第3章

指令系统

主要内容：

- 指令系统的一般概念
- 对操作数的寻址方式
- 各类指令的功能与使用介绍：

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

- 指令及指令系统；
- 指令的格式；
- 指令中的操作数类型；
- 指令的执行时间

一、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。

二、指令格式

- **指令中应包含两部分内容：**
  - **指令执行的功能--操作码**
  - **指令执行的对象--操作数**

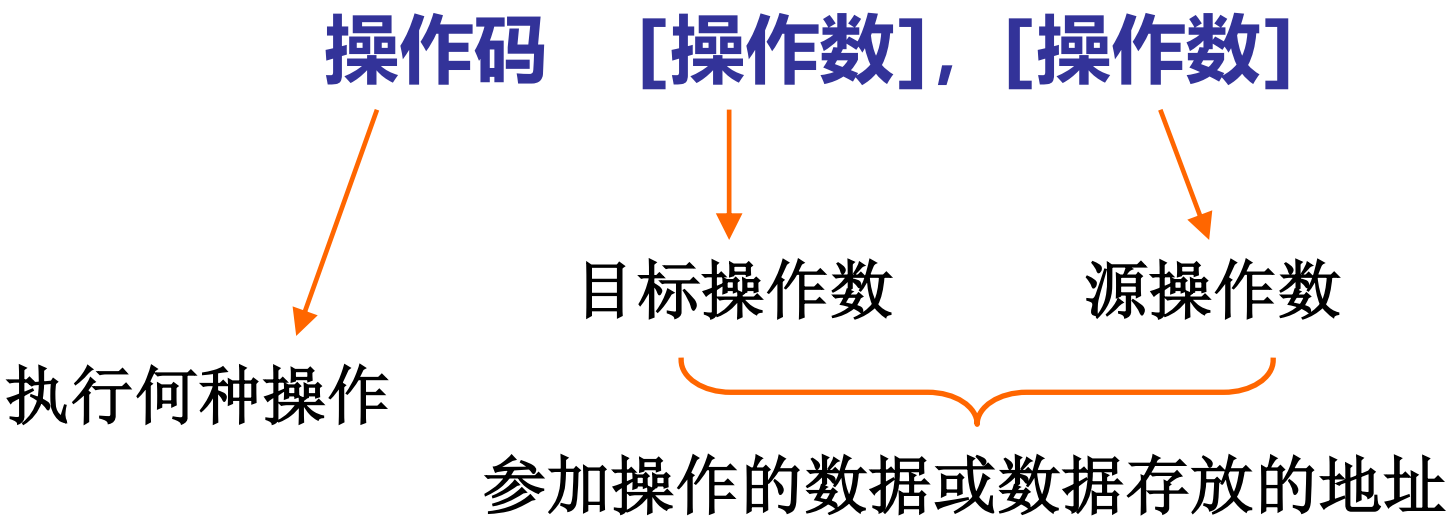
具体应包括以下三个内容：

执行的操作

运算数据的来源

运算结果的去向

指令的一般格式



指令格式：

- **根据操作数的数量，指令可以有以下格式：**

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

三、指令中的操作数的类型

- **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、32位，只能作为源操作数。

  例： MOV AX，1234H

  MOV BL，22H

  MOV EDX, 12345678H

- 立即数是运算数据本身，无地址含义，故无法作为目标操作数

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32位、16位或8位。
- 例：
  - MOV AX，BX
  - MOV DL，CH
  - MOV ECX, EAX

存储器操作数

- 参加运算的数存放在存储器的某个单元中
- 表现形式：[　　]

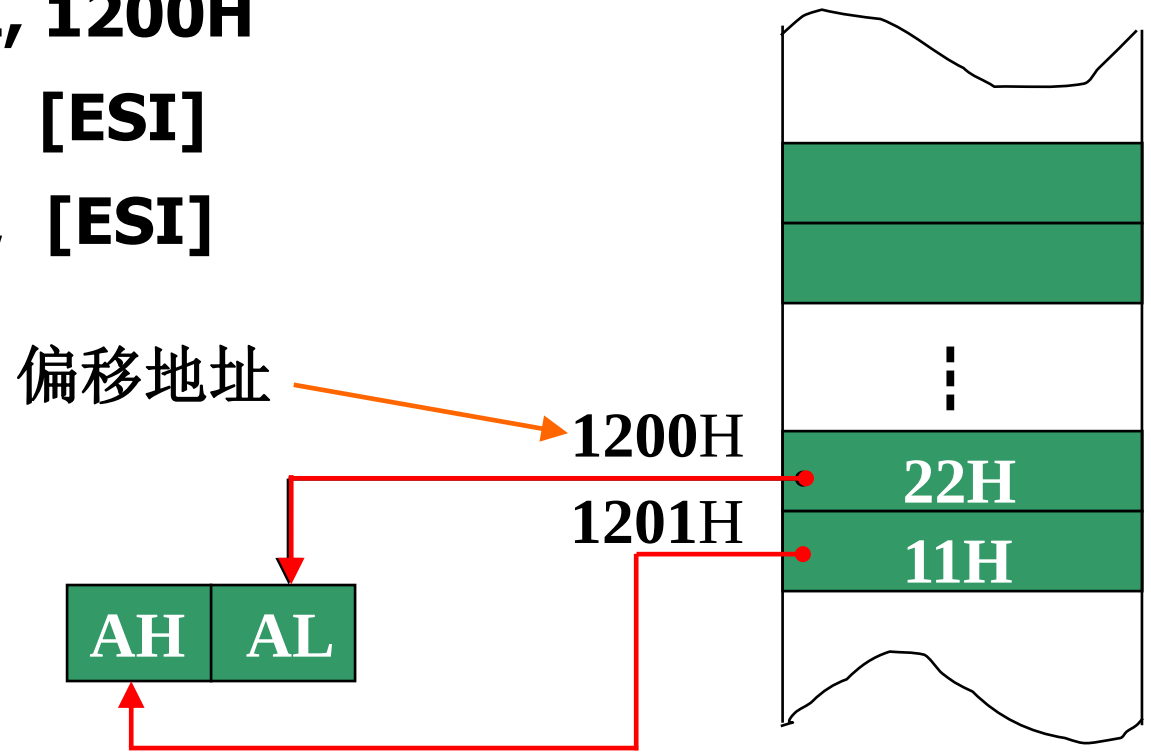
寄存器

或 符号地址（可以省略方括号）

寻找存储器操作数的关键是确定数据在内存中的存放地址

存储器操作数举例

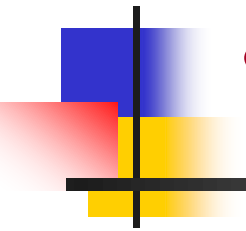
- 例：
  - MOV ESI, 1200H
  - MOV AL，[ESI]
  - MOV AX，[ESI]



四、指令的执行速度

- 指令的字长影响指令的执行速度

- 对不同的操作数，指令执行的时间不同：



§3.2 寻址方式(32位模式)

寻址方式

寻找操作数所在地址的方法

寻找转移地址的方法

本节

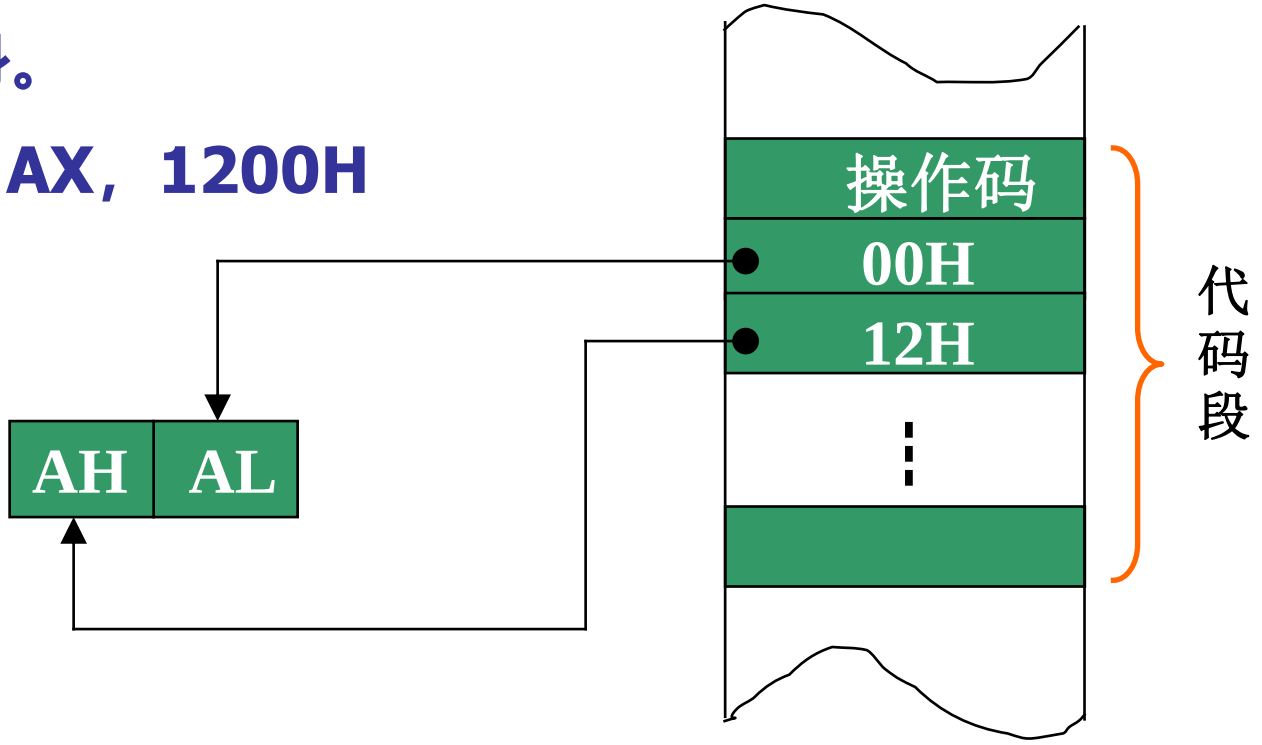
寻址方式

- **操作数可能的来源或存放处：**
  - **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** --变化最多

