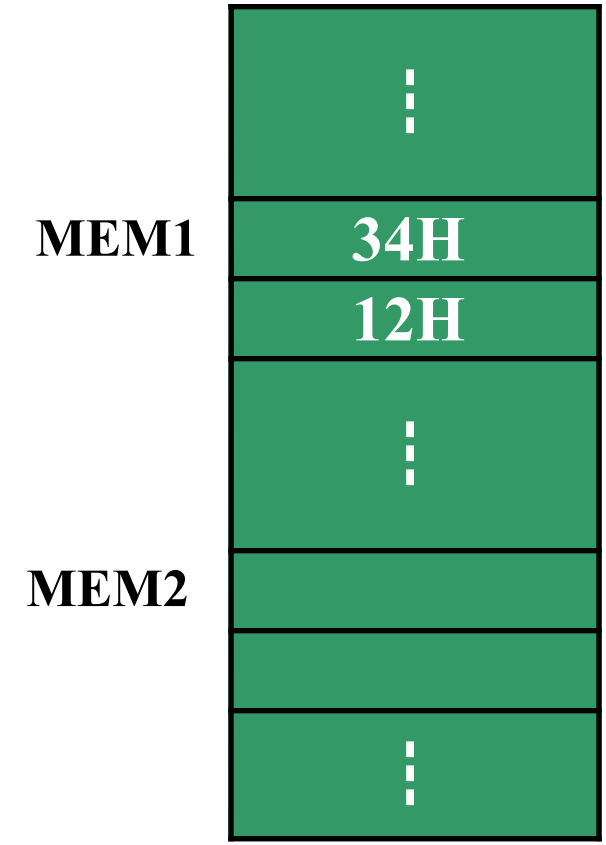
- 将数据段中首地址为MEM1 的50个字节的数据传送到同一逻辑段首地址为MEM2的区域存放。编写相应的程序段 。

LEA指令在程序中的应用



LEA指令在程序中的应用

```
        LEA  SI，MEM1
        LEA  DI，MEM2
        MOV  CL，50
NEXT：  MOV  AL，[SI]
        MOV  [DI]，AL
        INC   SI
        INC   DI
        DEC   CL
        JNZ   NEXT      ；CL ≠0则转NEXT
        HLT
```



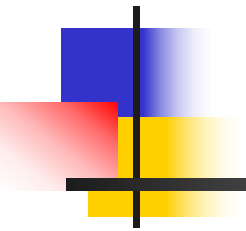
4. 其他传送指令　见教材P116

- LAHF
- SAHF

} 隐含操作数AH

- PUSHF
- POPF

} 隐含操作数FLAGS



3.3.2 算术运算类指令

算术运算类指令

- 加法运算指令
- 减法运算指令
- 乘法指令
- 除法指令

算术运算指令的执行大多对状态标志位会产生影响

1. 加法指令

- 普通加法指令ADD
- 带进位位的加法指令ADC
- 加1指令INC

加法指令对操作数的要求与MOV指令相同

1）ADD指令

- 格式：
  - ADD　OPRD1，OPRD2
- 操作：
  - OPRD1+OPRD2 ⟶ **OPRD1**

ADD指令的执行对全部6个状态标志位都产生影响

ADD指令例

```
MOV   AL，78H
ADD   AL，99H
```

指令执行后6个状态标志位的状态

ADD指令例

$$
\begin{array}{rr}
 & 0111\ 1000 \\
+ & 1001\ 1001 \\
\hline
\boxed{1} & 0001\ 0001
\end{array}
$$

标志位状态：

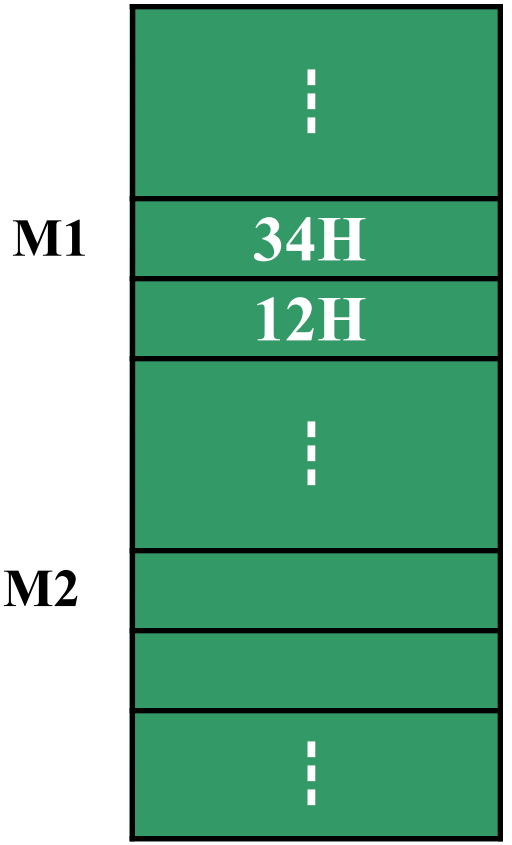
CF= 1	SF= 0
AF= 1	ZF= 0
PF= 1	OF= 0

2）ADC指令

- 指令格式、对操作数的要求、对标志位的影响与ADD指令完全一样
- 指令的操作：
  - OPRD1+OPRD2+CF → **OPRD1**
- ADC指令多用于多字节数相加，使用前要先将CF清零。

ADC指令应用例——求两个20B数之和

```
        LEA  SI，M1
        LEA  DI，M2
        MOV CX，20
        CLC                    ；使CF=0
NEXT：  MOV AL，[SI]
        ADC [DI]，AL
        INC SI
        INC DI
        DEC CX
        JNZ NEXT
        HLT
```



3）INC指令

- 格式：
  - INC　OPRD

不能是段寄存器
或立即数

- 操作：
  - OPRD+1 ⟶ **OPRD**

常用于在程序中修改地址指针

2. 减法指令

- 普通减法指令SUB
- 考虑借位的减法指令SBB
- 减1指令DEC
- 比较指令CMP
- 求补指令NEG

减法指令对操作数的要求与对应的加法指令相同

1）SUB指令

- 格式：
  - SUB　OPRD1，OPRD2
- 操作：
  - **OPRD1- OPRD2 ⟶ OPRD1**
- 对标志位的影响与ADD指令同

2）SBB指令

- 指令格式、对操作数的要求、对标志位的影响与SUB指令完全一样
- 指令的操作：
  - OPRD1- OPRD2- CF → **OPRD1**

3）DEC指令

- 格式：
  - DEC　OPRD
- 操作：
  - OPRD - 1 ⟶ **OPRD**

指令对操作数的要求与INC相同

指令常用于在程序中修改计数值

应用程序例

```
        MOV  BL，2
NEXT1：MOV  CX，0FFFFH
NEXT2：  DEC  CX
         JNZ  NEXT2          ；ZF=0转NEXT2
         DEC  BL
         JNZ  NEXT1          ；ZF=0转NEXT1
         HLT                 ；暂停执行
```

4）NEG指令

- 格式：
  - **NEG OPRD**

8/16位寄存器或
存储器操作数

- 操作：
  - **0 - OPRD ⟶ OPRD**

用0减去操作数，相当于对该操作数求补码

两点注意请
见教材p120

5）CMP指令

- 格式：
  - CMP　OPRD1，OPRD2
- 操作：
  - OPRD1- OPRD2

- 指令执行的结果不影响目标操作数，仅影响标志位！

CMP指令

- 用途：
  - 用于比较两个数的大小，可作为条件转移指令转移的条件
- 指令对操作数的要求及对标志位的影响与SUB指令相同

CMP指令

- 两个无符号数的比较：
  - **CMP AX，BX**
- 若 AX ≥ BX **CF=0**
- 若 AX < BX **CF=1**

CMP指令

- 两个带符号数的比较
  - **CMP AX，BX**

两个数的大小由OF和SF共同决定

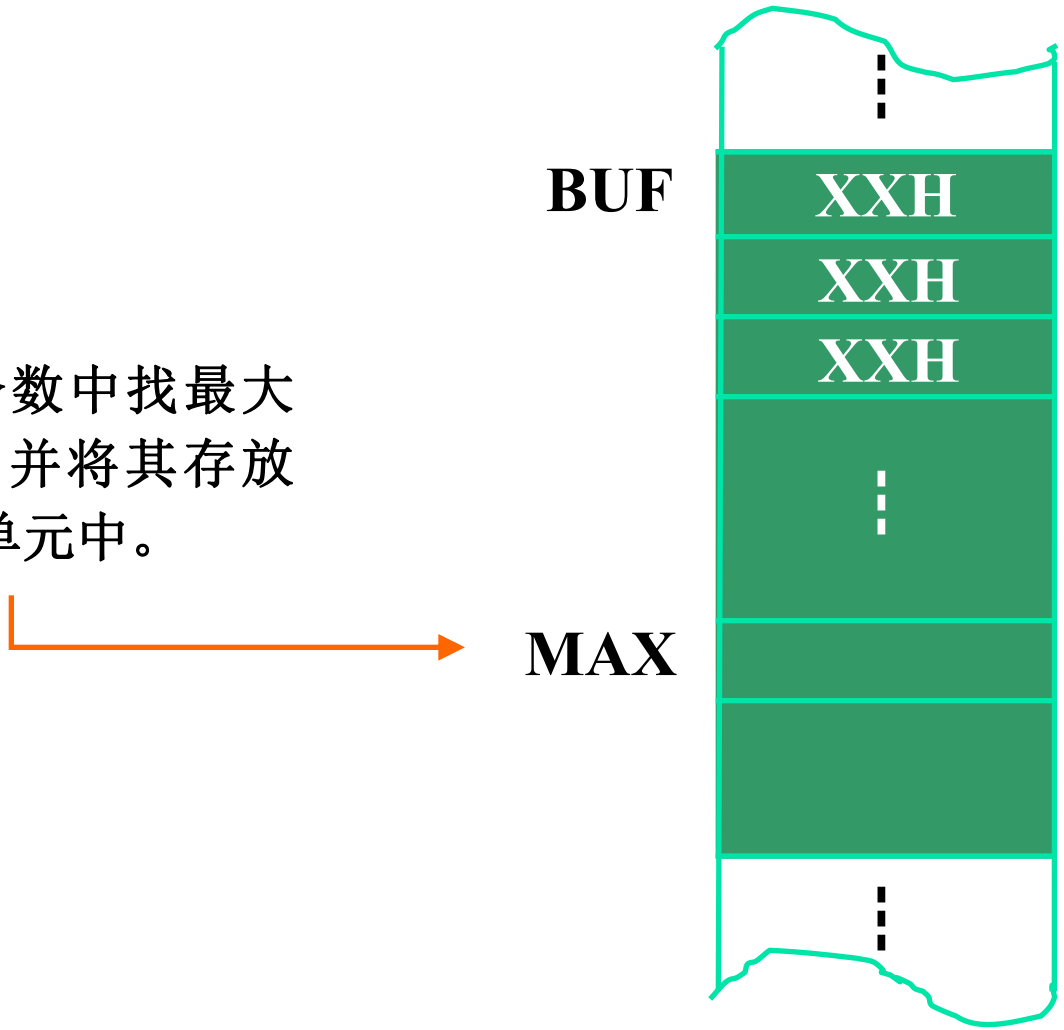
OF和SF状态相同	AX ≥BX
OF和SF状态不同	AX < BX

CMP指令例

```
        LEA BX，MAX
        LEA SI，BUF
        MOV CL，20
        MOV AL，[SI]
NEXT：INC  SI
        CMP AL，[SI]
        JNC  GOON          ；CF=0转移
        XCHG  [SI]，AL
GOON：DEC  CL
        JNZ  NEXT
        MOV [BX]，AL
        HLT
```

