第三章 80x86寻址方式与指令系统

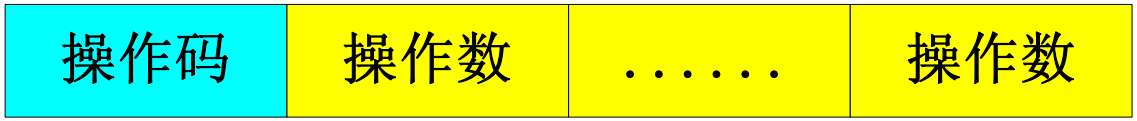
3-0 基础知识

一、指令系统概念

- 每种计算机都有一组指令集提供用户使用，这组指令集就称为计算机的指令系统。
- 机器指令：计算机能识别和执行的指令的二进制代码。如： **0000 0110 0010**
- 汇编指令：用助记符表示机器指令的操作码和操作数。

二、指令组成

- 计算机中指令由操作码字段和操作数字段两部分组成。



  - 操作码字段-----指示计算机要执行的操作
  - 操作数字段-----指出在指令执行操作过程中所需要的操作数；该字段可以是：

    1）操作数本身

    2）操作数地址或是地址的一部分；

    3）指向操作数地址的指针；

    4）或其他有关操作数的信息。

  - 大多数指令包括1～2个操作数，少数隐含第三个操作数。

三、操作数的存放

- 操作数的存放不外乎三种情况：

（1） 操作数包含在指令中

- 即指令的操作数字段包含操作数本身。这种操作数称为立即数。
- 例：**MOV AL , 08H ；** **08H**为立即数

（2）操作数包含在**CPU**的内部寄存器中

- 例：**INC CX** ；操作数存放在**CX**寄存器中

（3）操作数在内存数据区

- 操作数在内存数据区，操作数字段包含着此操作数的地址。

四、寻址

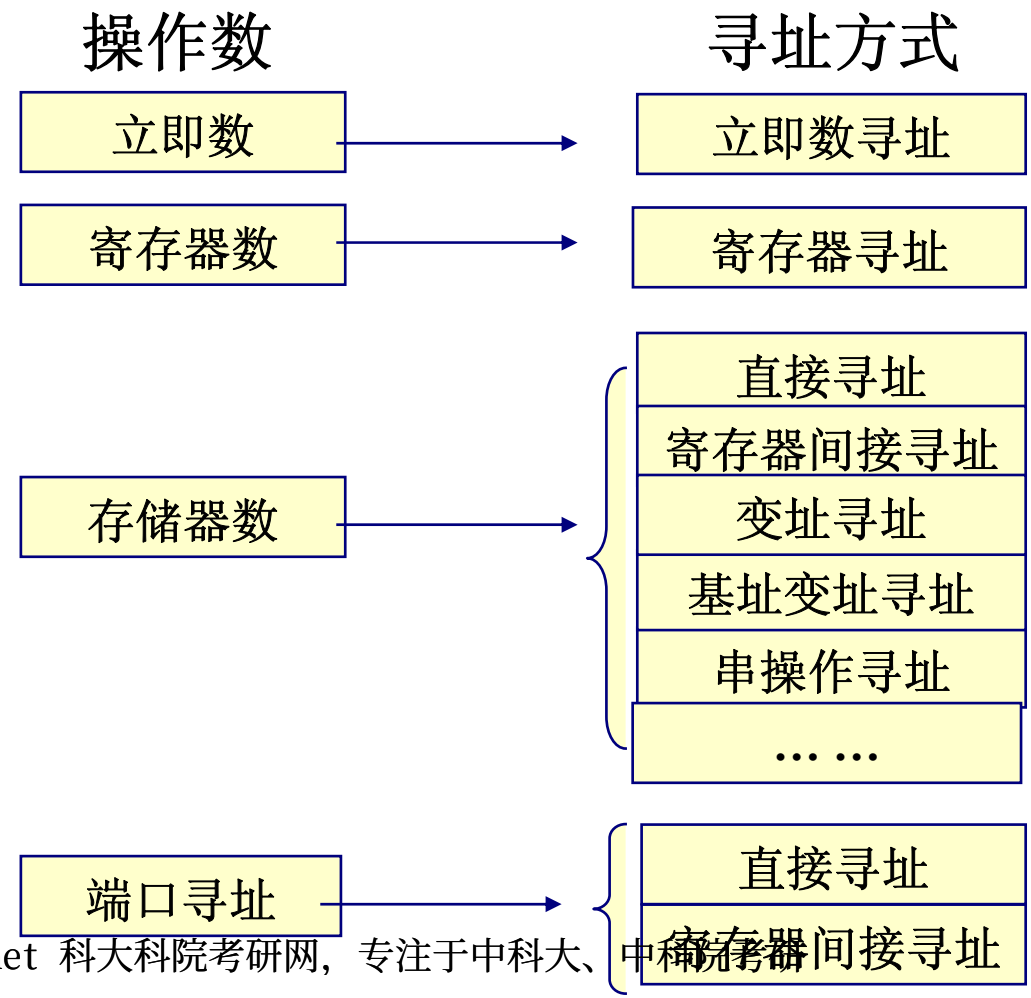
- 寻址——根据指令内容确定操作数地址的过程。

- 有效地址——根据寻址方式计算所得到的地址（**Effective Address-EA)**，也就是段内偏移地址。有效地址还需要与相应的段基地址组合才能产生物理地址（**PA**），这部分工作由**CPU**完成。

- 寻址方式——如何寻找内存操作数的方法。不同寻址方式实质上是构成段内的偏移量的方法不同。

80x86操作数的寻址方式

- 80x86计算机中操作数按存放的方法分为：
  - 立即数（指令中）
  - 寄存器数
  - 存储器数
  - I/O端口



五、80x86的数据类型与存储方式

- 80x86结构的基本数据类型
  - 字节：　　8位
  - 字：　　16位，2个字节
  - 双字：　　32位，4个字节
  - 四字：　　64位，8个字节（80486CPU引入）
  - 双四字：　　128位，16个字节（Pentium III)
- 数据在内存中存储方式
  - 多字节数据的存放原则：低位字节在低端地址，高位字节在高端地址。最低地址就是操作数的地址。
  - 对准存放。不是必须的，但是对准存放可以减少CPU读数据所需的操作步骤。