一、立即寻址

立即寻址只能用于指令的源操作数！

- 指令中的源操作数是立即数，即源操作数是参加操作的数据本身。
- 例：MOV AX，1200H



注意：立即数不能超过目的操作数表示范围

例：MOV AL, 260 ; ✗ 260超过8位的AL范围0~255

二、寄存器寻址

- 参加操作的操作数在CPU的通用寄存器或段寄存器中。

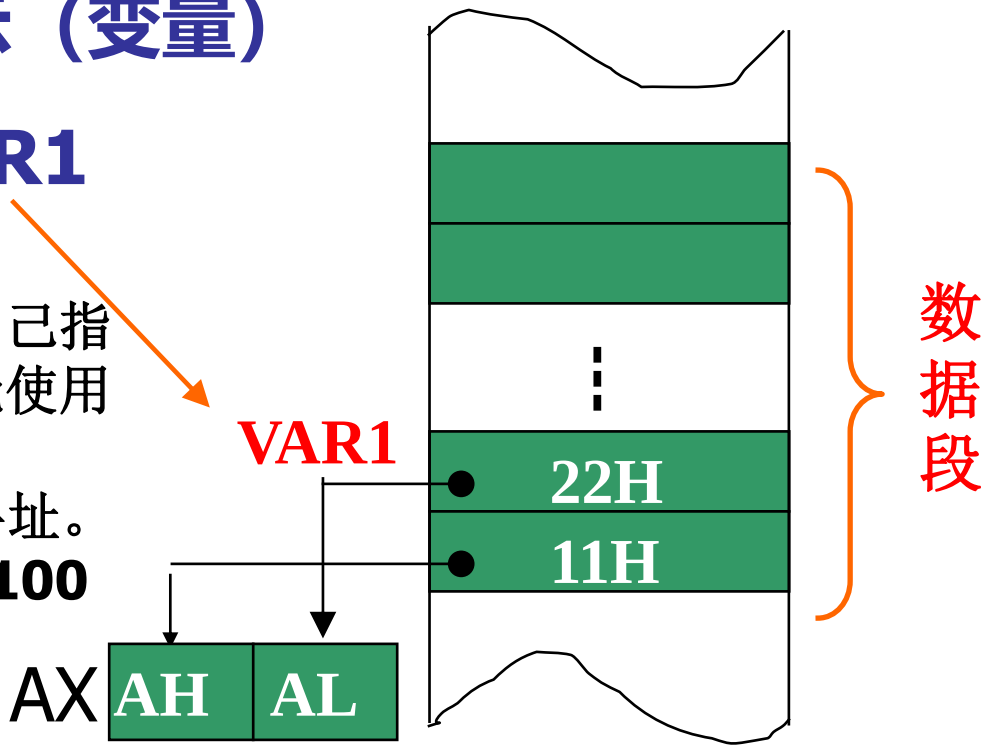
- 例：MOV AX，BX

三、直接寻址

- 指令中直接给出操作数的偏移地址
- 偏移地址用符号表示（变量）
- 例：MOV AX，VAR1

注意：在保护模式下，用户是不能自己指定一个具体的存储单元地址，即不能使用常数作为存储单元地址。

若使用了常数则被视为立即数寻址。例如，**MOV AX, [100]**与**MOV AX,100**为相同功能。



- **注意点：**
  - **存储器操作数的长度与指令中另一个操作数的长度要一致。**
  - **例：**

```
VAR1 BYTE  10
.....
MOV  CL, VAR1
MOV  AX, VAR1 ;出错,长度不一致
```

- **注意点（续）：**
  - **在16位模式下，直接寻址方式可以使用常数来表示**

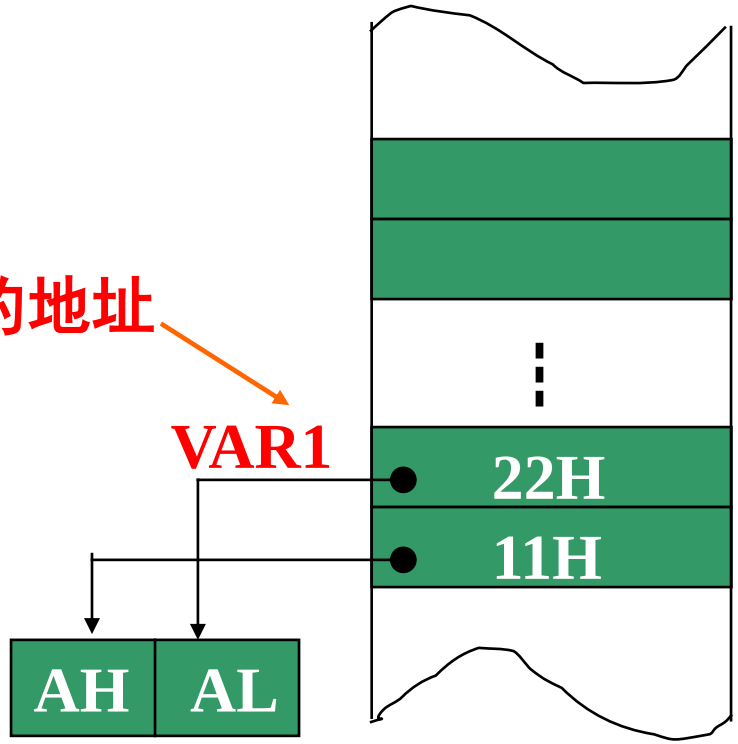
    例：MOV AX，[1200H]
  - **但在32位模式下这种形式就表示立即数，而不是直接寻址方式。它与下面指令等效：**

    MOV AX, 1200H

四、寄存器间接寻址

- 参与操作的数据存放在内存中，其偏移地址为指令中的一个32位寄存器的内容。
- 例：LEA EBX，VAR1

  MOV AX，[EBX]



- **注意点：**
  - **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.**
  - **间址寄存器为8个32位通用寄存器：EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP**
  - **不能使用16位模式下的16位寄存器：AX、BX、CX、DX、SP、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：**
    - **EBP、ESP：默认在堆栈段（SS）**
    - 其他的寄存器默认的段都是在数据段
    - 允许修改默认段寄存器，即段超越
  - **保护模式下不能直接访问具体的存储单元，故间址寄存器不能由用户指定一个具体数值，否则将发生存储器访问冲突错误。**

五、寄存器相对寻址

- 操作数的偏移地址为间址寄存器的内容加上一个位移量，位移量可以是一个常数，也可以是符号（定义的变量）。
- 使用示例：
  - MOV AX, [EBX+DATA]
  - 或：MOV AX, DATA[EBX]
  - 或：MOV AX, [EBX]+DATA
  - MOV CL, [EBX+5]
  - 或MOV CL, 5[EBX]
  - 或MOV CL, [EBX]+5
  - 或MOV CL, 5+[EBX]
  - 但不能是 MOV AX, [EBX]DATA或MOV CL, [EBX]5

位移量不能放在括号后面

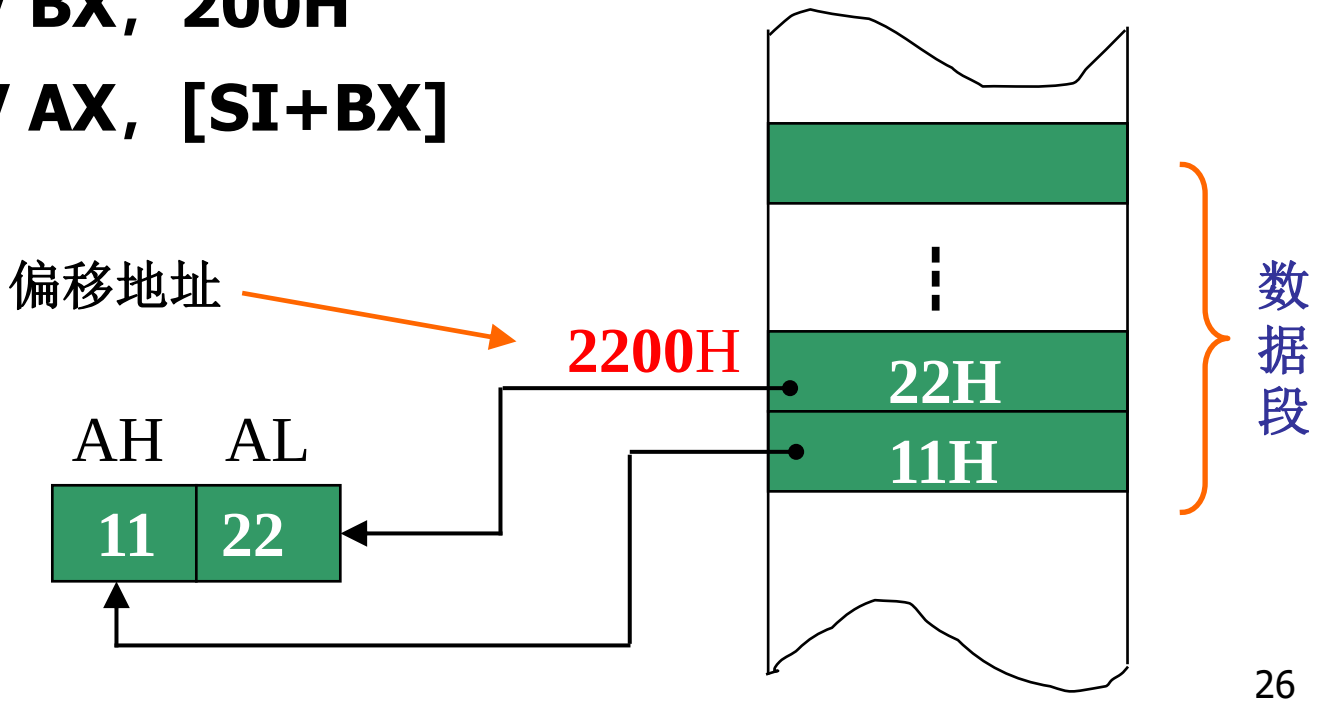
六、基址-变址寻址

1. 对于16位模式

- **操作数的偏移地址为**
  - **基址寄存器的内容 + 变址寄存器的内容；或：**
  - **基址寄存器内容+变址寄存器内容+位移量**
  - **基址寄存器：BX和BP，变址寄存器：SI和DI**
- **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**