程序功能

在20个数中找最大的数，并将其存放在MAX单元中。



3. 乘法指令

- 无符号的乘法指令MUL
- 带符号的乘法指令IMUL

- 注意点：
  - 乘法指令采用隐含寻址，隐含的是存放被乘数的累加器AL或AX及存放结果的AX，DX；

无符号数乘法指令

- 格式：
  - **MUL OPRD**

    不能是立即数

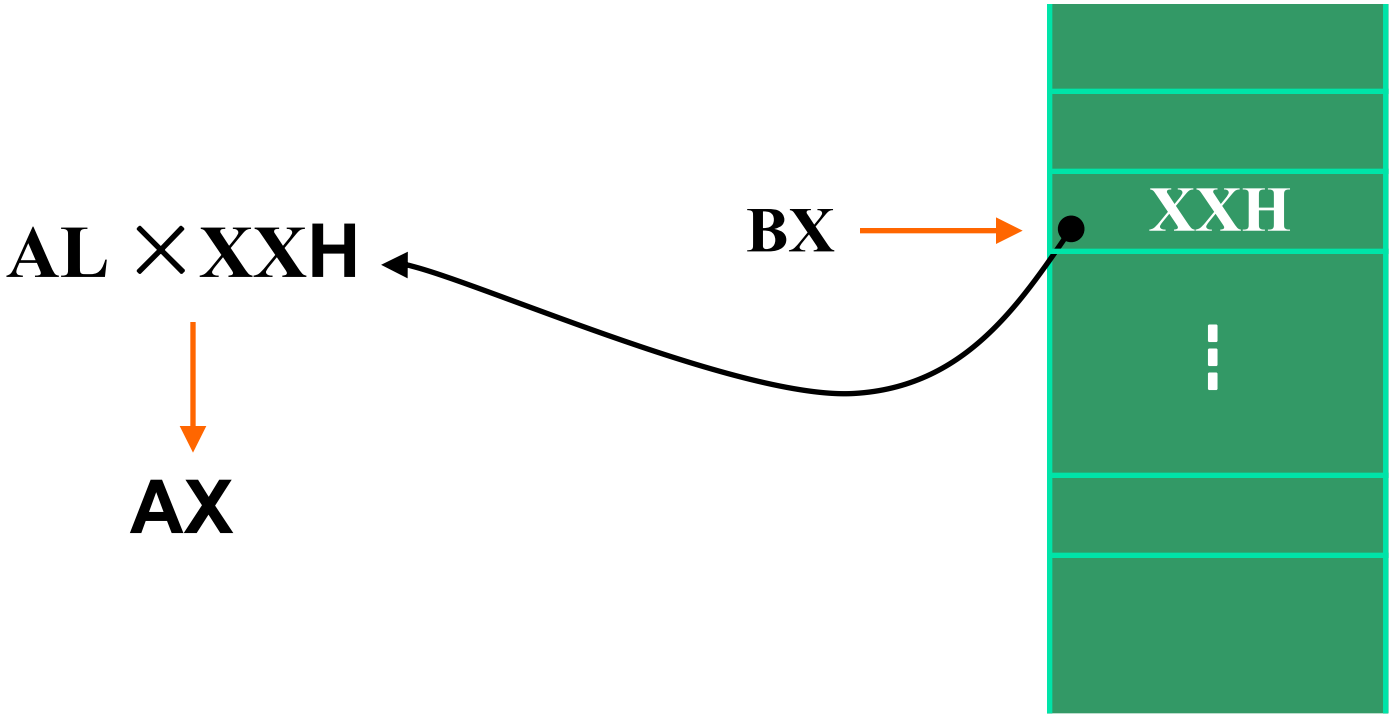
- 操作：
  - OPRD为字节数 → AL×OPRD → AX
  - OPRD为16位数 → AX×OPRD → DX:AX

如：MUL BL；AX ← AL×BL

MUL BX；DX:AX ← AX×BX

无符号数乘法指令例

- MUL BYTE PTR[BX]



有符号数乘法指令

- 格式：
  - IMUL　OPRD
- 指令格式及对操作数的要求与MUL指令相同。
- 指令执行原理：
  ① 将两个操作数取补码（对负数按位取反加1，正数不变）；
  ② 做乘法运算；
  ③ 将乘积按位取反加1。

4. 除法指令

无符号除法指令

- 格式：
  - DIV OPRD

有符号除法指令

- 格式：
  - IDIV OPRD

除法指令的操作

若OPRD是字节数

- 执行：**AX/OPRD**
- 结果：
  - AL=商　　AH=余数

若OPRD是双字节数

- 执行：　**DXAX/OPRD**
- 结果：
  - AX=商　DX=余数

指令要求被除数是除数的双倍字长

5. 其他算术运算指令

- 见教材P122-123

算术运算指令小结

- 指令执行影响状态标志位
- 乘法指令执行结果为相乘数的双倍字长；除法指令要求被除数是除数的双倍字长。
- 例：
  - **MOV SI，1200H**
  - **MOV WORD PTR[SI]，8765H**
  - **MOV AX，[SI]**
  - **MUL WORD PTR[SI]**
  - **AL×[SI] → AX=3543H**

3.3.3 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

1. 逻辑运算指令

- 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影响除AF 外的5个状态标志；
  - 无论执行结果任何，都会使标志位OF=CF=0。

1）“与”指令：

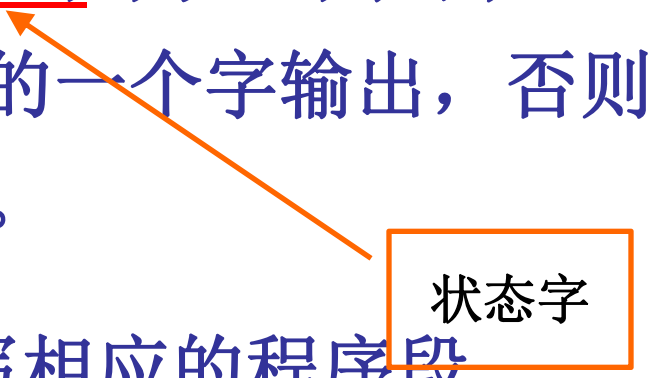
- 格式：
  - AND　OPRD1，OPRD2
- 操作：
  - 两操作数相“与”，结果送目标地址。

“与”指令的应用

- 实现两操作数按位相与的运算
  - AND　BL，[SI]
- 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

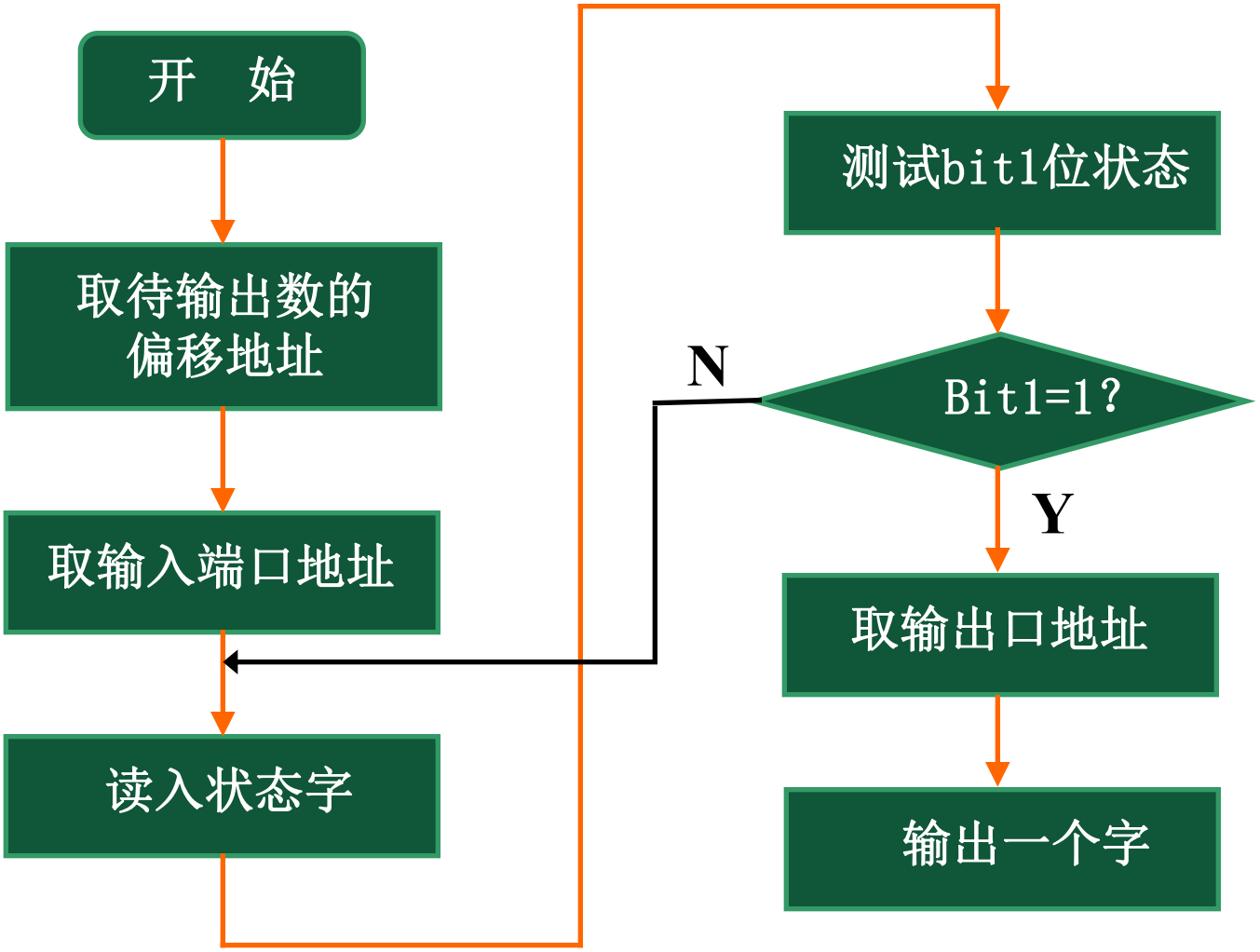
“与”指令应用例

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



编写相应的程序段。

“与”指令应用例


开　始
取待输出数的偏移地址
取输入端口地址
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

“与”指令应用例

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

2）“或”运算指令

- 格式：
  - OR　OPRD1，OPRD2
- 操作：
  - 两操作数相“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，[DI]**
- 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：….
```

“或”指令的应用

将一个二进制数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：
  - 指令的执行对标志位无影响
- 例：
  - **NOT BYTE PTR[BX]**

4）“异或”运算指令

- 格式：
  - XOR　OPRD1，OPRD2
- 操作：
  - 两操作数相“异或”，结果送目标地址
- 例：
  - XOR　BL，80H
  - XOR　AX，AX

5）“测试”指令

- 格式：
  - TEST　OPRD1，OPRD2
- 操作：
  - 执行“与”运算，但运算的结果不送回目标地址。
- 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数，当该数的 bit1， bit3， bit5位同时为1时，可从38FH端口将DATA为首地址的一个字输出，否则就不能进行数　据传送。

编写相应的程序段。

源程序代码：

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

2. 移位指令

- 非循环移位指令
- 循环移位指令

- 注：
  - 移动一位时由指令直接给出；
  - 移动两位及以上，则移位次数由CL指定。

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，1
  SAL OPRD，CL

  } 有符号数

- 逻辑左移指令：

  SHL OPRD，1
  SHL OPRD，CL

  } 无符号数

逻辑右移

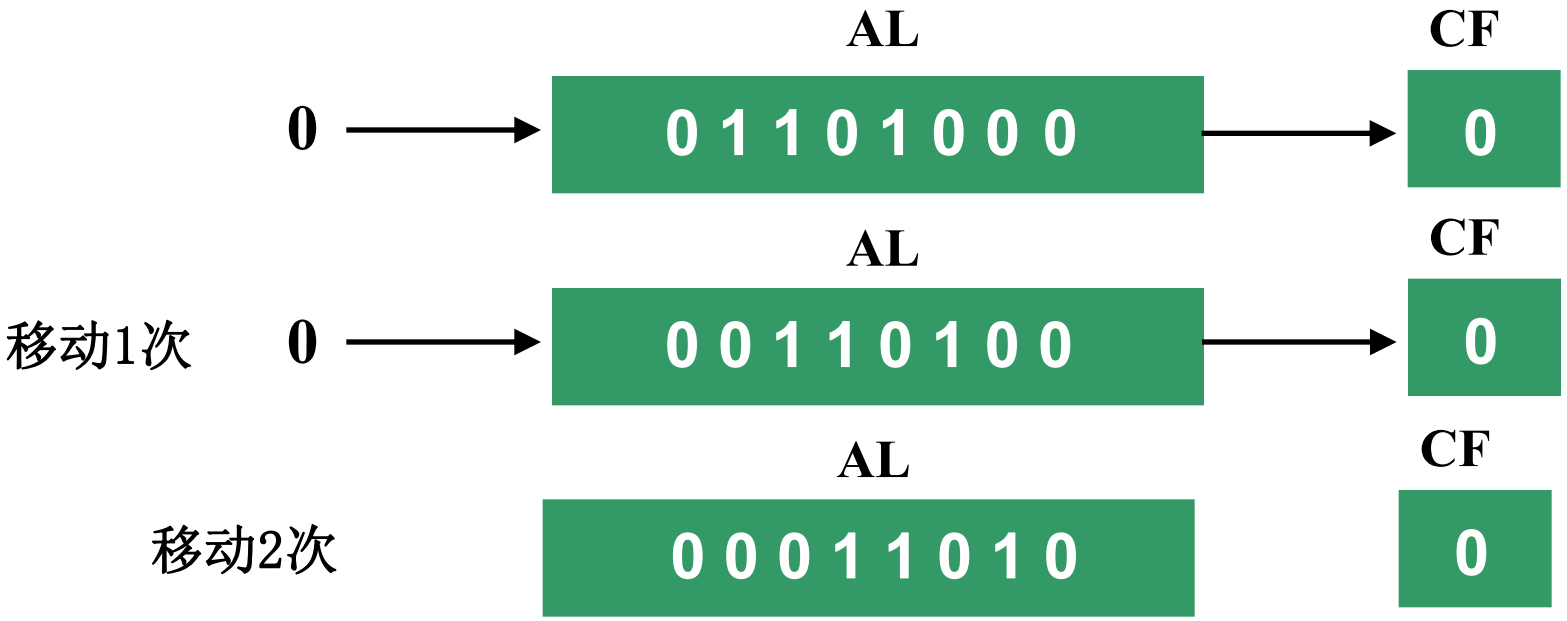
- 格式：