3-1 80x86的寻址方式

80x86的寻址方式可以分为10种

- 1、立即寻址
- 2、寄存器寻址
- 3、直接寻址
- 4、寄存器间接寻址
- 5、寄存器相对寻址
- 6、基址变址寻址
- 7、基址变址相对寻址
- 8、比例变址寻址
- 9、基址比例变址寻址
- 10、相对基址比例变址寻址

其中：方式3～方式10是与存储器有关的寻址方式；
方式8～方式10仅适用于80386及其后继CPU。

一、与存储器无关的二种寻址方式

1、立即寻址（Immediate addressing）

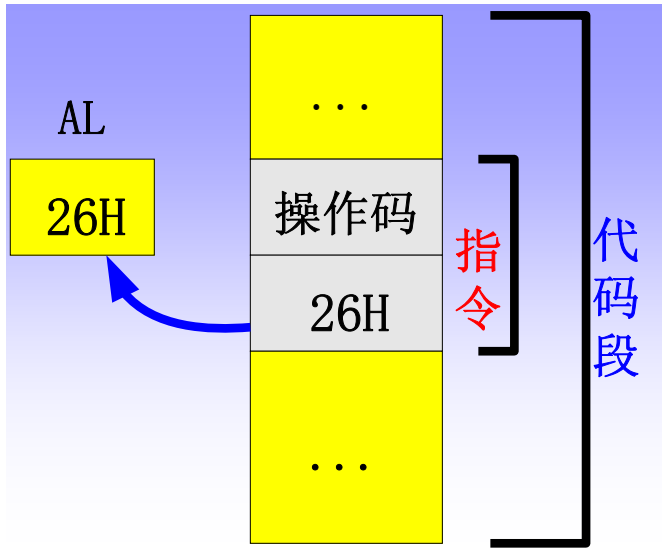
- 操作数以常量形式直接放在指令中，紧跟操作码之后。
- 机器码存放形式如下：

8位立即数

16位立即数

- 立即寻址示例1

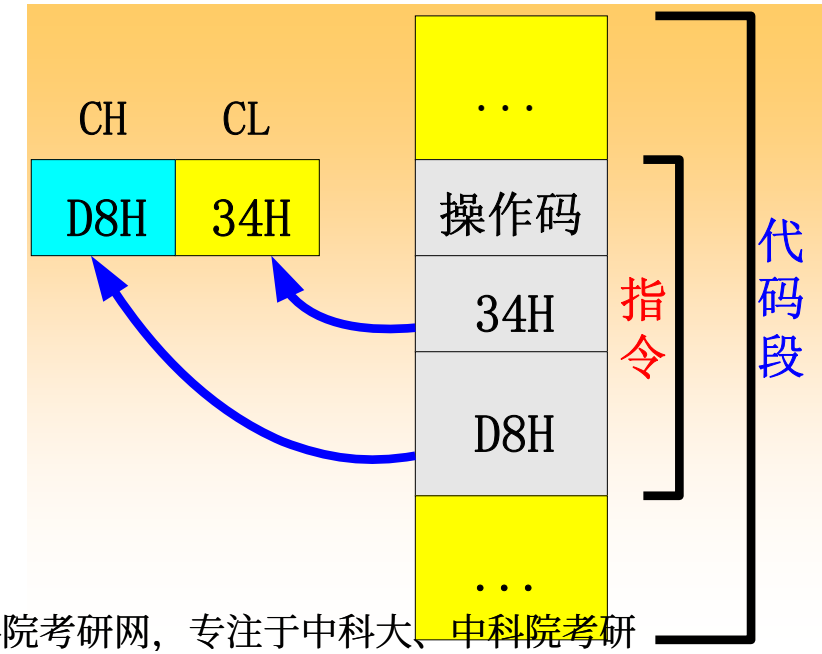
MOV AL，26H； 执行后26H →AL，即（AL）＝26H



- 立即寻址示例2

MOV CX， 0D834H；

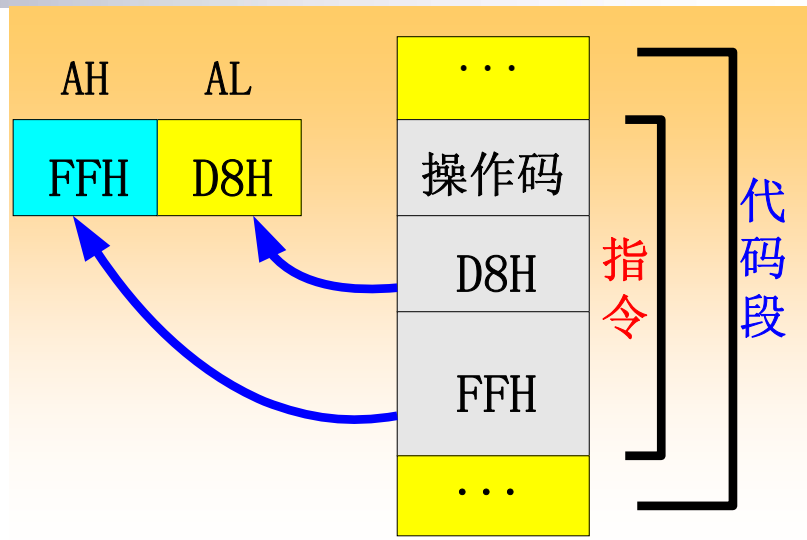
执行后（CX）＝0D834H



- 立即寻址示例3

MOV AX，-40；

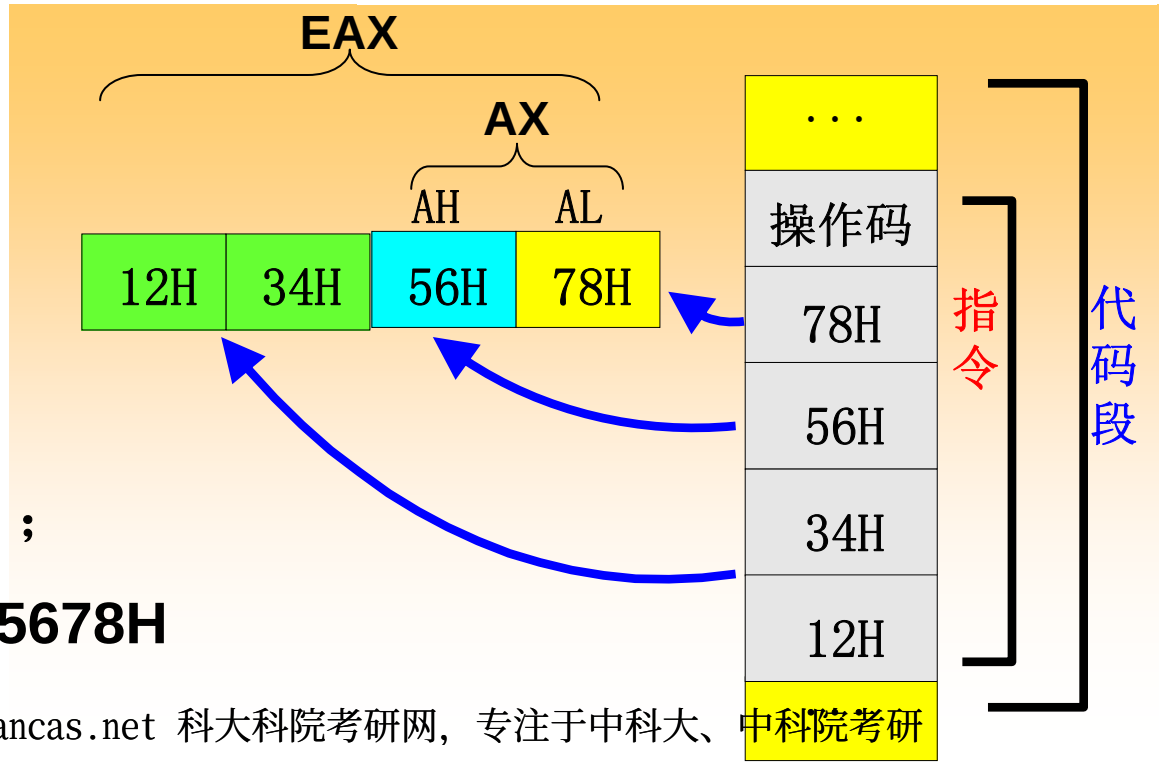
执行后（AX）＝ 0FFD8H（-40）



- 立即寻址示例4

MOV EAX, 12345678H ；

执行后（EAX）＝12345678H



- 关于立即寻址方式的几点说明：

1）立即寻址常用于给寄存器赋值；

2）立即数不仅能送到寄存器，而且也可送往存储单元；

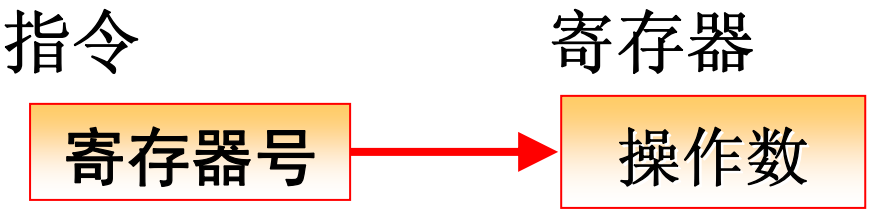
3）立即数只能是源操作数，不能作为目的操作数；

4）立即寻址的操作数就存在指令序列中，CPU不需要访问内存就可以获得操作数，指令执行速度快。

此外，以A～F开头的数字出现在指令中，前面一定要加数字0，以区别其它符号。

2、寄存器寻址（Register addressing）

- 操作数存放在某个寄存器中，指令指定寄存器号。



- 寄存器寻址示例

```
MOV AH, BL      ；(BL)→AH，即(AH)＝(BL)
MOV  DS, AX     ；(AX)→DS，即(DS)＝(AX)
MOV  SI, AX     ；(AX)→SI ，即(SI)＝(AX)
MOV  ECX, EDX   ；(EDX)→ECX，即(ECX)＝(EDX)
```

- 寄存器寻址方式也不需访问存储器即可 得到操作数，指令执行速度快。