例：

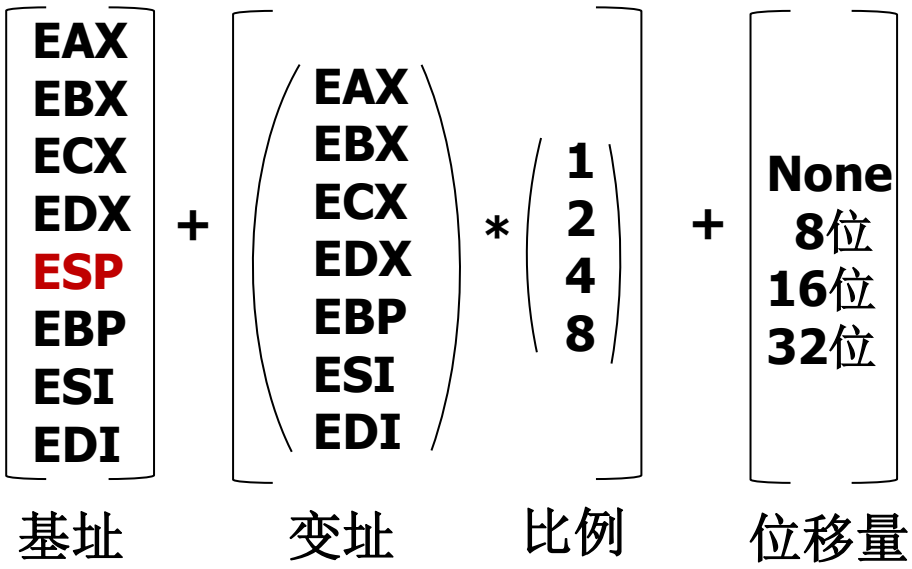
- 执行下列程序段：
  - MOV SI，2000H
  - MOV BX，200H
  - MOV AX，[SI+BX]



2. 对于32位模式

- 32位的变址寄存器可以乘上一个比例常数2、4和8，从而形成比例变址寻址方式、基址比例变址寻址方式和基址比例变址位移寻址方式。

上述三种情况可用公式表示：



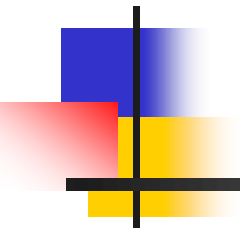
例如：

```
MOV  EBX, ADDR[ESI*2]       ;比例变址寻址方式
MOV  EAX, [EDI*4][EDX]      ;基址比例变址寻址方式
MOV  EBX, [EDI*8][EBP+10]   ;基址比例变址位移寻址方式
```

七、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
- 指令执行功能：
  - AL×BL ⟶ AX

隐含了寄存器操作数AL以及存放结果的AX



§3.3 80x86指令系统

掌握：

- 指令的功能
- 指令对操作数的要求
- 指令对标志位的影响：
  - 指令执行是否影响标志位
  - 对加、减运算，需明确指令执行如何影响标志位

整数操作类指令

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

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

一、通用数据传送

- 一般数据传送指令
- 堆栈操作指令
- 交换指令
- 字位扩展指令

- 特点：
  - 该类指令的执行对标志位不产生影响

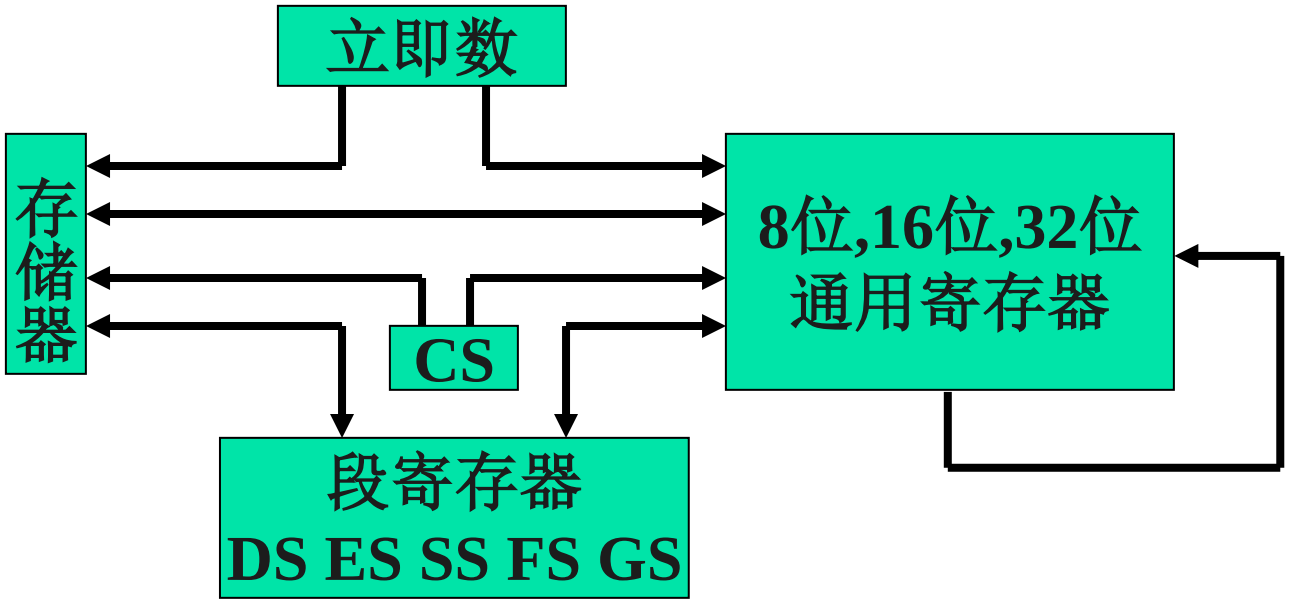
1. 一般数据传送指令

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

MOV指令使用上的注意

- 两操作数长度必须相同；
- 存储单元之间不能直接传送；
- 段寄存器CS只能作源操作数，段寄存器之间不能直接传送；
- 在源操作数是立即数时，目标操作数不能是段寄存器；
- 标志寄存器（EFLAGS或FLAGS）一般不作为操作数在指令中出现。
- MOV指令不影响标志位

MOV指令的传送方向示意图



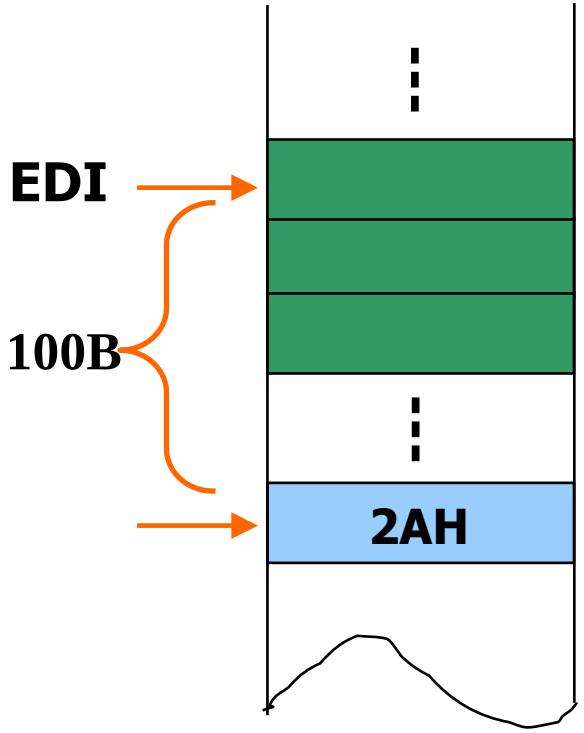
MOV传送指令示例

- **判断下列指令的正确性：**
  - **MOV AL，BX** ×
  - **MOV AX，[ESI]05H** × 位移量表示错
  - **MOV [BX][SI]，AX** × **32**位模式下出错
  - **MOV DS，1000H** ×
  - **MOV DX，09H** √
  - **MOV [1200]，SI** × **32**位模式下出错
  - **MOV AX，CS** √
  - **MOV DS，CS** ×

一般数据传送指令应用例

- 将符号“*”的ASCII码2AH送入内存数据段中以变址指针EDI所指的单元再偏移100个字节单元中。

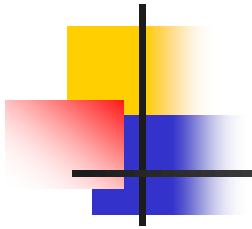
```
MOV   AL,  ‘*’或MOV AL,2AH
MOV   100[EDI], AL
```