```
SHR   OPRD，1
SHR   OPRD，CL
```

无符号数的右移

逻辑右移例：

- MOV AL，68H
- MOV CL，2
- SHR AL，CL



算术右移

- 格式：

```
SAR   OPRD，1
SAR   OPRD，CL
```

有符号数的右移

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

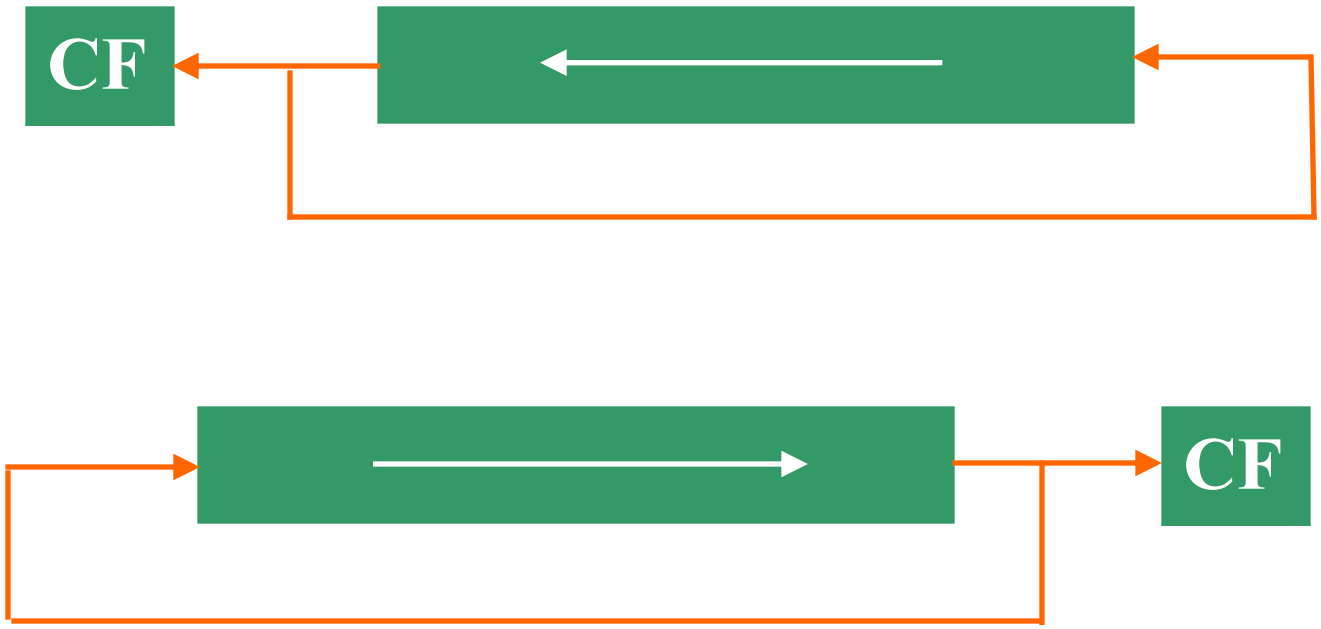
教材p127例

2）循环移位指令

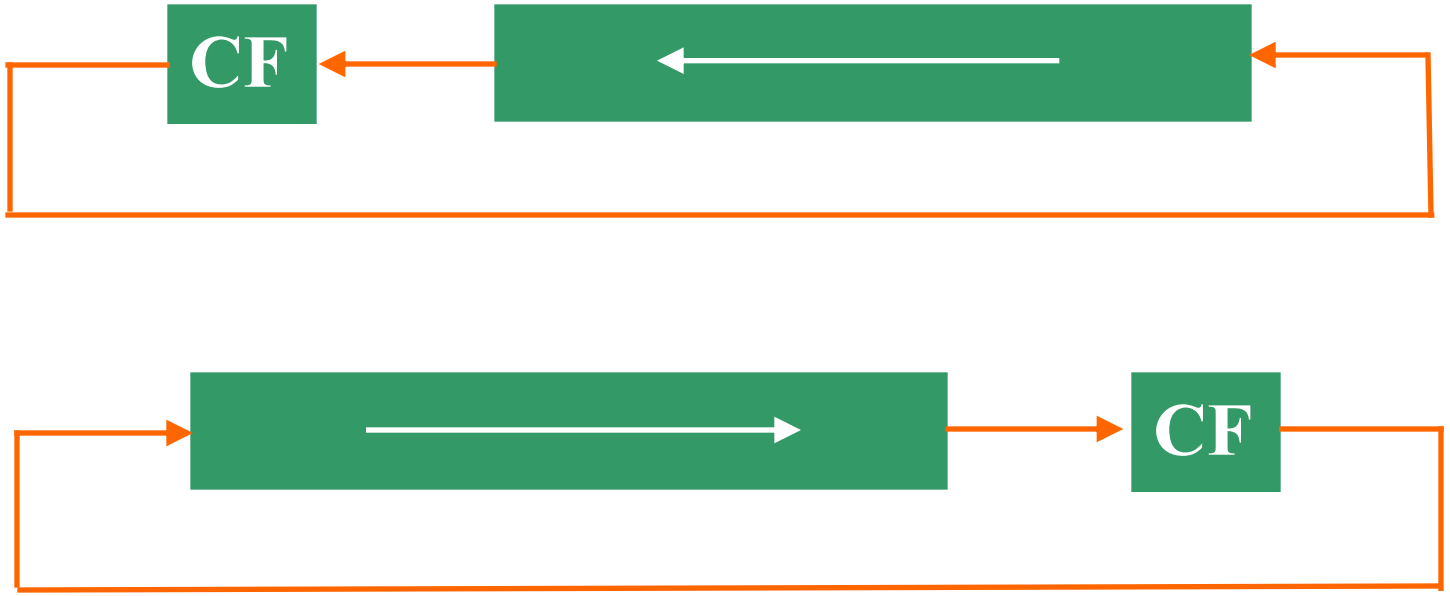
- 不带进位位的循环移位
  - 左移 ROL
  - 右移 ROR
- 带进位位的循环移位
  - 左移 RCL
  - 右移 RCR

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



带进位位的循环移位

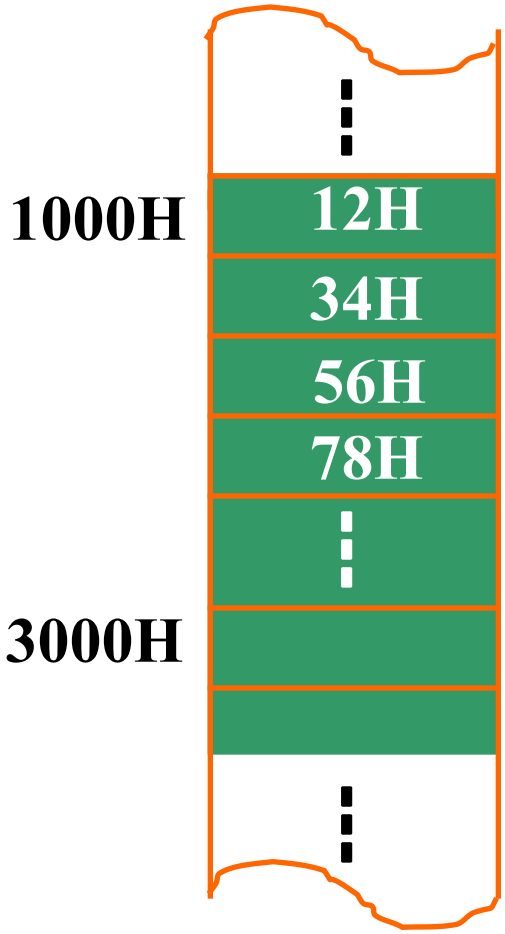


循环移位指令的应用

- 用于对某些位状态的测试；P130例3-34
- 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。P130例3-35

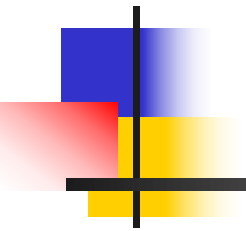
程序功能

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



程序例

```
        MOV SI,1000H
        MOV DI,3000H
        MOV CX,4
Next:   MOV AL,[SI]
        MOV BL,AL
        AND AL,0FH
        OR AL,30H
        MOV [DI],AL
        INC DI
        MOV AL,BL
        PUSH CX
        MOV CL,4
        SHR  AL,CL
        OR   AL,30H
        MOV  [DI],AL
        INC  DI
        INC SI
        POP CX
        DEC CX
        JNZ  Next
        HLT
```



3.3.4 串操作指令

串操作

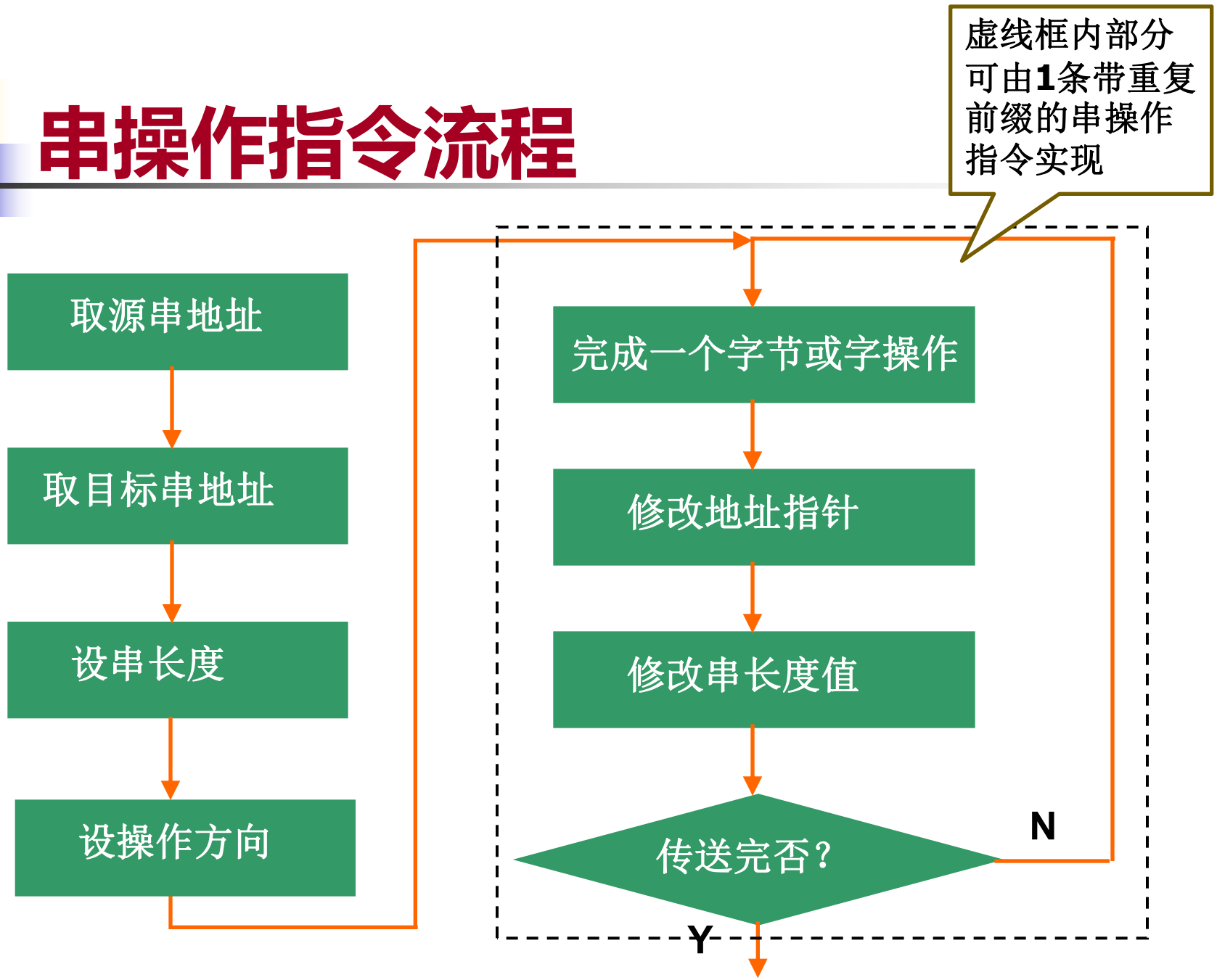
实现两个字符串或数据块的操作

源操作数称为源串，目标操作数称为目标串

串操作指令要求

- 目标操作数必须在附加段，源操作数默认在数据段（允许段重设）。
- 源串和目标串指针分别为SI和DI
- 串长度值必须由CX给出
- 需要设置数据的操作方向
  - 确定DF的状态
- 串操作指令前可以加重复前缀。使用时应注意：
  - 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

串操作指令流程


虚线框内部分
可由1条带重复
前缀的串操作
指令实现
取源串地址
取目标串地址
设串长度
设操作方向
完成一个字节或字操作
修改地址指针
修改串长度值
传送完否？
N
Y

1. 串传送指令

- 格式：

```
MOVS     OPRD1，OPRD2

MOVSB

MOVSW
```

- 串传送指令常与无条件重复前缀连用

串传送指令

- 对比用MOV指令和MOVS指令实现将200个字节数据从内存的一个区域送到另一个区域的程序段。

P110例3-12

串传送指令例

- 用串传送指令实现200个字节数据的传送：