二、与存储器有关的寻址方式

- 有效地址的计算：

  EA＝基址 +（变址×比例因子）+ 位移量

- 有效地址(EA)的4种组成成分

  1）位移量（displacement）

  - 存放在指令中的8位、16位或32位的数字，是一个地址。

  2）基址（base）

  - 存放在基址寄存器中的内容，用于指向数组的首地址。

  3）变址（index）

  - 存放在变址寄存器中的内容，用于访问数组的某个元素。

  4）比例因子（scale factor）

  - 其值可为1，2，4或8，386及其后继机型新增加的。

16/32位寻址时有效地址四种成分的组成

四种成分	16位寻址	32位寻址
位移量	0, 8, 16位	0, 8, 32位
基址寄存器	BX, BP	任何32位通用寄存器(包括ESP)
变址寄存器	SI, DI	除ESP以外的32位通用寄存器
比例因子	无	1, 2, 4, 8

默认段选择规则

访存类型	所用段及寄存器	缺省选择规则
指令	代码段 CS	用于取指
堆栈	堆栈段 SS	所有的堆栈的进栈和出栈任何用ESP或EBP作为基址寄存器的访存
目的串	附加段 ES	串处理指令的目的串
局部数据	数据段 DS	除相对于堆栈以及串处理的目的串以外的所有数据访问

访问非默认段数据的方法——段超越

- 数据存放较灵活，除了放置在默认的DS段外，还可放在其它段，对其访问时需使用段超越前缀，可用的段超越前缀有CS:、DS:、ES:、SS:、FS:和GS:。
- 段超越举例：
  - MOV AX, [10H] ；默认DS段10H处的一个字赋给AX
  - MOV AX, ES:[10H] ；ES段10H处的一个字的数据赋给AX
- 不允许使用段超越前缀的情况

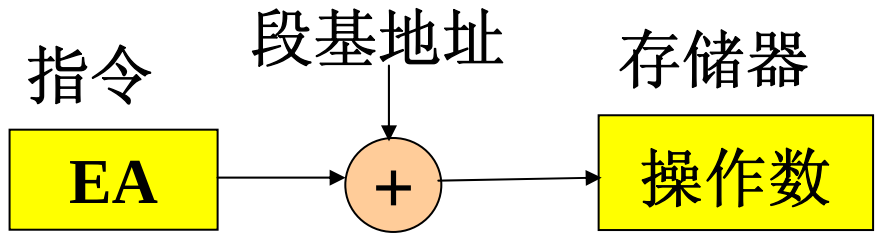
（1）串操作指令的目的串必须用ES段

（2）PUSH指令的目的和POP指令的源必须用SS段

（3）程序的指令必须存放在CS段

3、直接寻址（Direct addressing）

- 操作数在存储器中，而存放操作数的内存单元地址的偏移量（有效地址）在指令中。
- **EA＝基址＋（变址×比例因子）＋ 位移量**



- 例如：**MOV　AX, [2000H]**
  - 对比：　**MOV　AX, 2000H**　　；立即寻址
- 指令中的有效地址必须加方括号，以便与立即数相区别。
- 直接寻址方式适用于处理单个变量。
- 直接寻址方式隐含段寄存器是 **DS**，但允许段跨越。用段超越前缀说明所使用的段。