零扩展和符号扩展传送指令

MOVZX/MOVSX

- MOV指令要求两个操作数的位数一致，如果要把位数短的操作数传到较长的操作数，则可使用零扩展指令或符号扩展指令。
- MOVZX指令有三种形式：
  - MOVZX reg32,reg/mem8
  - MOVZX reg32,reg/mem16
  - MOVZX reg16,reg/mem8
- 指令执行后，目的操作数比源操作数长的高位部分全部用0填充。
- 指令将短数传送到长数时把数字视为无符号数。



- 例如：

```
mov bx, 0A68CH
movzx eax, bx    ;EAX=0000A68CH
movzx edx,bl     ;EDX=0000008CH
movzx cx,bh       ;CX  =00A6H
```

- MOVSX指令有三种形式：
  - MOVSX reg32,reg/mem8
  - MOVSX reg32,reg/mem16
  - MOVSX reg16,reg/mem8
- 指令执行后，目的操作数比源操作数长的高位部分全部用源操作数的最高位填充。
- 指令将短数传送到长数时把数字视为带符号数。
- 例如：

```
mov bx, 0A68CH
movsx eax, bx    ;EAX=FFFFA68CH
Movsx edx,bl     ;EDX=FFFFFF8CH
movsx cx,bh      ;CX  =FFA6H
```

2. 堆栈操作指令

(1) 实模式

- **堆栈操作以字为单位**
  - **堆栈操作指令的操作数必为16位**
- **堆栈指令对操作数的要求：**
  - **不能是立即数；**
  - **可以是16位寄存器或存储器的1个字单元；**
  - **若为存储器操作数，需要声明为字存储单元。**