```
LEA   SI，MEM1
LEA   DI，MEM2
MOV   CX，200
CLD
REP   MOVSB
HLT
```

2. 串比较指令

- 格式：

  CMPS　　OPRD1，OPRD2

  CMPSB

  CMPSW

- 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。
- 前缀的操作对标志位不影响

串比较指令例

测试200个字节数据是否传送正确：

```
LEA  SI，MEM1
LEA  DI，MEM2
MOV  CX，200
CLD
REPE   CMPSB
TEST  CX，00FFH
        JZ  STOP
        DEC  SI
        MOV  AL，[SI]
        MOV  BX，SI
STOP：HLT
```

3. 串扫描指令

- 格式：

  SCAS　　OPRD

  SCASB

  SCASW

目　标
操作数

- 执行与CMPS指令相似的操作，只是这里的源操作数是AX或AL

串扫描指令的应用

- 常用于在指定存储区域中寻找某个关键字。

教材p133例

4. 串装入指令

- 格式：

源操作数

LODS　　OPRD

LODSB

LODSW

- 操作：
  - 对字节：AL ← [DS:SI]
  - 对　字：AX ← [DS:SI]

4. 串装入指令

- 用于将内存某个区域的数据串依次装入累加器，以便显示或输出到接口。
- LODS指令一般不加重复前缀。

5. 串存储指令

- 格式：

  STOS　　OPRD

  STOSB

  STOSW

目　标
操作数

- 操作：
  - 对字节：　AL → [ES:DI]
  - 对　字：　AX → [ES:DI]

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）；
  - 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

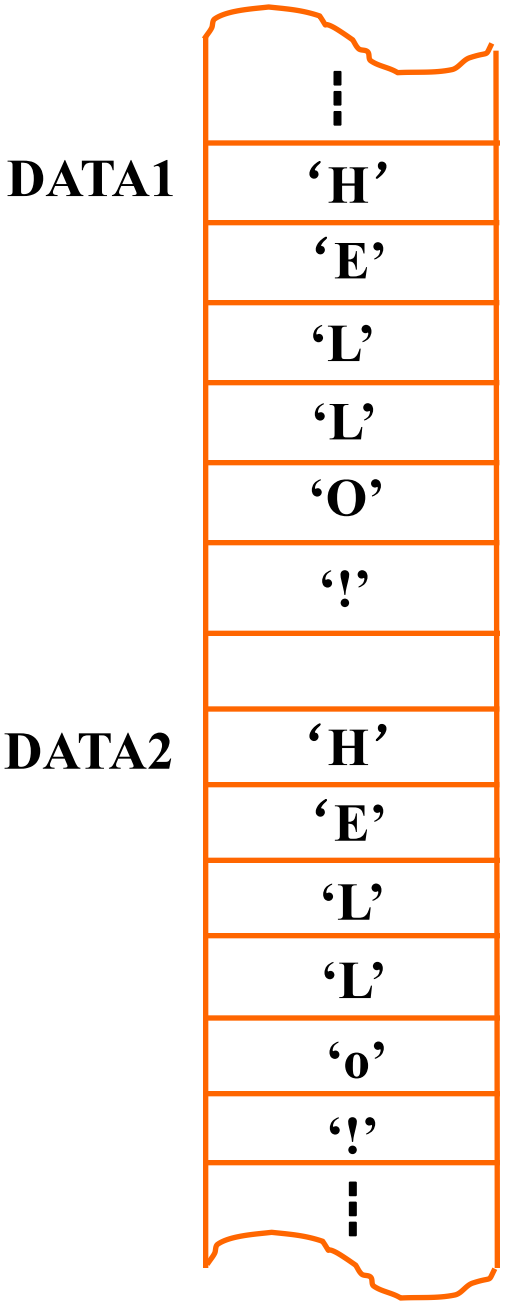
- 将内存某单元清零
  - P135例3-40

串操作指令应用注意事项

- 需要定义附加段
  - 目标操作数必须在附加段
- 需要设置数据的操作方向
  - 确定DF的状态
- 源串和目标串指针分别为SI和DI
- 串长度值必须由CX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

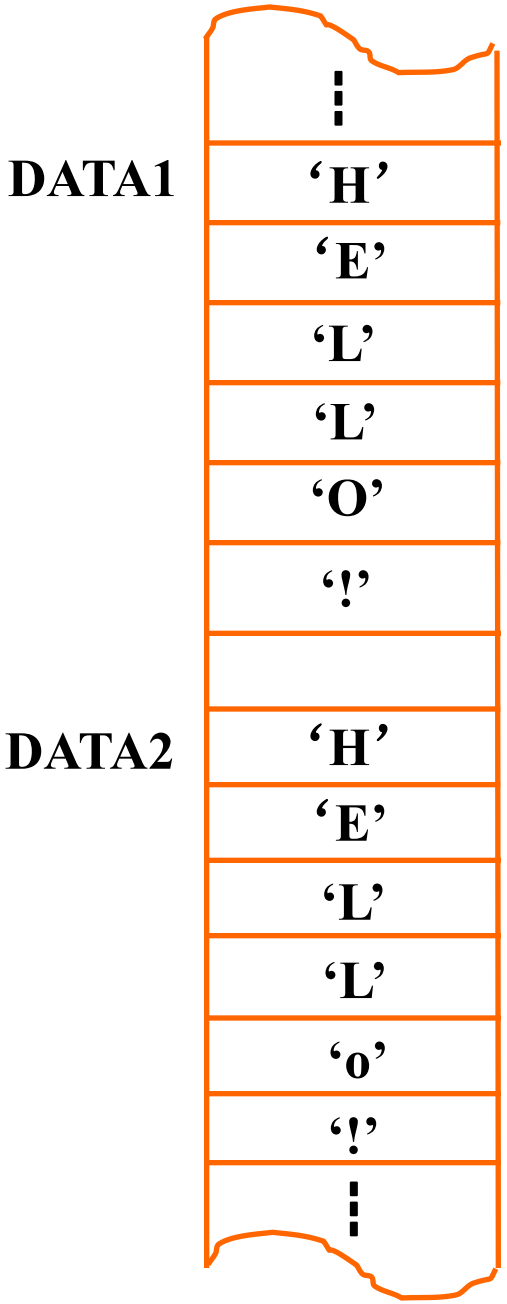
随堂练习（1）

- 字符串在内存中的存放如图所示。
  - LEA　SI，DATA1
  - LEA　DI，DATA2
  - MOV　CX，6
  - CLD
  - REPZ　CMPSB
- 执行完上述程序段后：
  - SI=（　　）
  - DI=（　　）
  - CX=（　　）
  - ZF=（　　）



随堂练习（1）

- 字符串在内存中的存放如图所示。
  - LEA　SI，DATA1
  - LEA　DI，DATA2
  - MOV　CX，6
  - CLD
  - REPZ　CMPSB
- 执行完上述程序段后：
  - SI=（ DATA1+5　）
  - DI=（ DATA1+5　）
  - CX=（　1　）
  - ZF=（　0　）



随堂练习（2）

- 判断以下程序段完成的功能