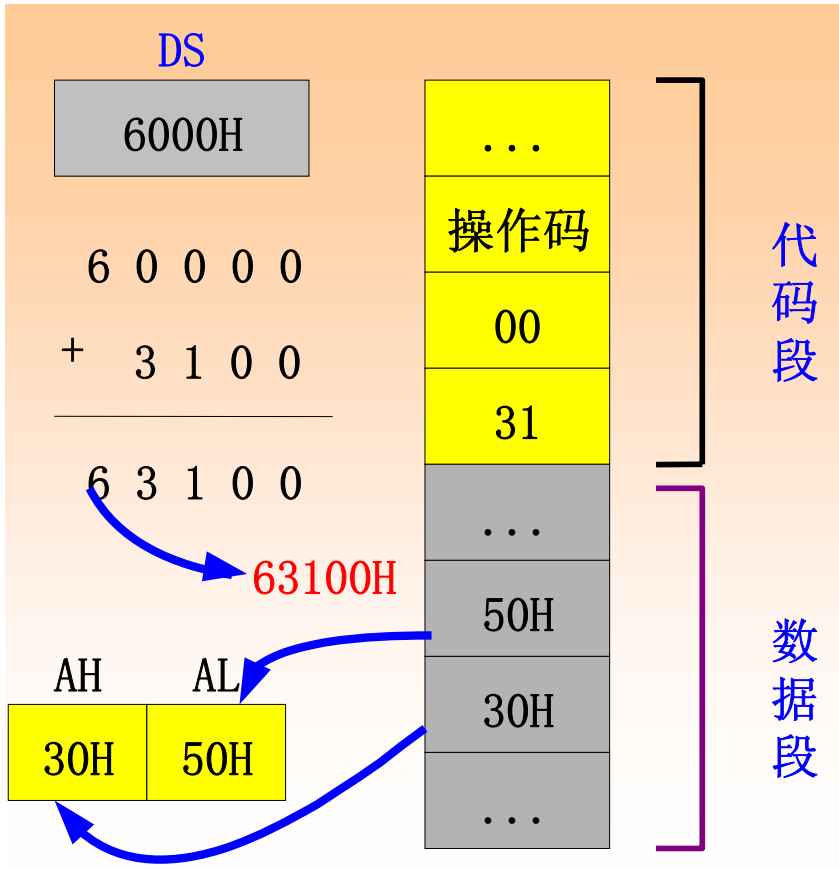
例1：MOV AX，[3100H]

假设:（DS）= 6000H，

（63100H）= 3050H 。

则指令执行后:

（AX）=3050H



若操作数不在数据段，则应指定段超越。如，假设操作数在附加段，则指令应为:

MOV AX，ES:[3100H]

“:”称为修改属性运算符号。

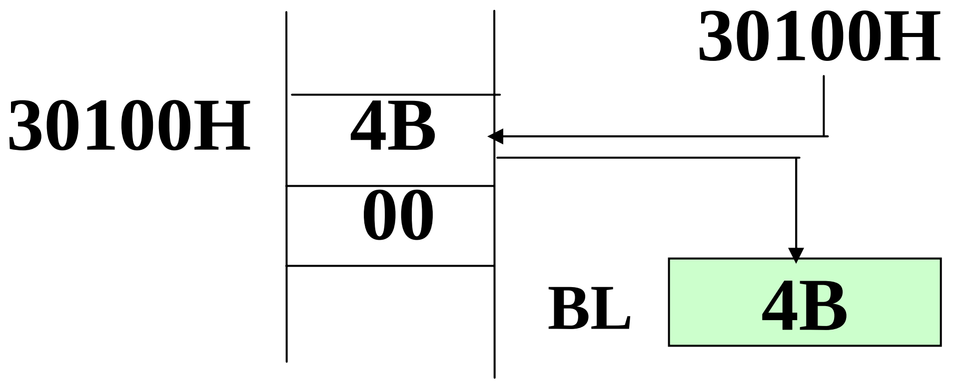
直接寻址方式

例2：假设(ES)=3000H，物理地址30100H的单元内容为004BH，以下指令执行后，BL寄存器的内容＝？

MOV BL, ES:[0100H]

物理地址

```
  30000H
+  0100H
---------
  30100H
```



在汇编语言指令中，可以用符号地址代替数值地址

如：　MOV AX, BUFF　或者MOV AX, [BUFF]

　　　MOV AX, AREA1　或者MOV AX, [AREA1]

其中：BUFF和AREA1为存放数据单元的符号地址。

- 说明：
  - 在汇编语言中，类似**BUFF**和**AREA1**这样的符号不仅可以代表指令中的符号地址，也可以表示一个立即数。程序中必须事先安排说明语句（伪指令）进行说明。
  - 伪指令——不属于**CPU**的指令集，没有对应的机器代码，只是用于在汇编语言指令的编译过程中，完成变量定义、存储分配、段定义等。

例**3-9**：

```
AREA1  EQU  0867H;
... ...
MOV   AX，AREA1
```

  - **EQU是赋值伪指令**，此后**AREA1**就代表一个立即数**0867H**；**MOV AX，AREA1**执行后，寄存器**AX**的内容为**0867H**。