堆栈操作指令

- **压栈指令 PUSH**
  - **格式:　PUSH　OPRD**

16位寄存器或
存储器字单元

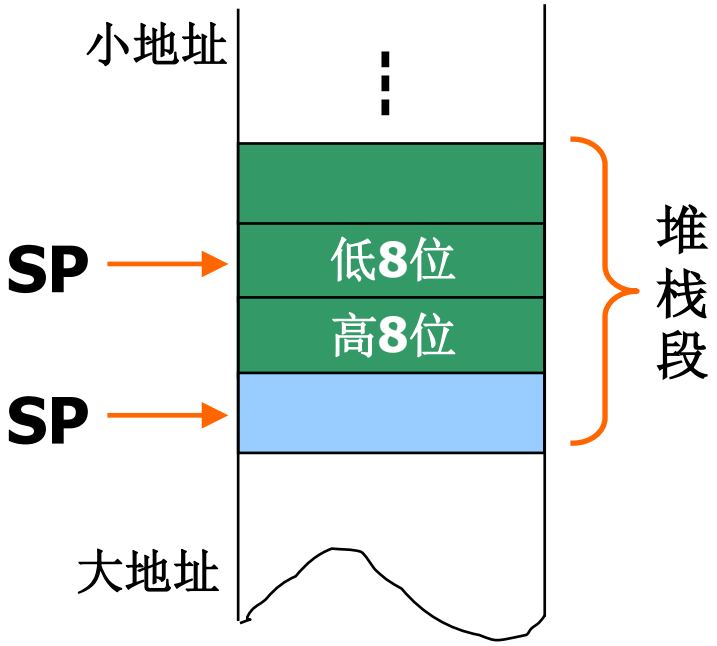
- **出栈指令 POP**
  - **格式: POP　OPRD**

16位寄存器或
存储器字单元

- **标志寄存器FLAGS进栈/出栈指令**
  - **格式: PUSHF 或 POPF**

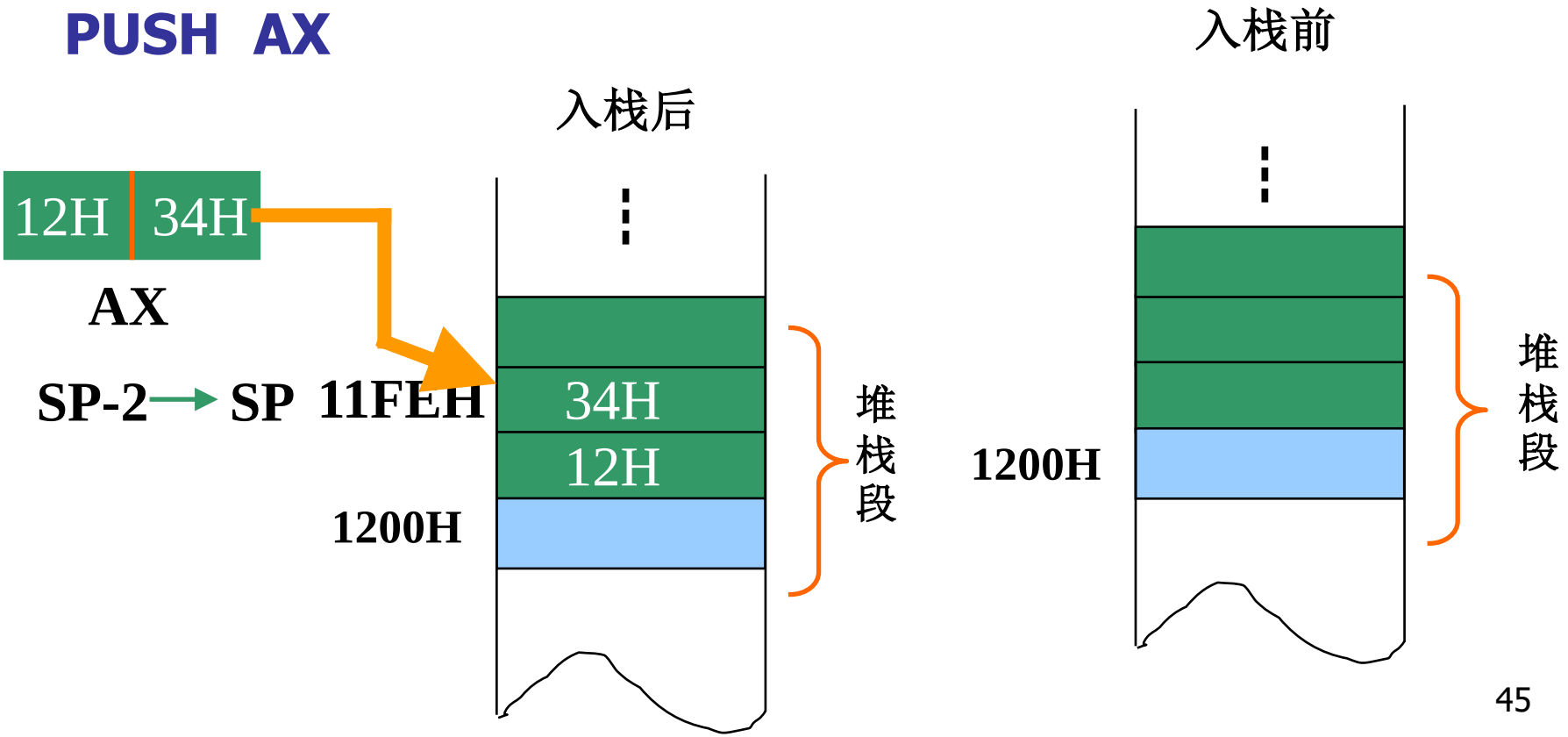
压栈指令 PUSH

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



压栈指令的操作示例

MOV AX，1234H
MOV SP，1200H
PUSH AX



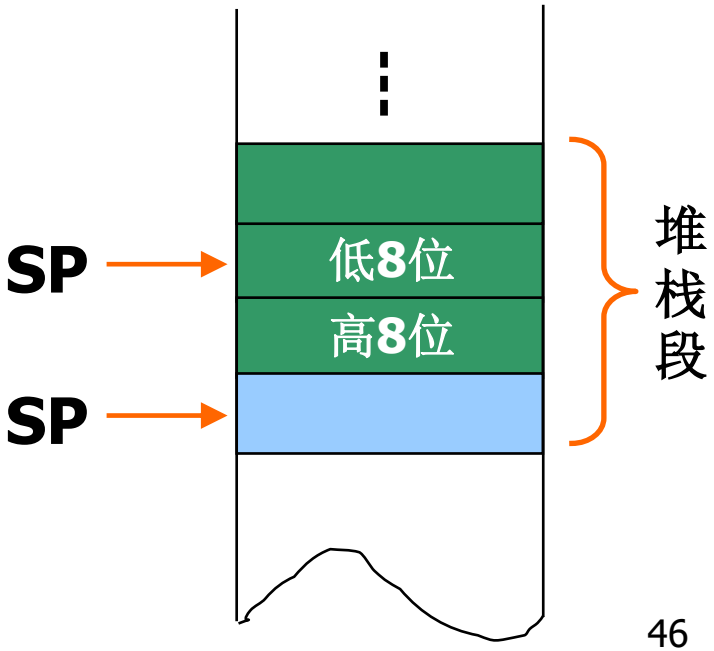
出栈指令POP

- **指令执行过程：**

(SP)操作数低字节 ⟶ 弹出

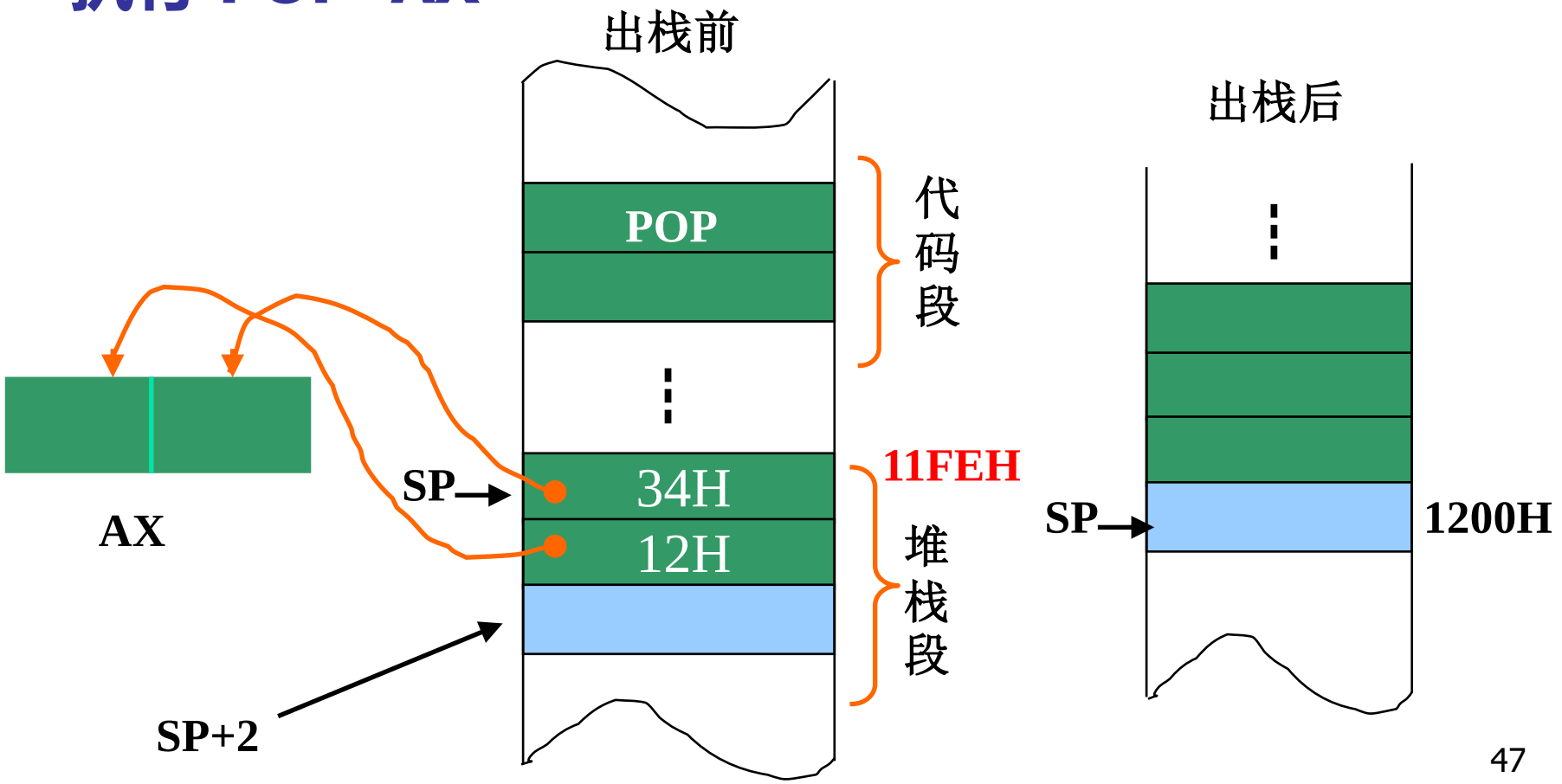
(SP+1)操作数高字节 ⟶ 弹出

SP ← SP+2



出栈指令的操作示例

执行 POP AX



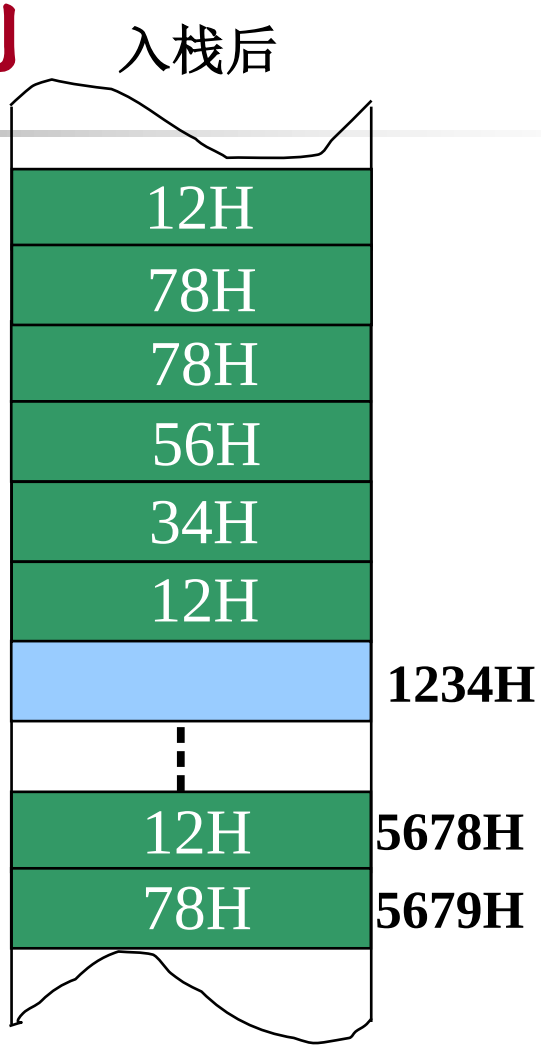
使用注意：

- 指令的操作数必须是16位；
- 操作数可以是寄存器或存储器字单元，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

（2）32位保护模式

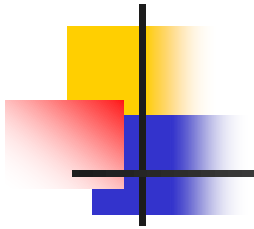
前面介绍的实模式下的指令都可以使用，还可以使用32位寄存器和存储单元，另外增加了以下指令。

- **PUSH允许立即数入栈**

例如：PUSH 0ABCDH　　;将16位立即数入栈

PUSH0ABCD0000H　;将32位立即数入栈

- **新增两条入栈指令：PUSHA和PUSHD**
  - **PUSHA：将8个16位通用寄存器按AX、CX、DX、BX、SP、BP、SI与DI的顺序入栈。**
  - **PUSHD：将8个32位通用寄存器按EAX、ECX、EDX、EBX、ESP、EBP、ESI与EDI的顺序入栈。**



- 新增两条出栈指令：POPA和POPD
  - POPA指令从栈顶弹出8个字数据分别送入DI、SI、BP、SP、BX、DX、CX与AX。
  - POPD指令从栈顶弹出8个双字数据分别送入EDI、ESI、EBP、ESP、EBX、EDX、ECX与EAX。
- 新增32位标志寄存器EFLAGS进栈/出栈指令
  - 格式：PUSHFD 或 POPFD

3. 交换指令

- 格式:

  XCHG REG/MEM，MEM/REG

- 注:
  - 两操作数至少有一个是寄存器操作数
  - 不允许使用段寄存器。
- 例:
  - XCHG AX，BX
  - XCHG [EBP]，CL

二、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
  - 输出指令 ： OUT PORT，acc
  - PORT为端口地址；acc为累加器，在16位模式可以是AL或AX，在32位模式还可以是EAX。

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PORT表现形式（16位模式和32位模式都一样）。
- 直接寻址
  - 端口地址为8位时，指令中的PORT直接由一个8位无符号常数；
  - 可寻址256个端口。
- 间接寻址
  - 当端口地址超过255时，地址码为16位，指令中的端口地址必须由DX指定(不能使用EDX指定)；
  - 可寻址64K个端口。
  - 当端口地址为0~255时也可以用DX间接寻址

I/O指令例

- IN AX，80H；从80H端口输入一个字
- MOV DX，2400H
- IN AL，DX ；从2400H端口输入一个字节
- OUT 35H，EAX；向35H端口输出一个双字数据

三、取偏移地址指令LEA

- 操作：
  - 将一个存储单元的16位(实模式)或32位(保护模式)偏移地址取出送目标寄存器
- 当程序中用符号地址表示内存偏移地址时，可以使用该指令方便获得偏移地址。
- 格式：

```
LEA REG, MEM
```