```
    MOV AX，2100H
    MOV DS，AX
    MOV ES，AX
    LEA  SI，M1
    LEA  DI，M2
    MOV  AL，’0’
    MOV  CX，100
L1：MOV  [SI]，AL
    INC  SI
    DEC  CX
    JNZ  L1
    LEA  SI，M1
    CLD
    MOV  CX，100
    REP MOVSB
    HLT
```

上述程序执行完后，SI=?

3.3.5 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是：
  - 控制程序的执行方向
- 决定程序执行方向的因素：
  - CS，IP
- 控制程序执行方向的方法：
  - 仅修改IP，则程序将改变当前的执行顺序，转向本代码段内其它某处执行；
  - 修改CS 和IP ，则程序转向另一个代码段执行。

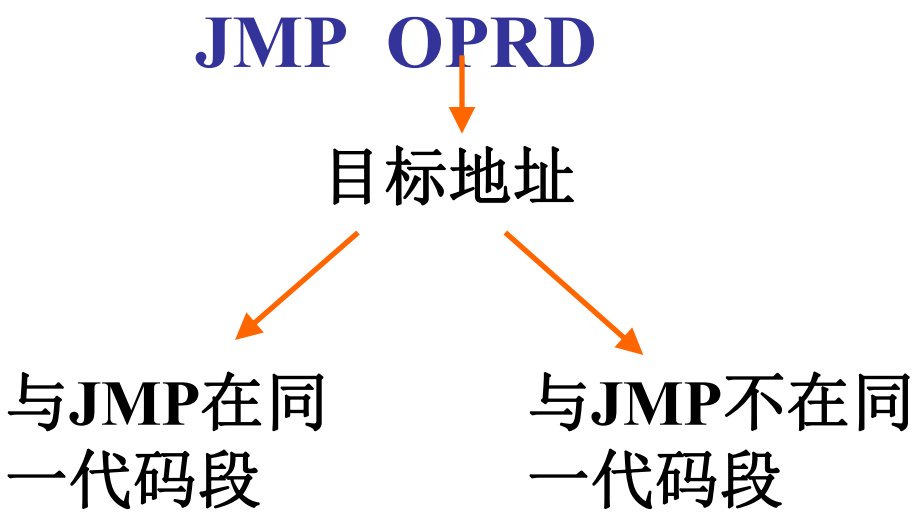
一、转移指令

通过修改指令的偏移地址或段地址及偏移地址实现程序的转移

- 无条件转移指令
  - 无条件转移到目标地址，执行新的指令
- 有条件转移指令
  - 在具备一定条件的情况下转移到目标地址

1. 无条件转移指令

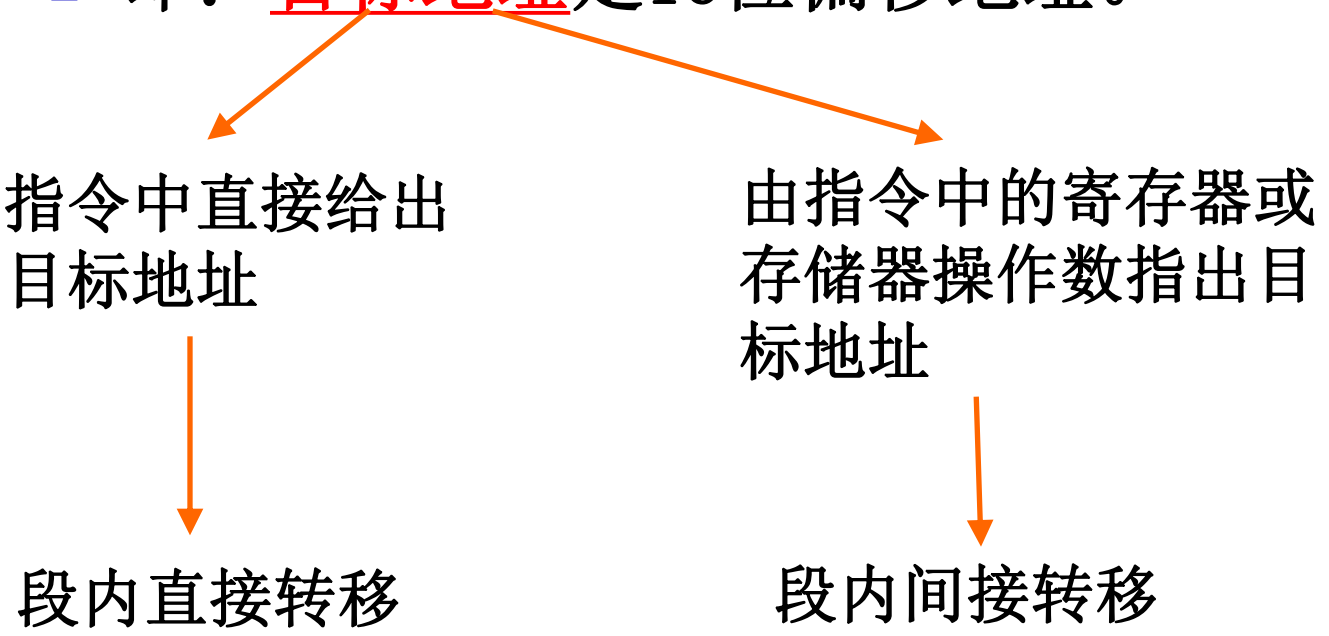
- 格式：



原则上可实现在整个内存空间的转移

无条件段内转移

- 转移的目标地址在当前代码段内，段地址不改变。
- 即：目标地址是16位偏移地址。

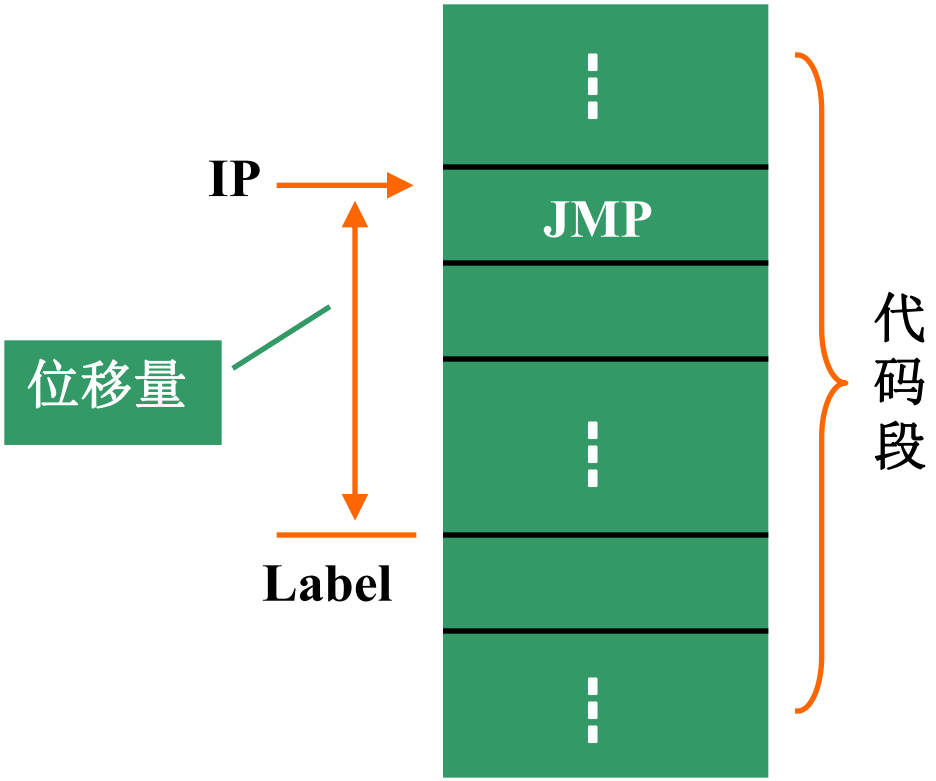


段内直接转移

- 转移的目标地址由指令直接给出
- 格式：
  - JMP　Label

段内直接转移示图

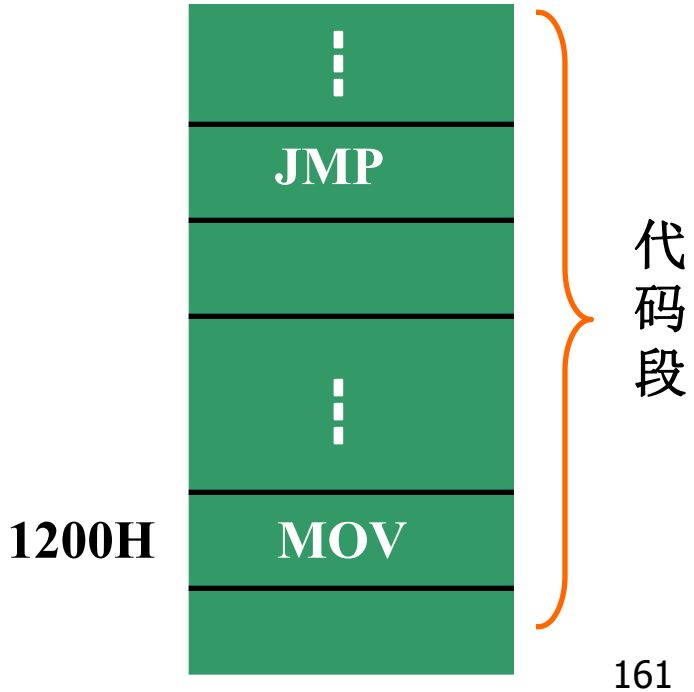
JMP Label



下一条要执行指令的偏移地址=当前IP+位移量

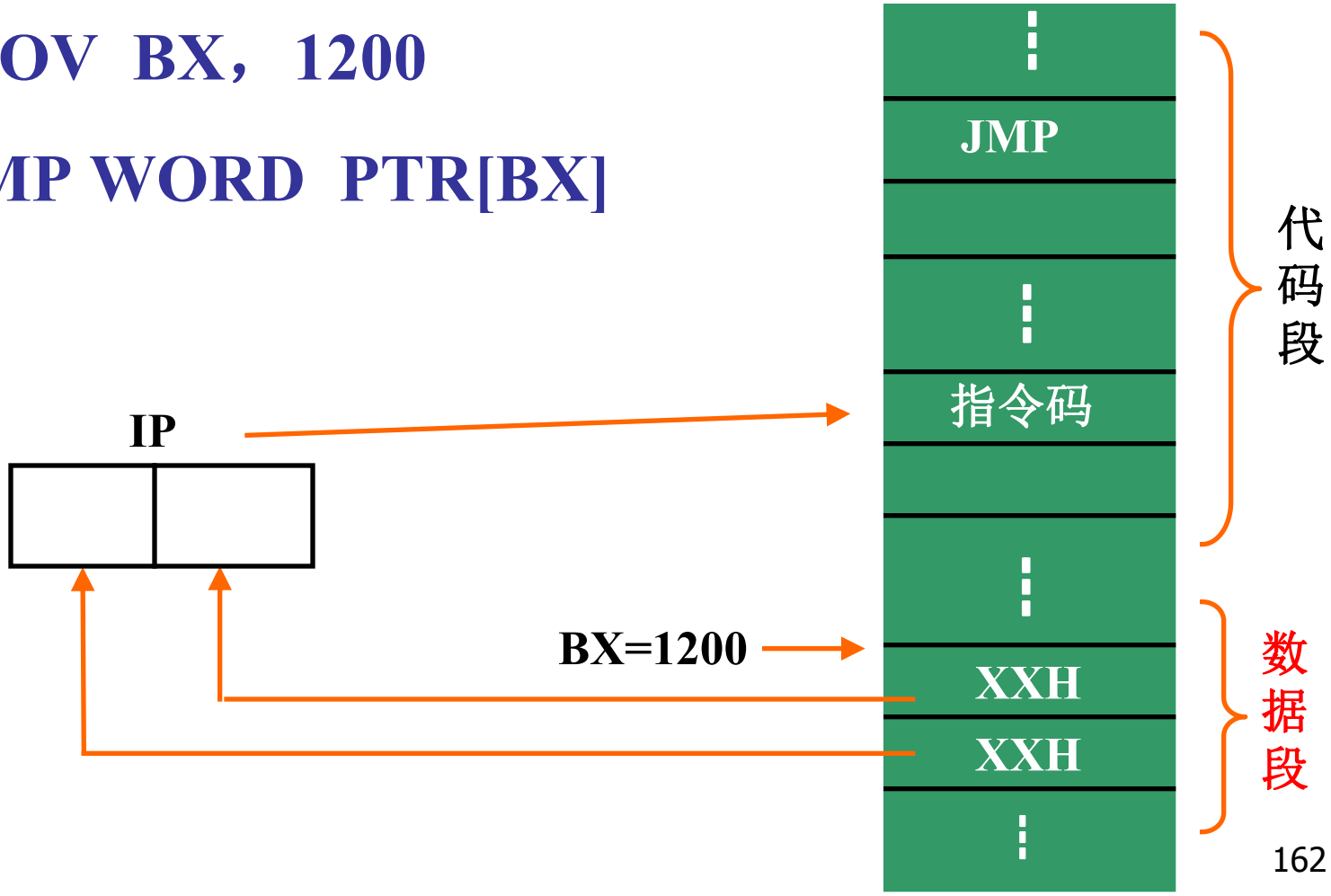
段内间接转移

- 段内间接转移
  - 转移的目标地址存放在某个16位寄存器或存储器的某两个单元中
- 例：
  - **MOV BX，1200H**
  - **JMP BX**
- 执行完上述指令后：
  - IP=1200H



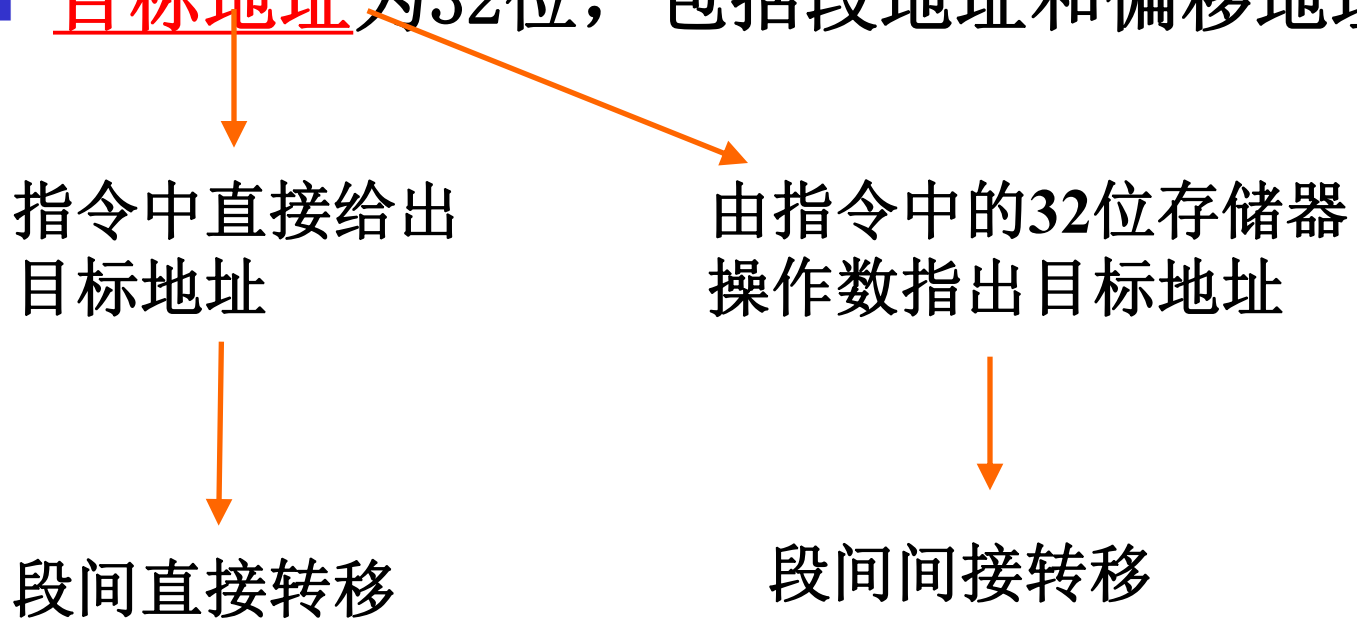
段内间接转移例

- MOV BX，1200
- JMP WORD PTR[BX]



无条件段间转移

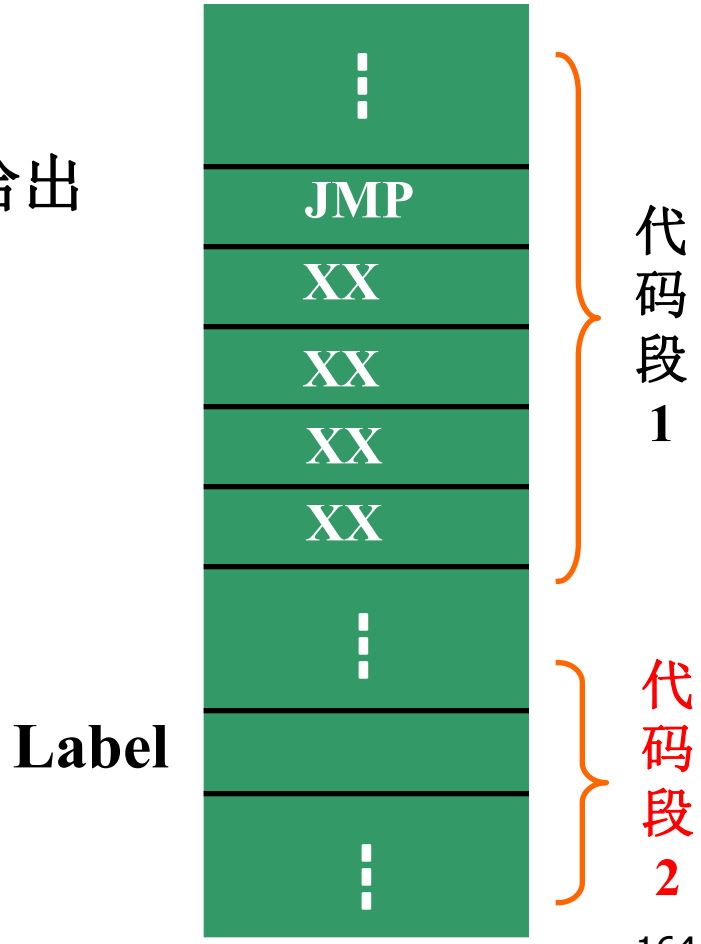