例3-10：AREA1 DW 0867H；

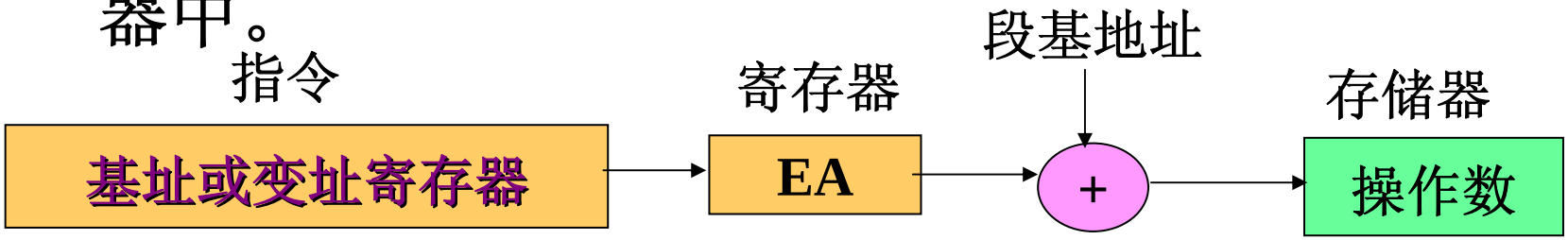
… …

MOV AX，AREA1

- DW是定义字变量伪指令语句，此后AREA1就是一个字变量的名称，变量在程序是可以修改的运算对象，是内存中的一个存储区域，AREA1就是该存储区域的符号地址，该地址存储了一个字0867H。
- MOV AX，AREA1执行后，CPU从逻辑地址为DS:AREA1的单元中取出一个字装入AX寄存器，AX的内容也是0867H。
- 注意：例3-9与例3-10的区别。

4、寄存器间接寻址方式（Reg. indirect addr.）

- 操作数在存储器中，操作数地址的偏移量在寄存器中。



- EA＝基址＋（变址×比例因子）＋ 位移量

1) 若指令中指定BX、SI或DI寄存器作为基址寄存器进行间接寻址，操作数一般在现行数据段区域中，应该使用(DS)作为段地址。即操作数物理地址（PA）

PA＝16 ×（DS）+（BX）

或 ＝16 ×（DS）+（SI）

或 ＝16 ×（DS）+（DI）

例：MOV AX，[DI]

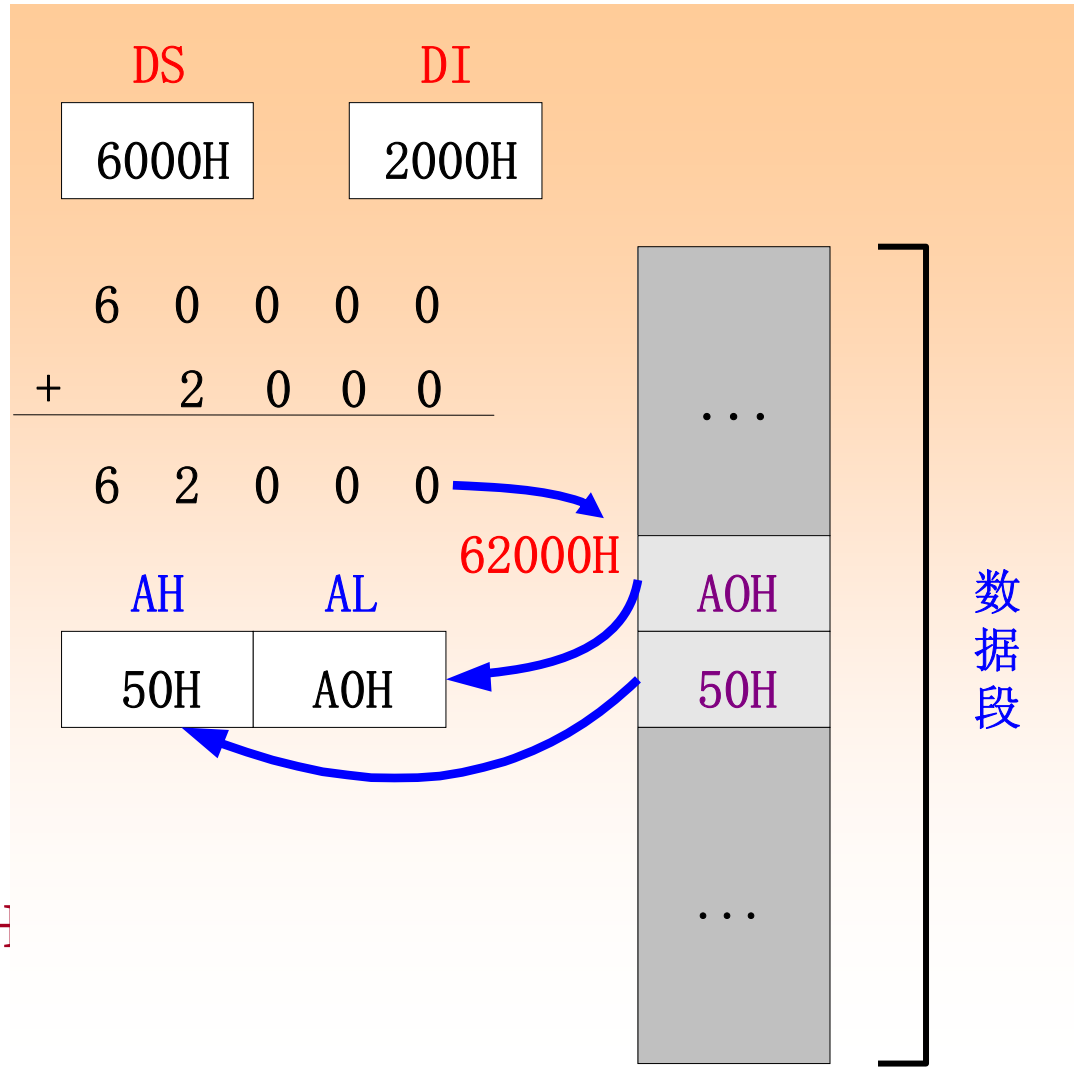
假设：(DS)=6000H

(DI)=2000H

则：PA=62000H

假设：(62000H)=50A0H

指令执行后，(AX)=50A0H



注意：寄存器名称必须加方括号，以便与寄存器寻址相区别

寄存器间接寻址方式

MOV AX，[DI]