- 指令要求：
  - 源操作数必须是一个存储器操作数，目标操作数必须是16位或32位通用寄存器。

- 在实模式下：

REG为**16**位通用寄存器，例如：

LEA AX, [SI][DI]；指令将**SI**和**DI**的内容相加送入**AX**中

- 在保护模式下：

REG可以是**16**位通用寄存器或**32**位通用寄存器

- 若为**16**位寄存器，则取**32**位偏移地址的低**16**位装入该寄存器中。这时就不能直接用该寄存器去访问存储单元。

例如：**LEA SI, [EDI]** ;**SI**中只存储了**EDI**的低**16**位

- 若为**32**位寄存器，则将存储单元的**32**位偏移地址装入该寄存器。

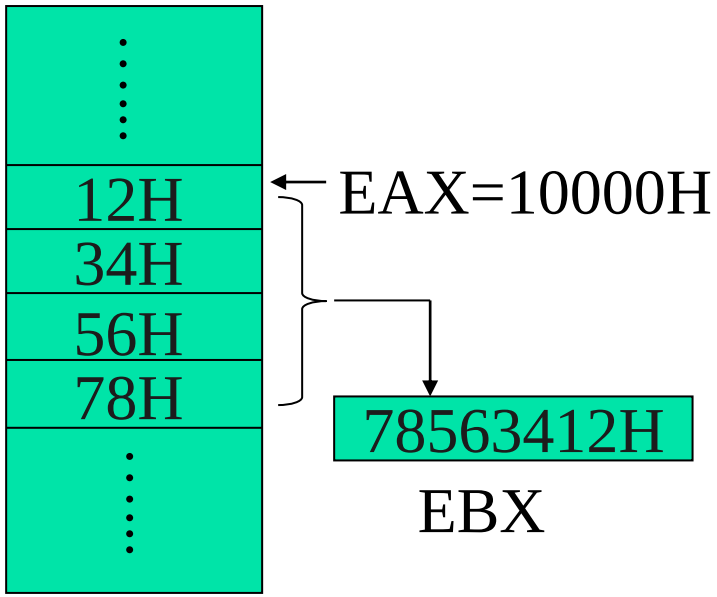
例如：**LEA ESI, [EAX][EBX]**

指令执行后**ESI**的内容为**EAX**和**EBX**两寄存器内容之和。

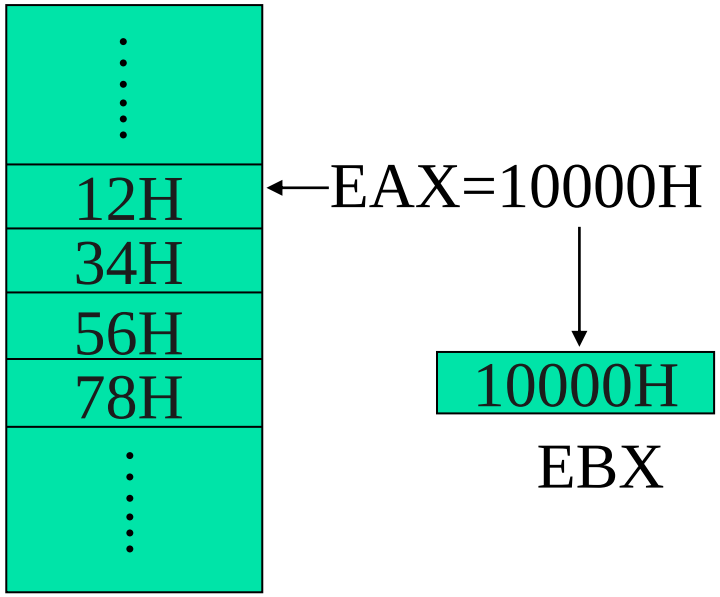
例：比较指令 MOV EBX，[EAX]与LEA EBX，[EAX]

设EAX=10000H

MOV EBX, [EAX]



LEA EBX, [EAX]



又例如：

符号地址

比较下列指令：

MOV ESI，DATA1

执行结果：**ESI=12345678H**

LEA ESI，DATA1

执行结果：**ESI=123H**

MOV EAX，[EBX]

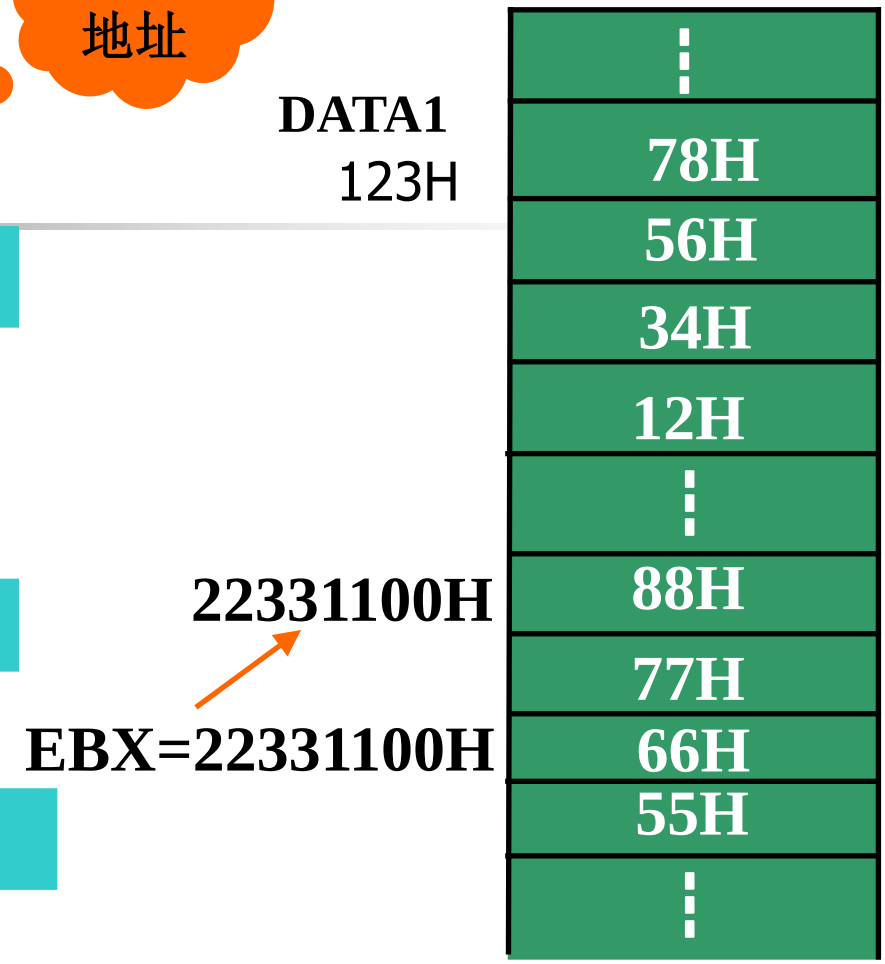
执行结果：**EAX=55667788H**

LEA EAX，[EBX]

执行结果：**EAX=22331100H**

LEA CX，[EBX]

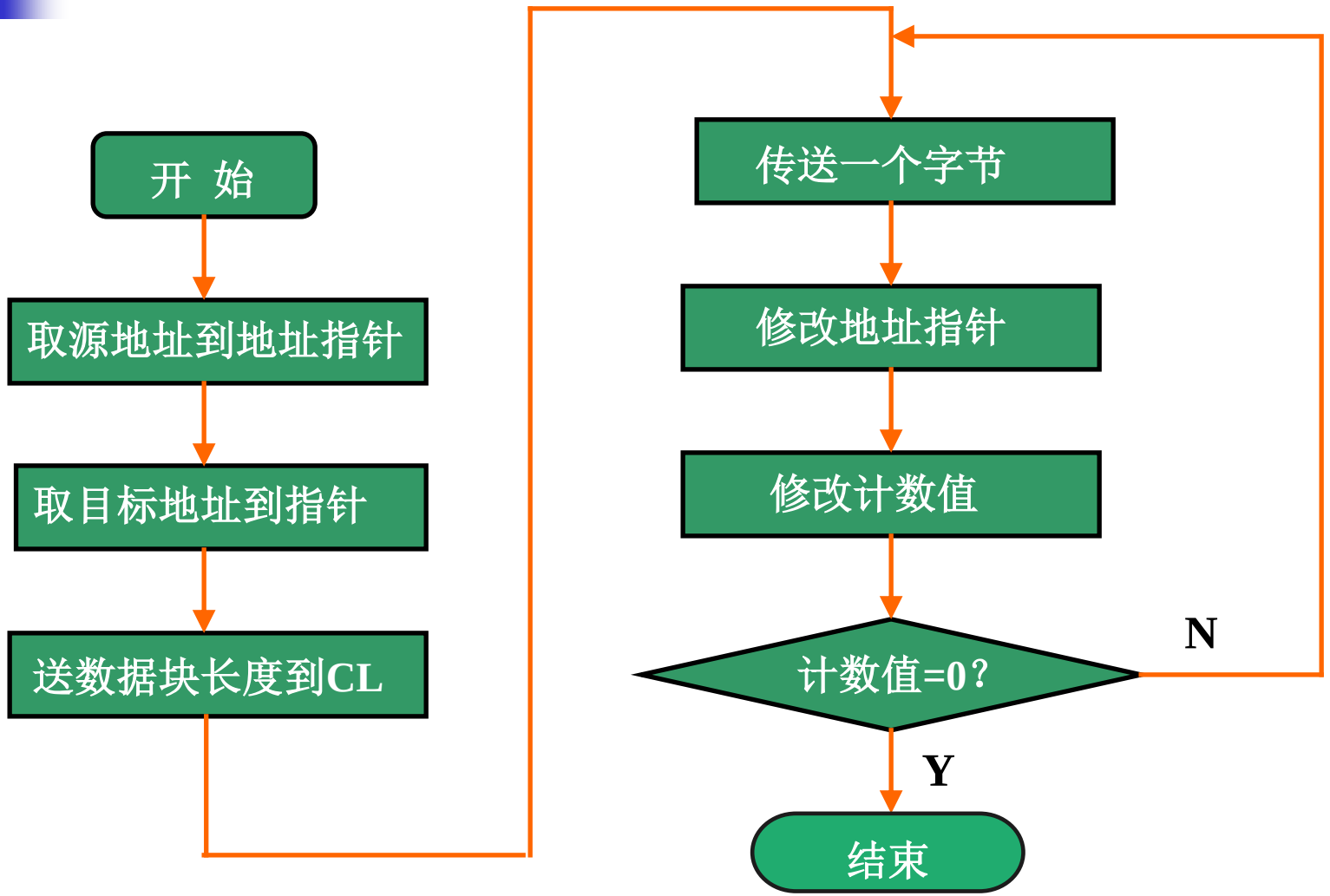
执行结果：**CX=1100H**



LEA指令在程序中的应用示例

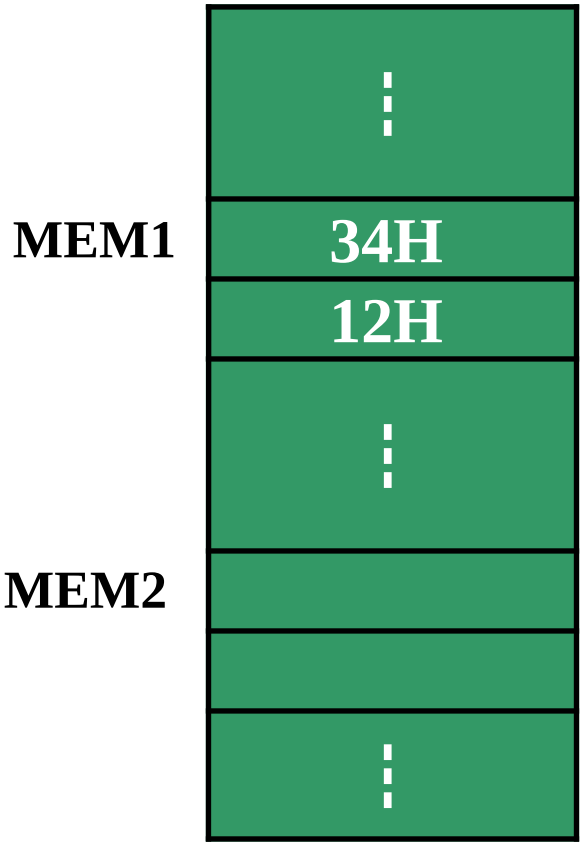
- 将数据段中首地址为MEM1 的50个字节的数据传送到同一逻辑段首地址为MEM2的区域存放。编写相应的程序段 。