- 转移的目标地址不在当前代码段内。
- 目标地址为32位，包括段地址和偏移地址。



段间直接转移

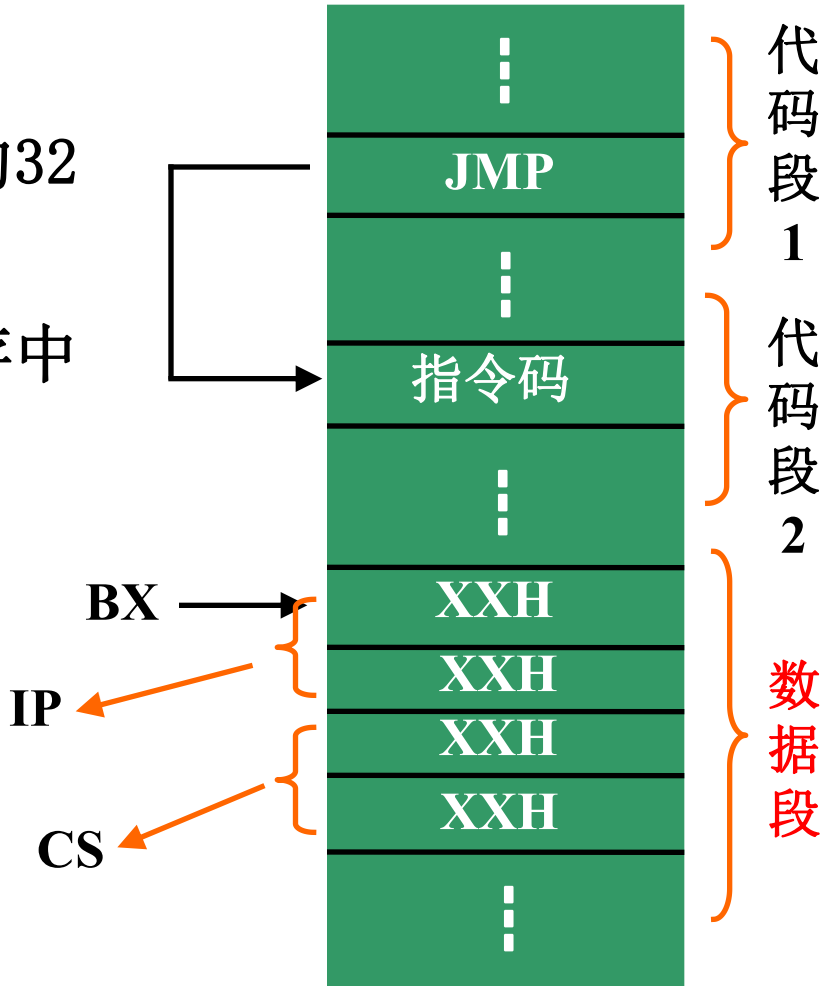
- 段间直接转移
  - 转移的目标地址由指令直接给出
- 格式：
  - **JMP FAR Label**

远地址标号



段间间接转移

- 段间间接寻址
  - 转移的目标地址由指令中的32位操作数给出
  - 32位目标地址须存放于内存中
- 例：
  - JMP DWORD PTR[BX]



无条件转移指令例

CS： IP

(1) 2000:0100　　MOV AX,1200H

(2) 2000:0103　　JMP NEXT

---

(3) 2000:0120　NEXT: MOV BX,1200H

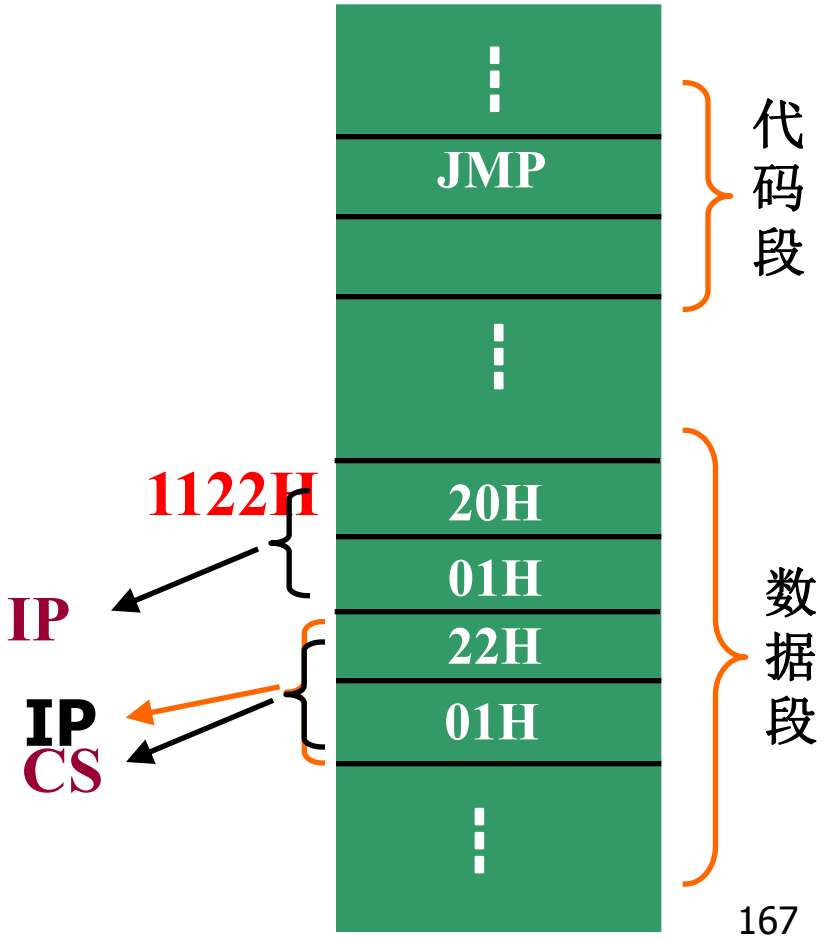
(4)　　JMP BX

---

(5) 2000:1200

无条件转移指令例

- MOV SI，1122H
- MOV WORD PTR[SI]，0120H
- ADD SI，2
- MOV WORD PTR[SI]，0122H
- JMP WORD PTR[SI-2]



单选题 1分

设置

执行以下操作后，哪个结果是错误的：
AND AX , AX

A AX不变

B AX=0

C CF=0

D OF=0

提交

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

- 条件转移指令均为段内短转移，即转移范围为：

  -128-------+127

p138

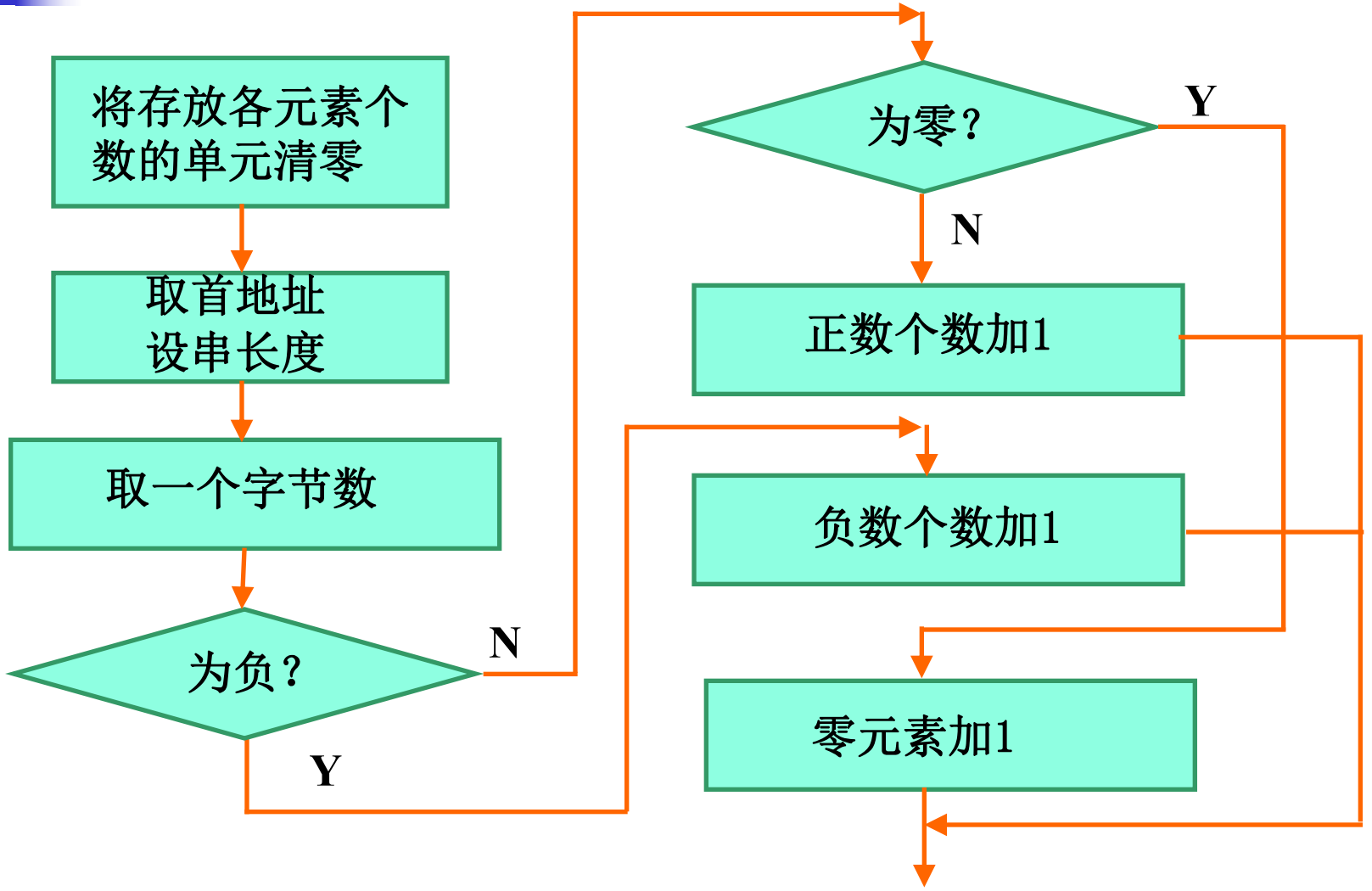
条件转移指令的应用

- 几种条件转移指令的应用
  - 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/JPE
    - 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
  - JA/JAE/JB/JBE （A，Above；E，Equal；B，Below；）
    - 判断CF或CF+ZF的状态。常用于无符号数的大小比较

转移指令例

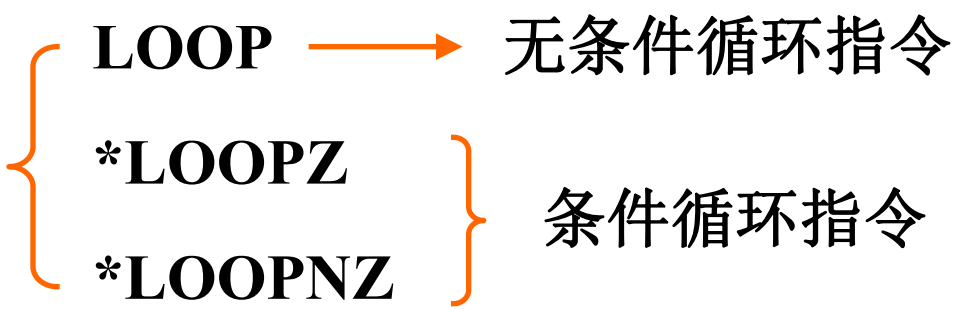
- 统计内存数据段中以TABLE为首地址的100个8位带符号数中正数、负数和零元数的个数。

转移指令例（流程图）


将存放各元素个数的单元清零
取首地址
设串长度
取一个字节数
为负？
N
Y
为零？
Y
N
正数个数加1
负数个数加1
零元素加1

3. 循环控制指令