2）若选择BP寄存器作为间接寻址

操作数在堆栈段区域中，用SS寄存器的内容作为段地址。

操作数物理地址：

PA=16×（SS）+（BP）

例：　MOV　[BP]，AX

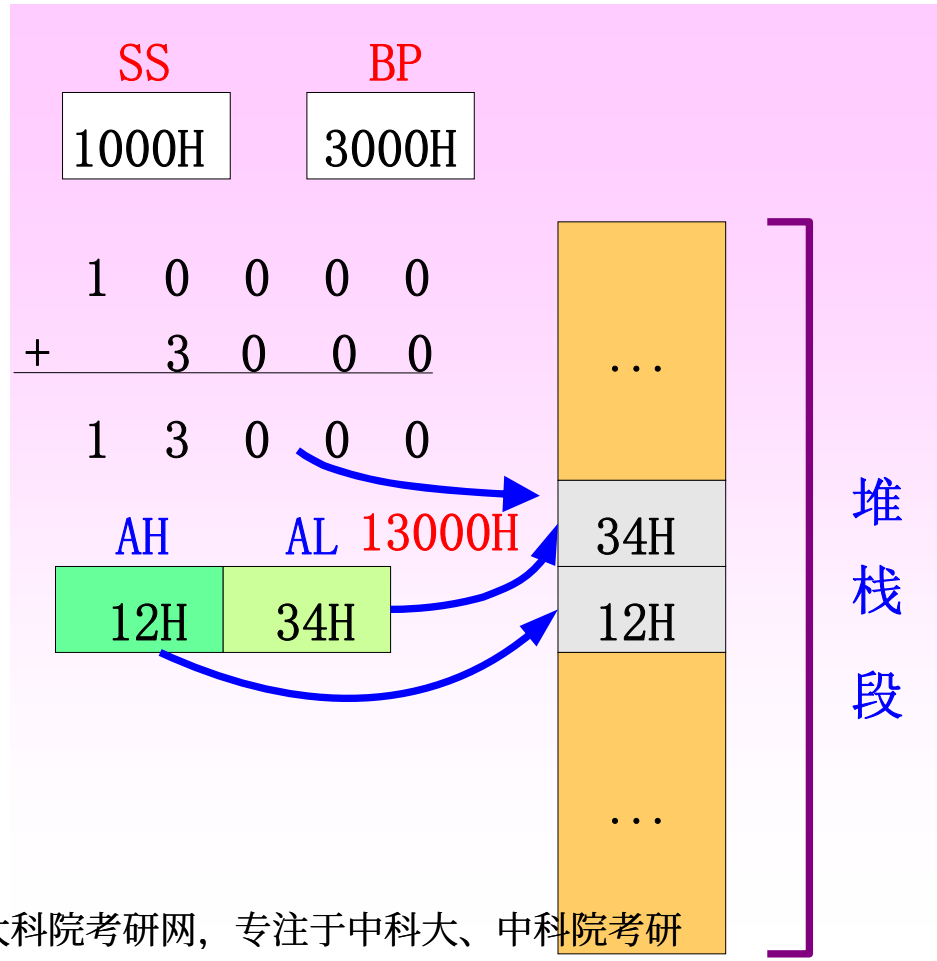
执行前：　(SS)=1000H，

(BP)=3000H，

(AX)=1234H

执行后：PA=13000H

(13000H)=1234H



3）用 SI、DI、BX 、BP作为间接寻址允许段跨越

指令中可以指定段跨越前缀来取得其它段中的数据。

例：　MOV　ES:[DI], AX；操作数在附加段ES中，
物理地址：PA = 16 ×（ES）+（DI）

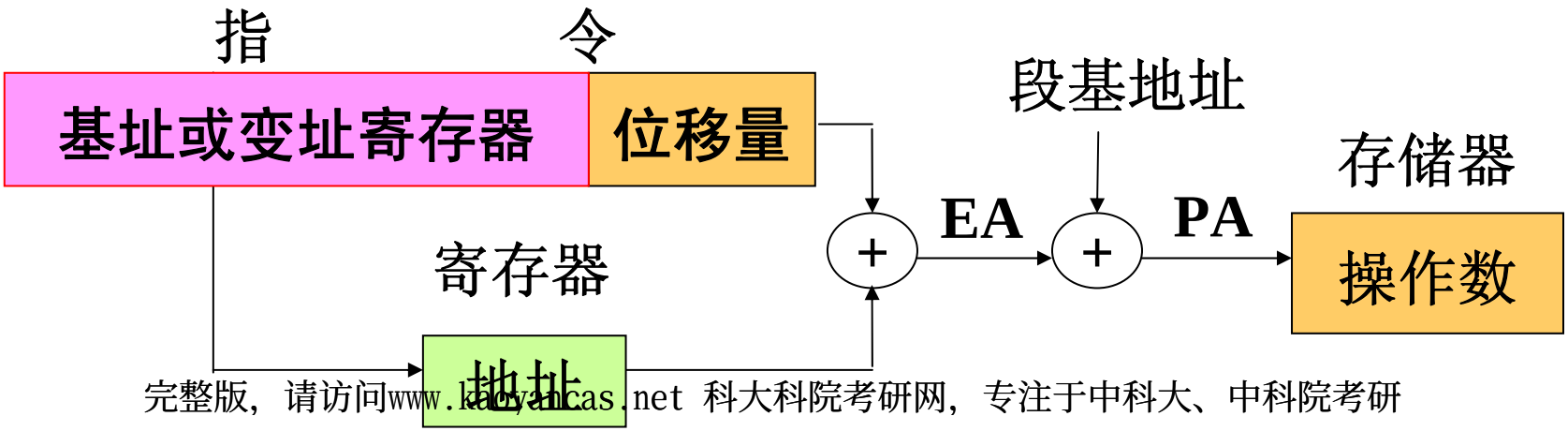
又例：MOV DX, DS:[BP];

指令中若指定BP间接寻址，默认操作数在堆栈段。但是这条指令用段超越前缀从默认段以外的DS段中取得数据，物理地址PA=16 ×（DS）+（BP）

这种寻址方法可以用于表格处理。

5、寄存器相对寻址方式（Reg. relative addr.）或变址寻址 （Index Addressing）

- 与寄存器间接寻址类似，只不过操作数的有效地址是基址（或变址）寄存器的内容再加上指令中指定的位移量。
- EA＝基址 +（变址×比例因子）+ 位移量
- 指令中指定的基址寄存器为BX、SI或DI时，默认的段寄存器是DS；若指定的寄存器是BP时，默认的段寄存器为SS。
- 同样，该方式可以使用段超越。