LEA指令在程序中的应用示例



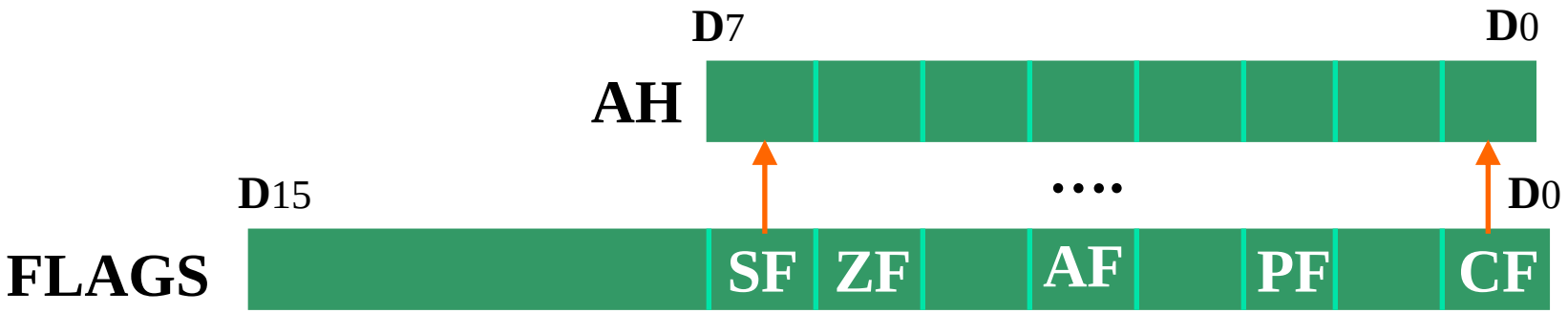
LEA指令在程序中的应用示例

```
        LEA  ESI, MEM1
        LEA  EDI, MEM2
        MOV  CL, 50
NEXT:   MOV  AL, [ESI]
        MOV  [EDI], AL
        INC   ESI
        INC   EDI
        DEC   CL
        JNZ   NEXT
```



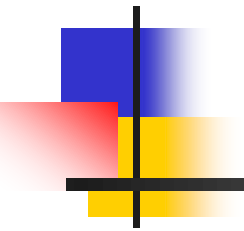
四、标志位操作指令LAHF/SAHF

- LAHF
- 操作:
  - 将FLAGS的低8位装入AH



- SAHF

执行与LAHF相反的操作



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

一、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指令基本相同

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

```
  01111000
+ 10011001
-----------
 [1]00010001
```

标志位状态： CF= 1　　SF= 0

AF= 1　　ZF= 0

PF= 1　　OF= 0

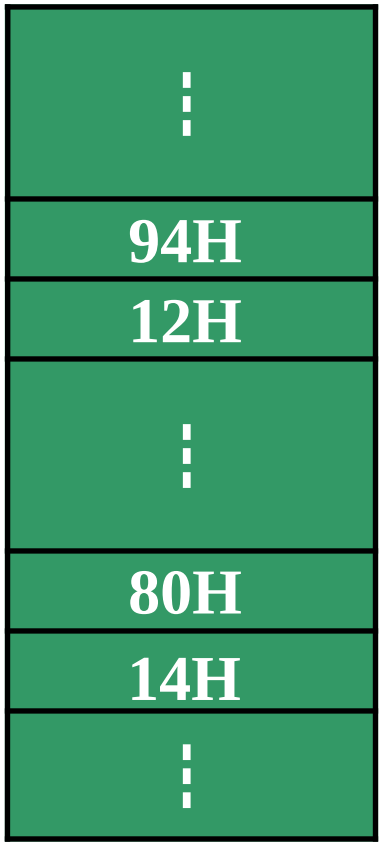
2. ADC指令

- 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
  - 注意：CF是该指令执行前的值
- ADC指令常用于多字节数相加，使用前要先将CF清零。

ADC指令应用例——求两个大数的和，两个数的长度都为20字节。

```
        LEA  ESI，M1
        LEA  EDI，M2
        MOV CL，20
        CLC                    ；使CF=0
NEXT ： MOV AL，[ESI]
        ADC [EDI]，AL
        INC ESI
        INC EDI
        DEC CL
        JNZ NEXT
        HLT
```

为什么要用ADC?



3. INC指令

- 格式：

不能是段寄存器
或立即数

INC OPRD

- 操作：

OPRD+1 ⟶ OPRD

注意：INC指令执行不影响CF标志

常用于在程序中修改地址指针

二、减法指令

- 普通减法指令SUB
- 带借位的减法指令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

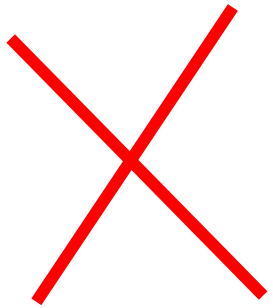
- 格式:
  - DEC OPRD
- 操作:
  - OPRD - 1 ⟶ OPRD

指令对操作数的要求以及对标志的影响与INC相同

指令常用于在程序中修改计数值

例如　实现一个计数循环程序

```
        MOV  AL，10H
LOP：   DEC   AL
        JNC   LOP
```



上述程序段中错误地使用了CF标志位，DEC指令不影响CF标志。

应用程序示例

```
        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

8/16/32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，但不是补码。

对标志位的影响

- NEG指令将影响标志PF、AF、ZF、SF、CF和OF。
- 对进位标志CF的影响
  - 只有当操作数为零时，进位标志CF被置零，其它情况都被置1
- 对溢出标志OF的影响
  - 当字节操作数为-128（80H），或字操作数为-32768（8000H）或双字操作数为80000000H时，结果将无变化，但溢出标志OF被置1.

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！
- 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

- **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**

两个数大小的比较

- 比较两个无符号数：

例如：CMP AX，BX

两个数的大小由CF或ZF来判断

若 AX > BX　　CF=0,ZF=0

若 AX < BX　　CF=1,ZF=0

若 AX = BX　　ZF=1

有专门的判断无符号数大小指令：

JA/JAE/JB/JBE

即在比较指令后执行判断指令

- **比较两个带符号数**

CMP AX，BX

两个数的大小由OF、SF和ZF共同决定

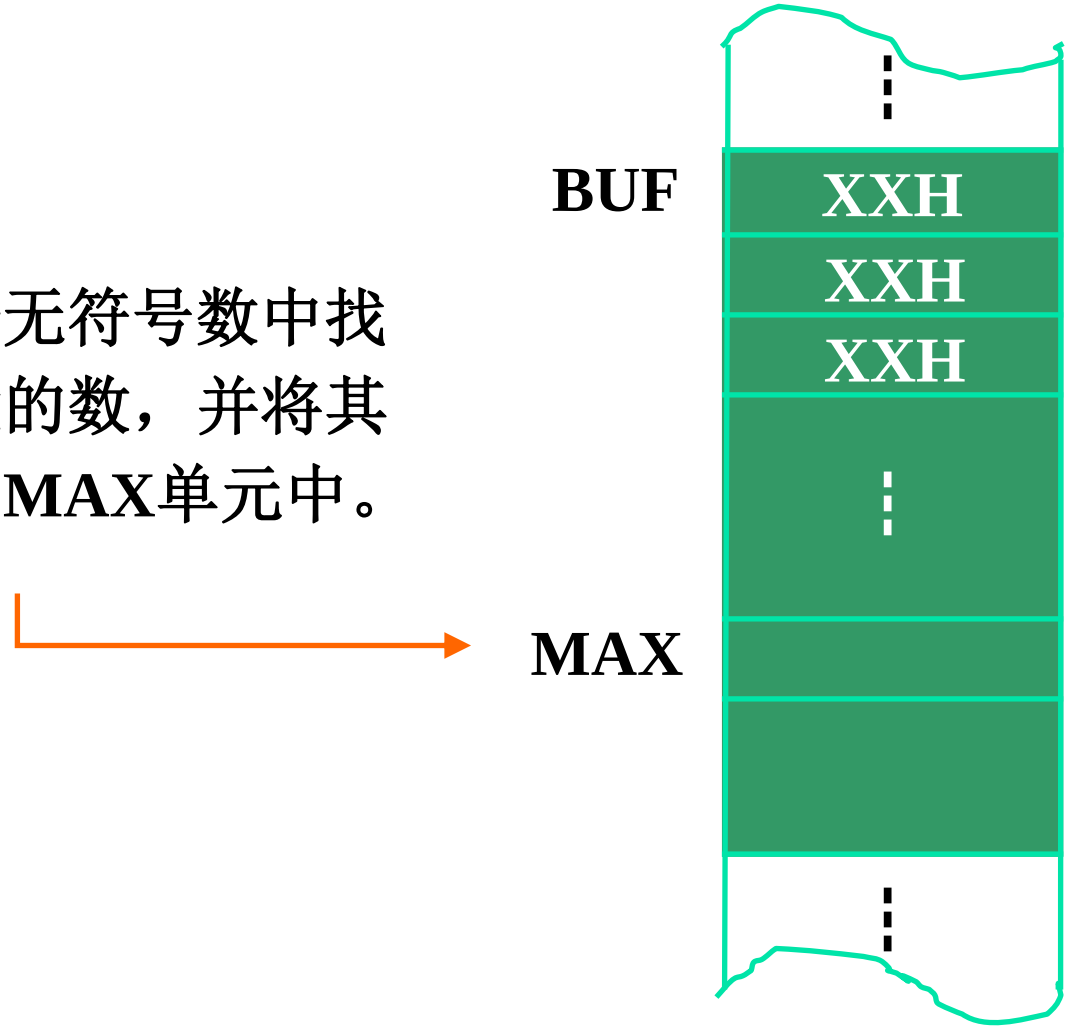
OF=SF,ZF=0	AX > BX
OF≠SF,ZF=0	AX < BX
ZF=1	AX=BX

有专门的判断带符号数大小的指令：

JG/JGE/JL/JLE

CMP指令示例

在20个无符号数中找出最大的数，并将其存放在MAX单元中。



实现程序