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



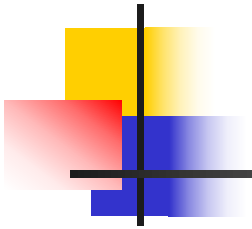
(1)无条件循环指令

- 格式：
  - LOOP　LABEL
- 循环条件：
  - CX ≠ 0
- 操作：先将CX内容减1，再判断其值是否为0

完全相当于
DEC　CX
JNZ　符号地址

(2) 条件循环指令

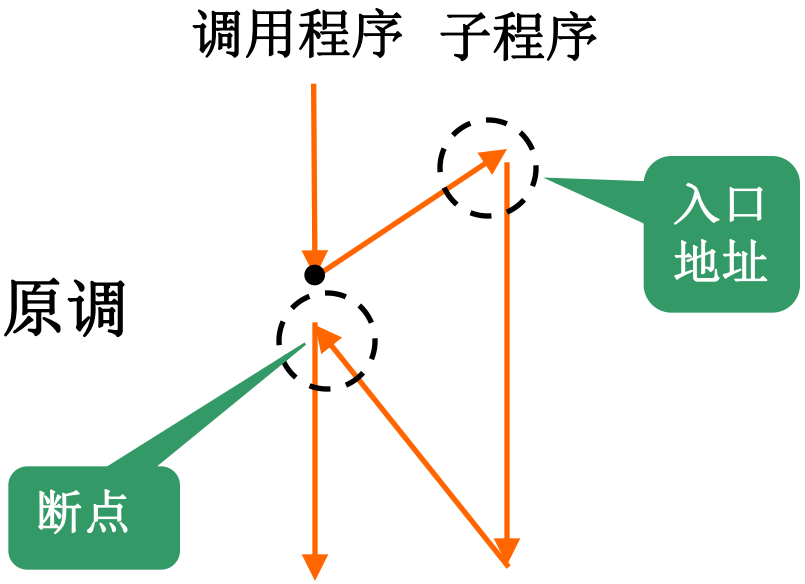
- 格式：
  - LOOPZ/LOOPE　LABEL
- 循环条件：
  - CX ≠ 0，且ZF=1（前面指令执行后内容为0）
- 操作：



- 格式：
  - LOOPNZ/LOOPNE　LABEL
- 循环条件：
  - CX ≠ 0，且ZF=0（前面指令执行后内容不为0）
- 操作：

4. 过程调用和返回

- 过程的定义，见教材P141
- 过程调用指令
  - 用于调用一个子过程
- 与转移指令的比较
  - 子过程执行结束后要返回原调用处
    - 必须保护返回地址



调用指令的执行过程

① 保护断点

- 将调用指令的下一条指令的地址（断点）压入堆栈

② 获取子过程的入口地址

- 子过程第1条指令的偏移地址

③ 执行子过程，含相应参数的保存及恢复

④ 恢复断点，返回原程序。

- 将断点偏移地址由堆栈弹出

过程调用

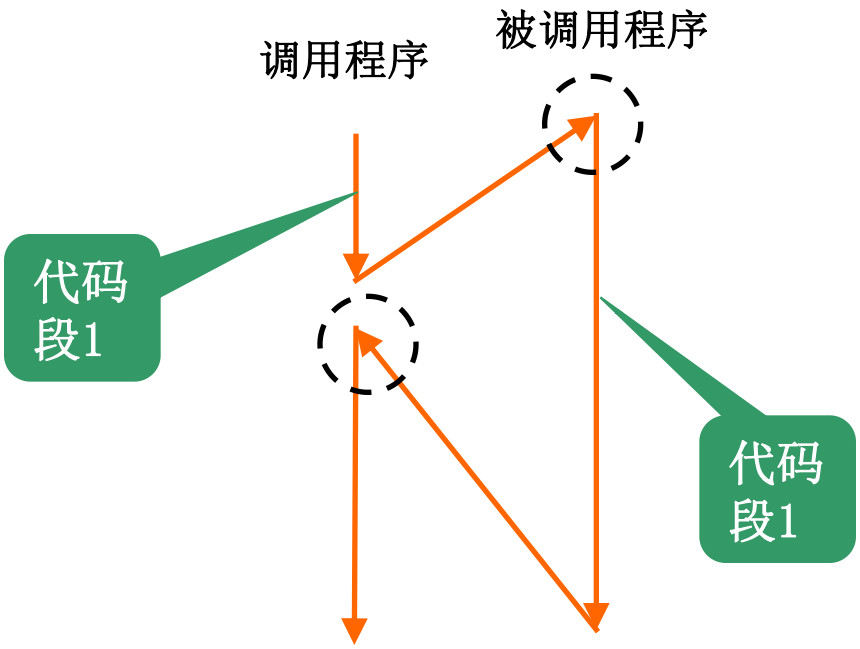
- 段内调用
  - 段内直接调用
  - 段内间接调用
- 段间调用
  - 段间直接调用
  - 段间间接调用

(1). 段内调用

- 被调用程序与调用程序在同一代码段
  - 调用前只需保护断点的偏移地址
- 格式：
  - CALL NEAR PROC

近过程名

- 执行过程：
  - 将断点的偏移地址压入堆栈
  - 根据过程名找子程序入口



段内调用例

（1）CALL TIMRE ⟶ 直接调用

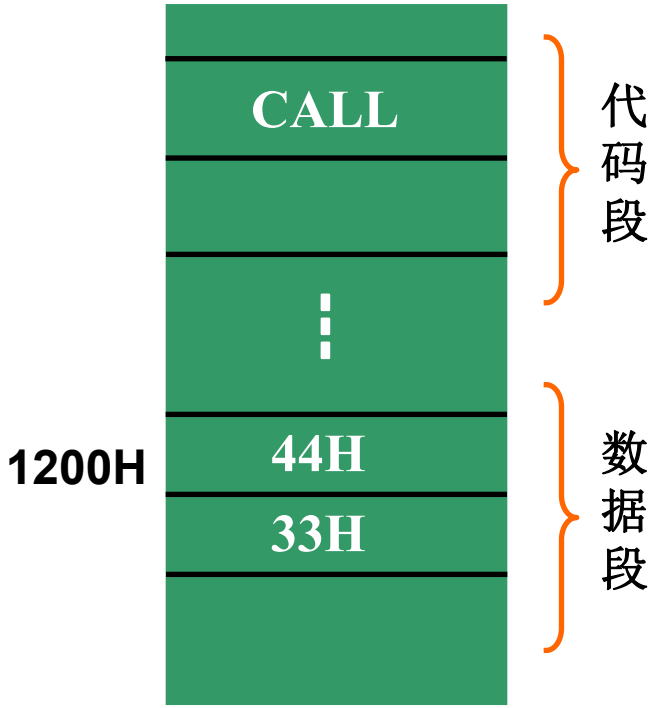
（2）CALL WORD PTR[SI] ⟶ 间接调用

设：SI=1200H
CS=6000H

执行第（2）条指令后：

CS = 6000H

IP = 3344H



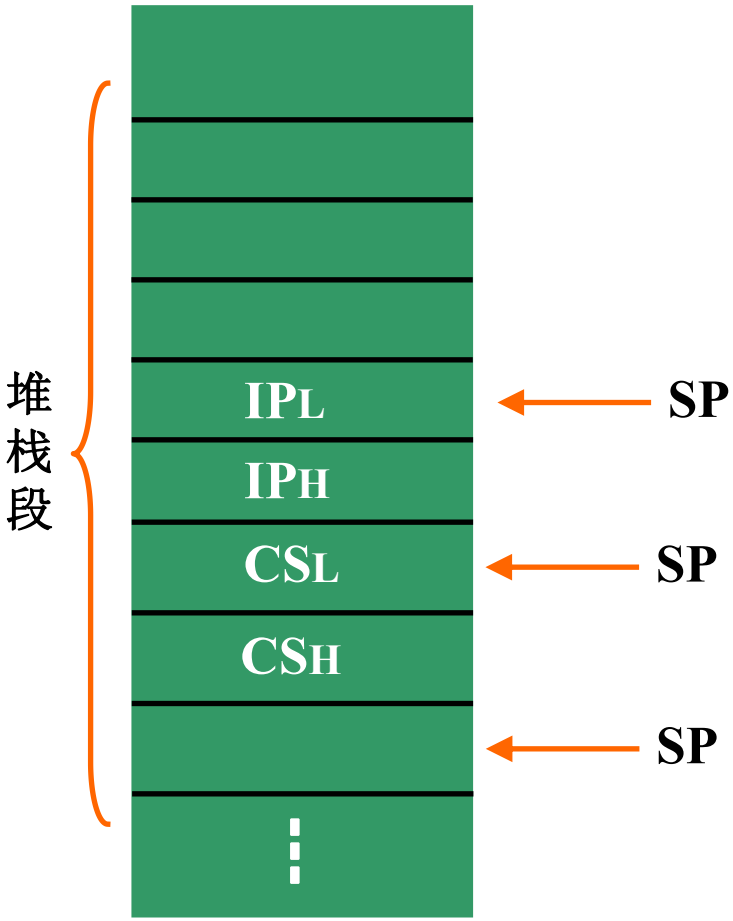
(2). 段间调用

- 子过程与原调用程序不在同一代码段

调用前需保护断点的段基地址和偏移地址

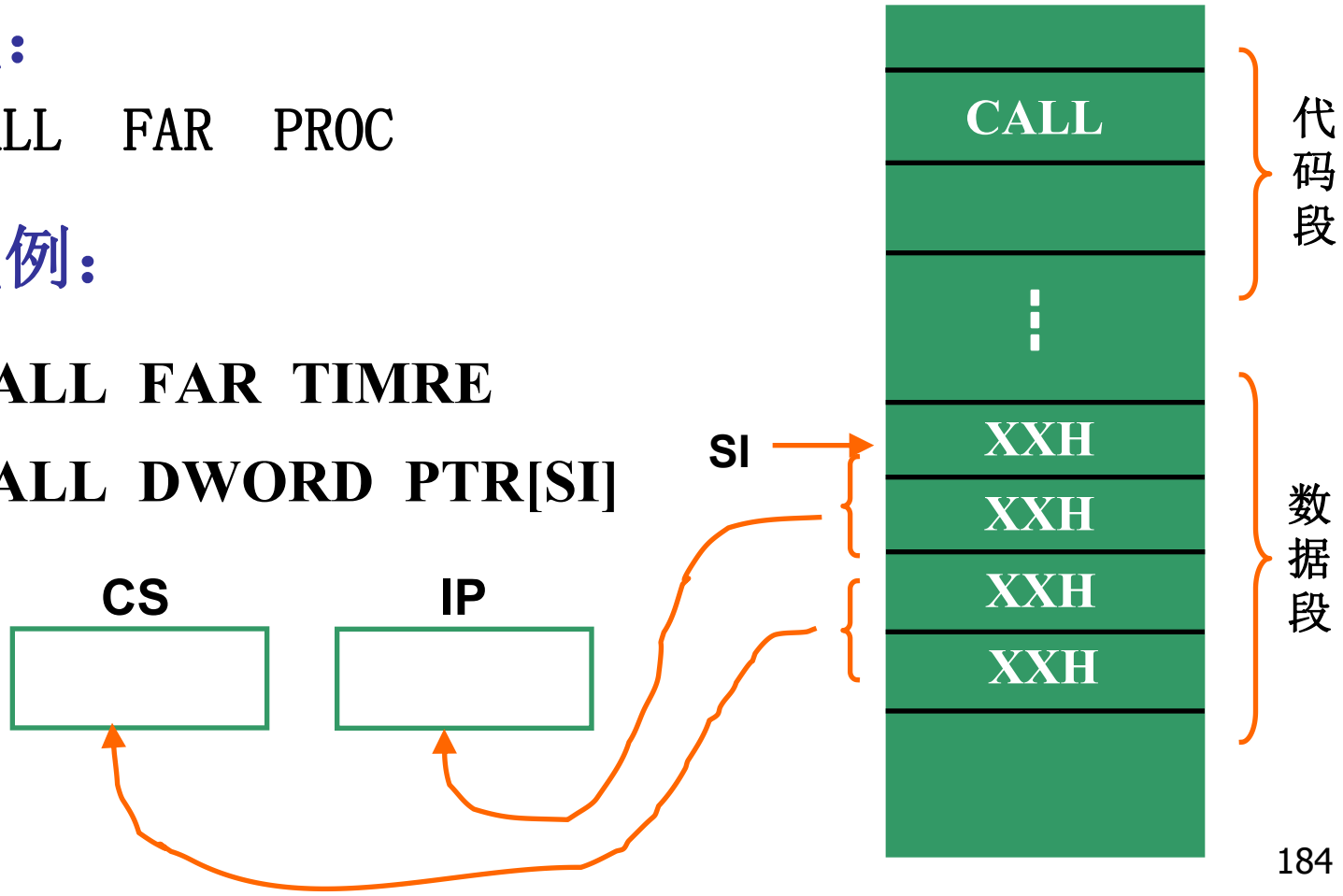
- 先将断点的CS压栈，再压入IP。

段间调用对堆栈区的影响