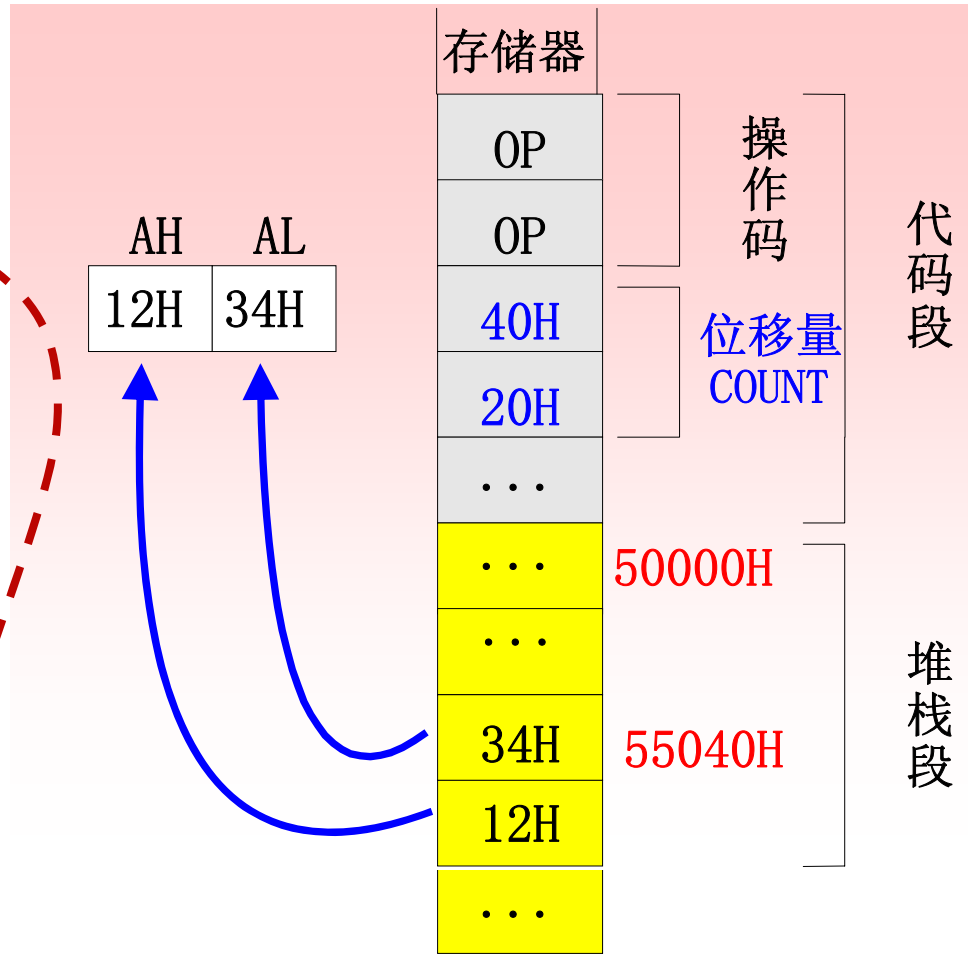
例：以下三条指令完全等价

MOV AX, COUNT [BP]
或MOV AX, [COUNT+BP]
或MOV AX, COUNT+[BP]

COUNT为符号地址，本例中为16位的偏移量。

假设指令执行前：
(SS)=5000H， (BP)=3000H，
COUNT=2040H，
(AX)=4321H

则指令执行后：
EA=5040H， PA=55040H
(55040H)=1234H
(AX)=1234H



- 用途：这种寻址方式适用于表格处理。
  - 表格首地址COUNT，修改基址或变址寄存器对表格中的表项进行访问。
- 例：某8位数据表的首地址为COUNT，欲将表中第10个数据读取到AL中。
  - 第10个数据的有效地址：EA= COUNT + 9

    MOV SI , 09H

    MOV AL , [SI+COUNT]
  - 要读取表格另外的数据，改变SI的内容即可。

6、基址加变址寻址方式（Based indexed addressing）

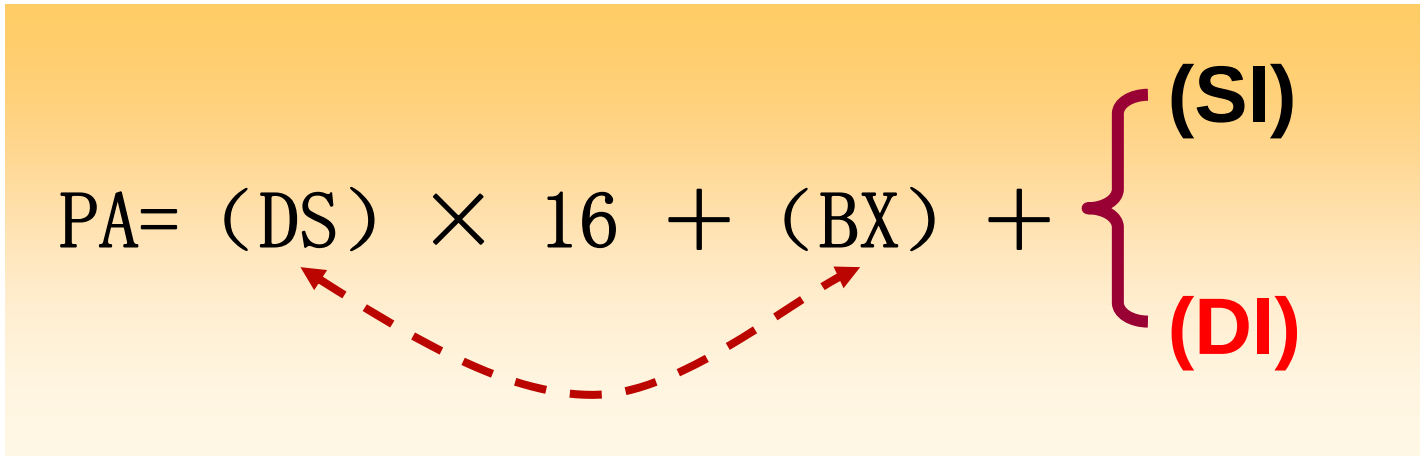
- 操作数的有效地址是一基址寄存器和一变址寄存器的内容之和，基址寄存器和变址寄存器均由指令指定。
- **EA＝基址 +（变址×比例因子）+ 位移量**
- 适用于16位和32位寻址。例如：