```
        LEA EBX，MAX
        LEA ESI，BUF
        MOV CL，20-1
        MOV AL，[ESI]
NEXT:   INC  ESI
        CMP AL，[ESI]
        JNC  GOON     ；CF=0转移
        MOV  AL, [ESI]
GOON：  DEC  CL
        JNZ  NEXT
        MOV [EBX]，AL
        HLT
```

三、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：
  - 运算结果长度是乘数的两倍，即8位->16位，16位->32位…
  - 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或EAX及存放结果的AX，DX，EDX；
  - 若运算结果的高半部分是无效数值，则OF=CF=0，否则OF=CF=1。

1. 无符号数乘法指令

指令格式：**MUL　OPRD**

- **OPRD**提供乘法运算的一个操作数，只能是寄存器或存储器操作数。
- 指令的另一操作数隐含使用**AL**或**AX**或**EAX**寄存器。
- 结果存放在**AX(**字节运算**)**或**DX:AX(**字乘法**)**或**EDX:EAX(**双字乘法**)**中。

- 字节运算：AX<=(AL) ×(OPRD)
- 字运算 ：DX:AX<= (AX)×(OPRD)
- 双字运算：EDX:EAX<=(EAX) ×(OPRD)

- **MUL只对CF和OF标志产生有效影响,其他标志位的值不确定。**
- **若结果的AH(字节运算)或DX(字运算)或EDX(双字运算)为全0，则CF=OF=0，否则CF=OF=1。**

2、带符号数乘法指令IMUL

指令格式：**IMUL　OPRD**

IMUL的功能除了操作数是带符号外，其余与MUL指令相同。

对标志位的影响：若乘积的高半部（**AH或DX或EDX**）是低半部的符号扩展(不是有效数值)，则**CF=OF=0** 。否则**CF=OF=1** 。

例如，对于字节乘法：

- 若乘积的(AH)=11111111，且AL最高位为1，表示符号扩展，则CF=OF=0；
- 若乘积的(AH)=00000000，且AL最高为0，表示符号扩展，则CF=OF=0；
- 若乘积的(AH)=11111111，但AL最高位为0，不是符号扩展，则CF=OF=1；
- 若乘积的(AH)=11111110，不是符号扩展，则CF=OF=1；
- 若乘积的(AH)=00000010，不是符号扩展，则CF=OF=1。

3. 32位模式新增IMUL指令格式

- 双操作数格式: IMUL DEST, SRC

  指令功能：DEST<=(DEST)×(SRC)

  - 16位操作数格式：
    - IMUL reg16,reg/mem16
    - IMUL reg16,imm8/imm16
  - 32位操作数格式：
    - IMUL reg32,reg/mem32
    - IMUL reg32,imm8/imm16/imm32
- 三操作数格式: IMUL DEST, SRC1, SRC2

  指令功能：DEST<=(SRC1) ×(SRC1)
  - IMUL reg16,reg/mem16,imm8/imm16
  - IMUL reg32,reg/mem32,imm8/imm16/imm32
- 如果存放结果的目的寄存器中丢弃了乘积高位的有效数值，则CF=OF=1，因此使用这些指令后应该检查CF或OF标志。

四、除法指令

无符号除法指令格式：

DIV reg/mem8或DIV reg/mem16或DIV reg/mem32

有符号除法指令格式：

IDIV reg/mem8或DIV reg/mem16或DIV reg/mem32

指令中只给出除数，而被除数和商、余数都为隐含。

被除数	除数	商	余数
AX	reg/mem8	AL	AH
DX:AX	reg/mem16	AX	DX
EDX:EAX	reg/mem32	EAX	EDX

指令要求被除数是除数的双倍字长，因此除法指令常与扩展指令**CBW**、**CWD**、**CDQ**配合使用。

字位扩展指令CBW、CWD、CDQ

- 指令为零操作数指令，采用隐含寻址，隐含的操作数为AX、DX或EAX、EDX。

字节到字的扩展指令CBW

- 功能：将AL的符号位扩展到AH
- 规则：
  - 若AL最高位=1，则执行后AH=FFH
  - 若AL最高位=0，则执行后AH=00H

字到双字的扩展指令CWD

- 功能：将AX符号位扩展到DX
- 规则：
  - 若AX最高位=1，则执行后DX=FFFFH
  - 若AX最高位=0，则执行后DX=0000H

双字到四字的扩展指令CDQ

- 功能：将EAX符号位扩展到EDX
- 规则：
  - 若EAX最高位=1，则执行后EDX=FFFFFFFFH
  - 若EAX最高位=0，则执行后EDX=00000000H

字位扩展指令示例

判断以下指令执行结果：

```
MOV  AL，44H
CBW
MOV  AX，0AFDEH
CWD
MOV  AL，86H
CBW
```

五、BCD码调整指令

- 共6条（详见教材表3-3）：

  DAA，AAA，DAS，AAS，AAM，AAD

- 都是隐含寻址方式，隐含的操作数是:

  AL 或者 AX（AL与AH）

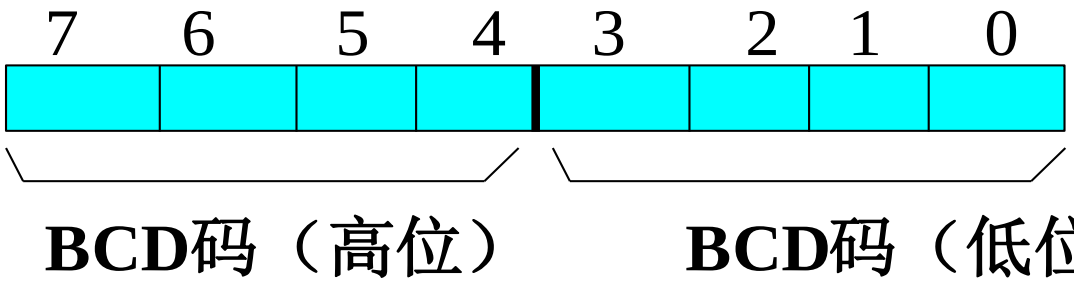
- 不能单独使用，要与相应的算术运算指令配合使用；加、减、乘、除指令。

- 执行结果为压缩BCD码或非压缩BCD码表示的十进制数。

十进制数的表示—BCD码

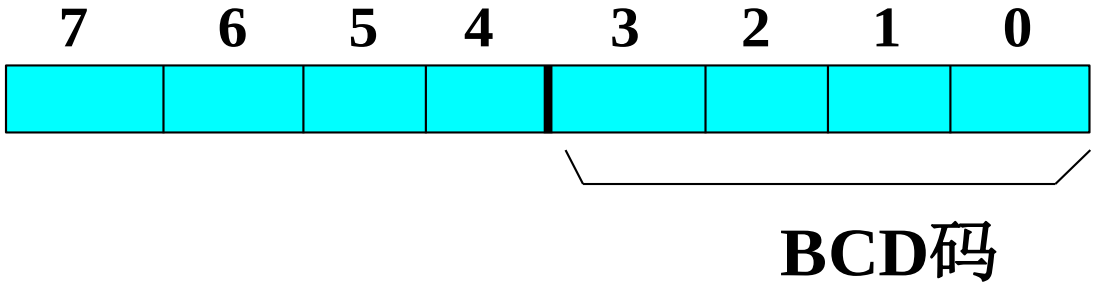
- 计算机中采用BCD码来表示十进制数。BCD码就是使用四位二进制数表示一位十进制数。
- BCD码分为两种格式：
  - 压缩型（组合型、装配型）
  - 非压缩型（非组合型、拆散型）

压缩型：一个字节表示两个**BCD**码，即两位十进制数。



例如：**0010 0011** 表示十进制数的**23**

非压缩型：一个字节的低四位表示一个**BCD**码，而高四位对所表示的十进制数没有影响，常为**0000**或**0011**。



例如：**0000 1001**与**0011 1001**都是十进制数**9**的非压缩型**BCD**码

十进制数(BCD码)的运算有两种方法

- **数制转换**: 先把十进制数转换为二进制数,再用二进制运算指令进行运算。最后将结果由二进制数转换为十进制数(BCD码)。
- **用BCD码处理指令**，直接进行十进制数运算。它又有两种方法:
  - 指令系统提供专门实现BCD码运算的加、减、乘、除运算指令。
  - 先用二进制数的加、减、乘、除运算指令对BCD码运算，再用BCD码调整指令对结果校正。--80x86就是用这种