段间调用例

- 格式：
  - CALL FAR PROC
- 格式例：
  - CALL FAR TIMRE
  - CALL DWORD PTR[SI]



(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

子程序的最后一条指令必须是RET

5. 中断指令

- 中断　　相关定义见教材P144
- 中断源
- 中断的类型
- 中断指令
  - 引起CPU产生一次中断的指令

中断与过程调用：

- 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内容；
- 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，入口地址则在向量码指向的内存单元中。

(1). 中断指令

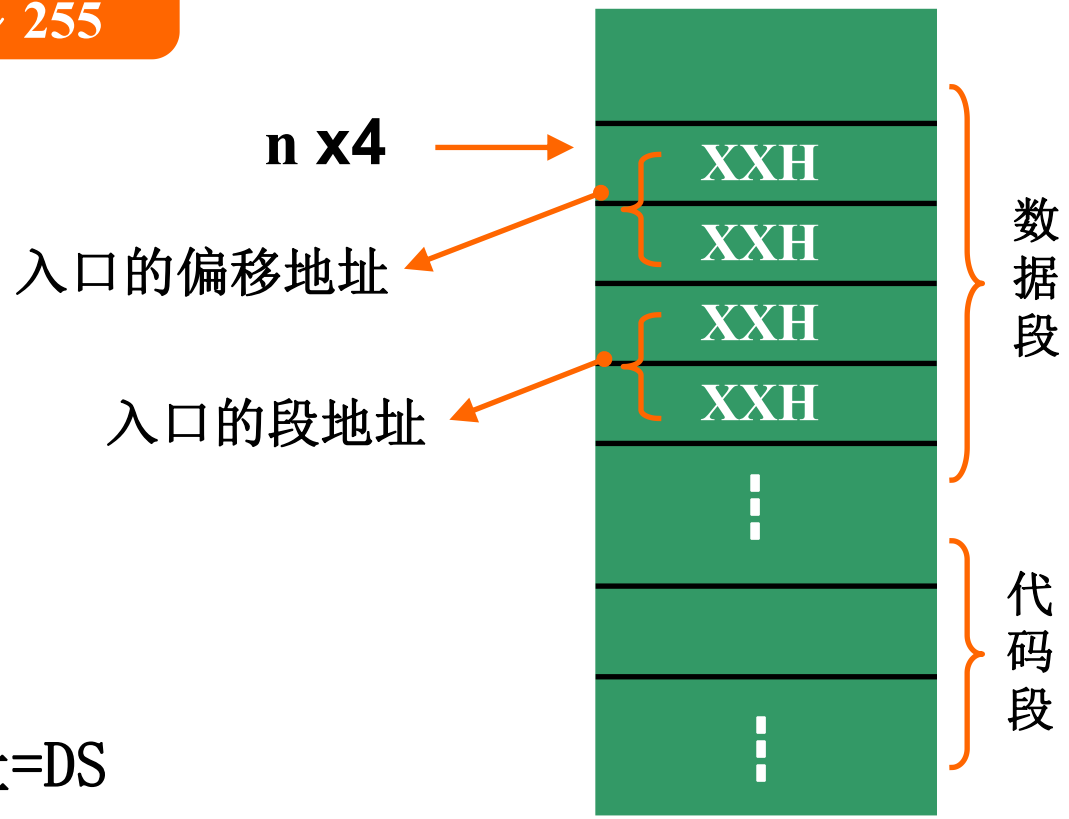
- 格式：
  - INT n

中断类型码
n=0 ～ 255

- 说明：
  - n x 4

存放中断服务子程序入口地址的单元的偏移地址

该单元在数据段，段地址=DS



中断指令的执行过程

① 将FLAGS压入堆栈；

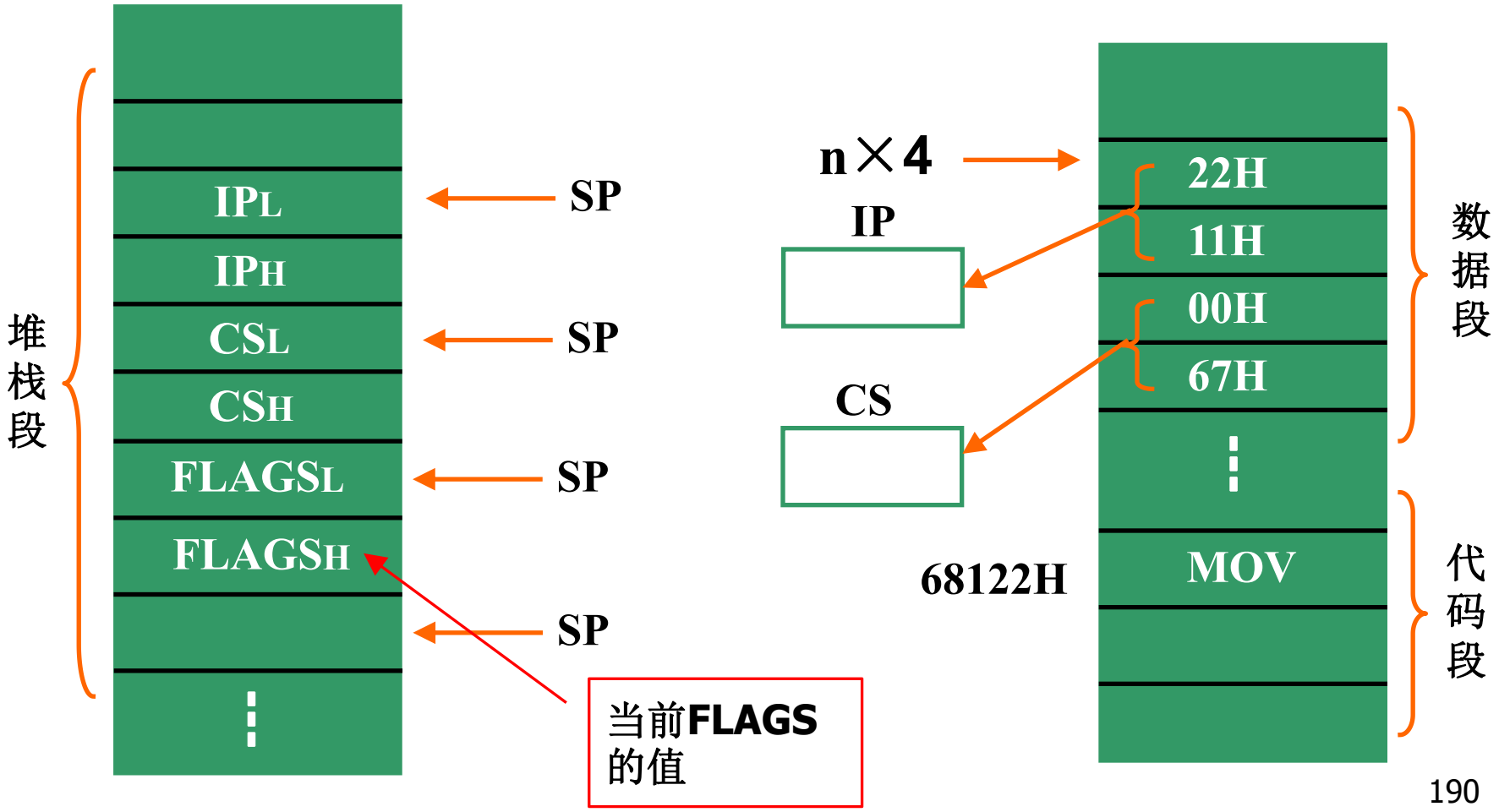
② 将INT指令的下一条指令的CS、IP压栈；

③ 由n×4得到存放中断向量的地址；

④ 将中断向量（中断服务程序入口地址）送CS和IP寄存器；

⑤ 转入中断服务程序。

中断指令的执行过程


堆栈段
IPL
IPH
CSL
CSH
FLAGSL
FLAGSH
SP
SP
SP
SP
当前FLAGS的值
n×4
IP
CS
68122H
22H
11H
00H
67H
MOV
数据段
代码段

中断指令例

执行程序段：

CS　　IP

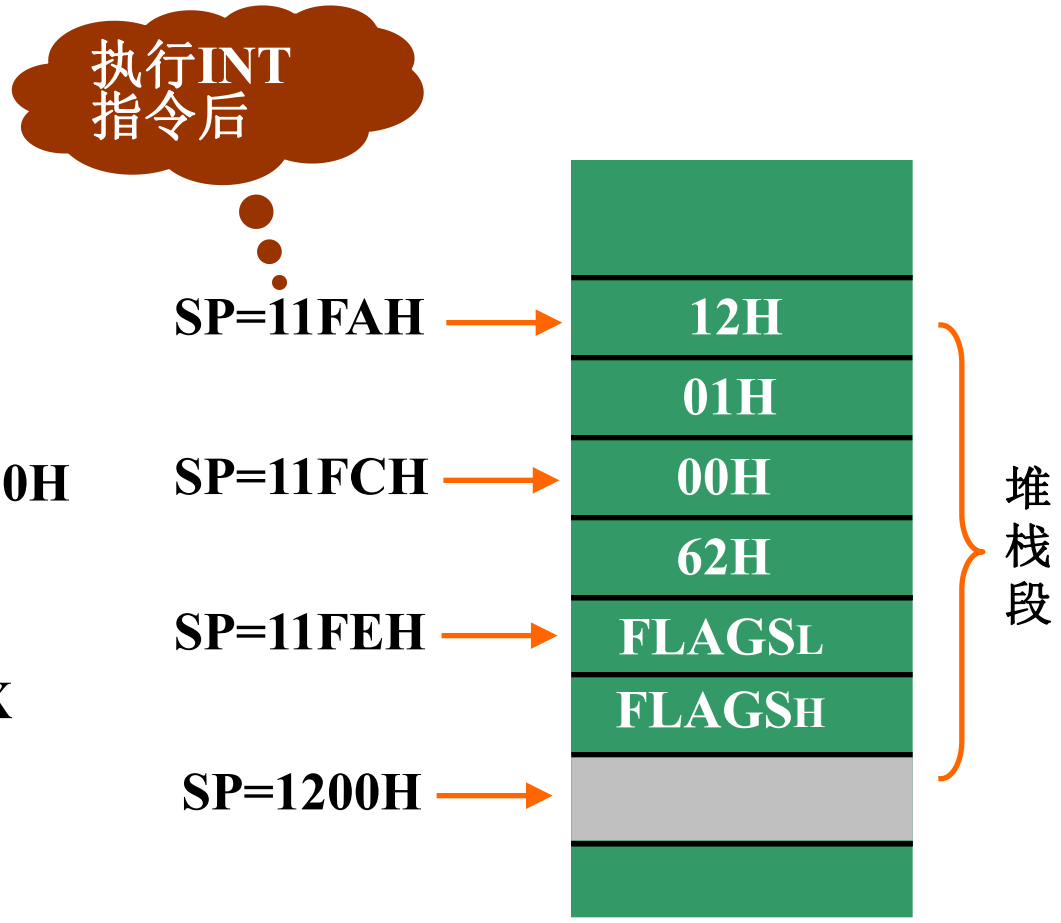
⋮

6200H:010DH　MOV　SP，1200H

6200H:0110H　INT　21H

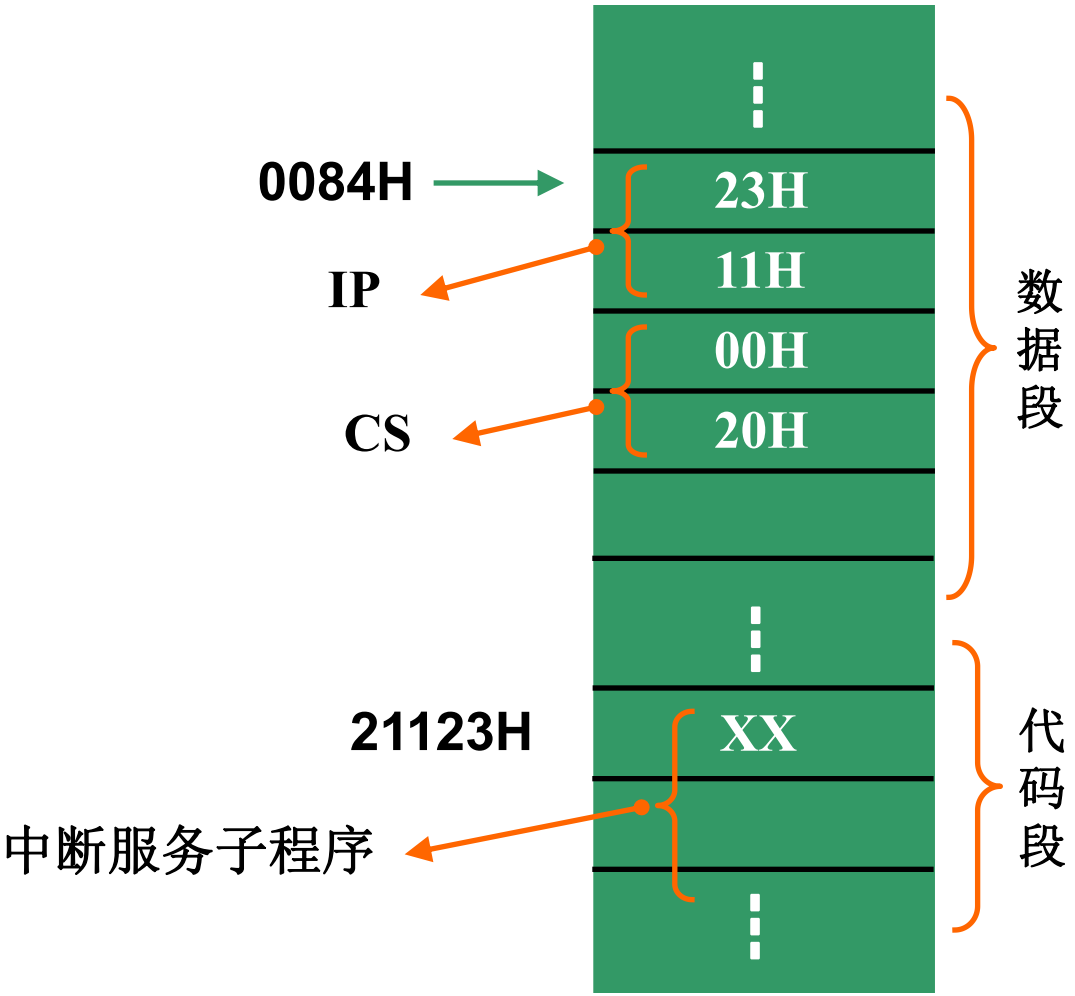
6200H:0112H　MOV　AX，BX

⋮



中断指令例

- 执行INT 21H指令后
  - IP=[21H×4]
  - CS=[（21H×4）+2]

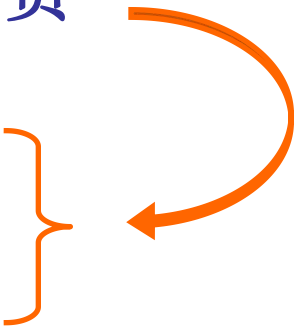


(2). 中断返回指令

- 格式：
  - IRET
- 中断服务程序的最后一条指令，负责

恢复断点

恢复标志寄存器内容



3.3.6 处理器控制指令

见教材P145

说明见
p145表

对标志位的操作

与外部设备的同步