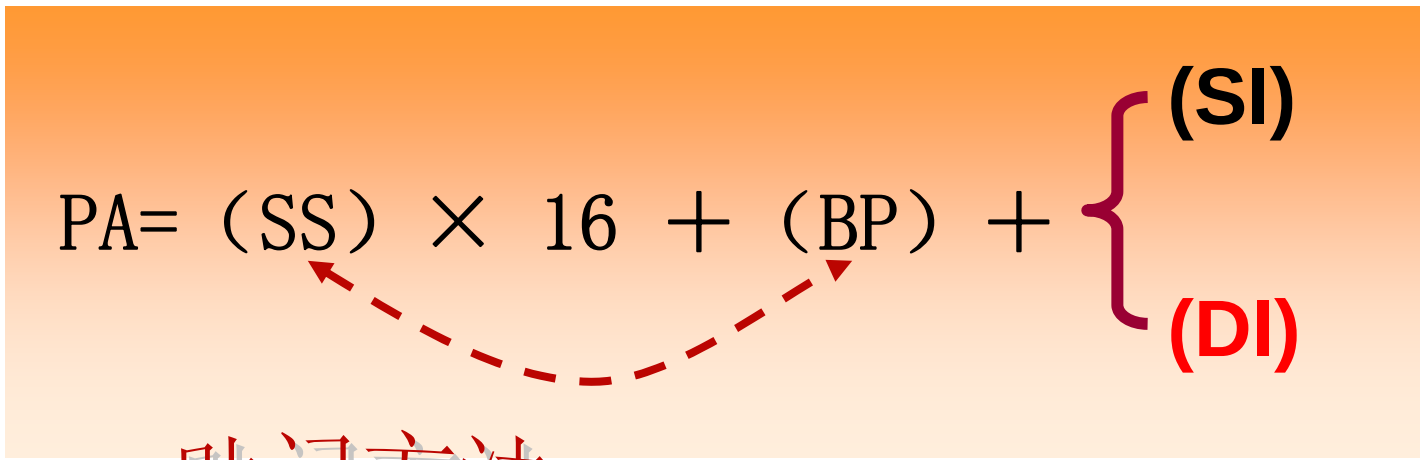
MOV AX, [BX+SI]　　MOV EAX, [EDX+EBP]

$$EA = (BX) + \begin{cases} (SI) \\ (DI) \end{cases}$$

$$EA = (BP) + \begin{cases} (SI) \\ (DI) \end{cases}$$

- 除有段跨越前缀之外，形成物理地址有二种方式：





助记方法：DX vs SP

例：以下两条指令等价

MOV AX, [BX][SI]

或 **MOV AX, [BX+SI]**

假设执行指令前：

(DS)=3200H,

(BX)=0456H,

(SI) =1094H

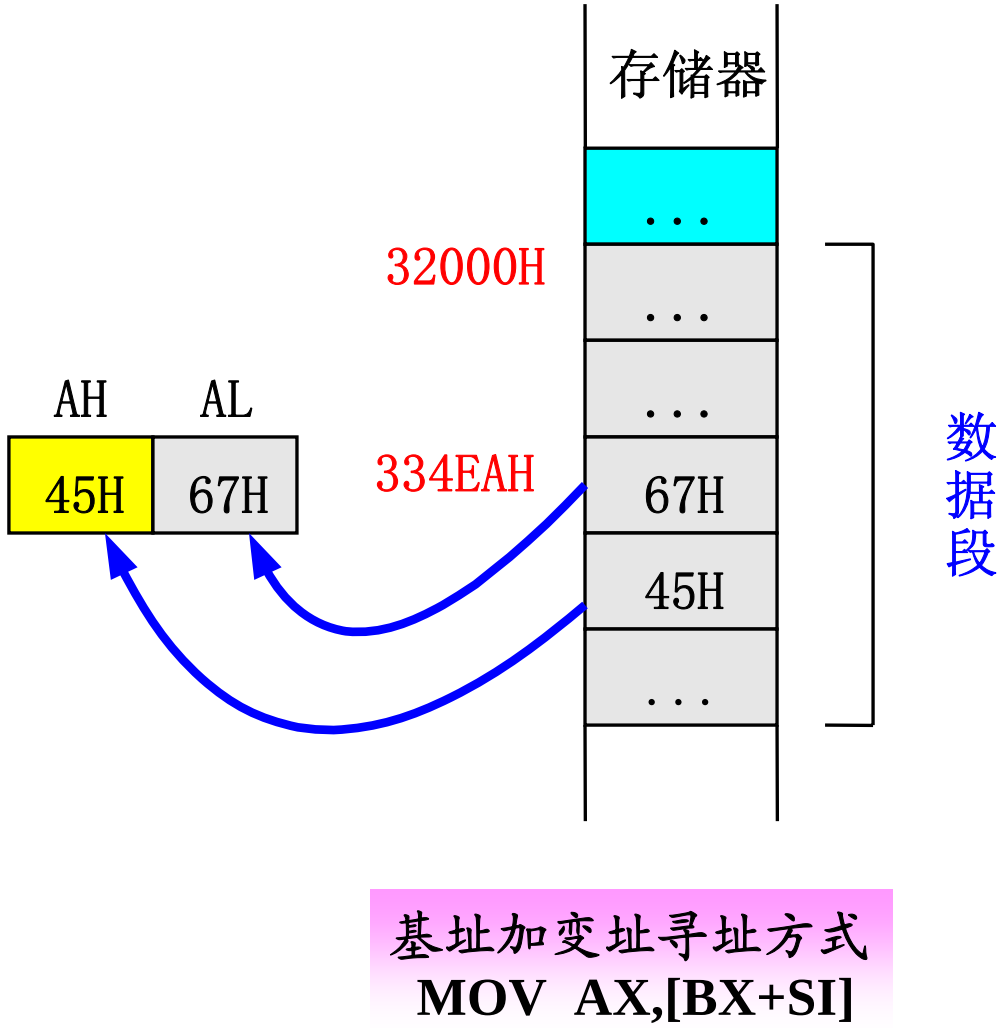
(334EAH)=4567H

执行指令后：

EA=14EAH

PA=334EAH

(AX)=4567H



基址加变址寻址方式

MOV AX,[BX+SI]

- 用途：
  - 这种寻址方式适用于二维数组操作和二重循环。
    - 访问二维数组：表格首地址放在基址寄存器中，用变址寄存器来访问数组中的元素。
    - 二重循环：基址寄存器用于外循环计数，变址寄存器用于内循环计数。
  - 二个寄存器都能修改，所以比直接变址方式更加灵活。

7、相对基址加变址寻址方式（Relative based indexed addressing）

- 操作数的有效地址是一个基址寄存器的内容、一个变址寄存器的内容再加上偏移量之和。即：

  EA＝基址 +（变址×比例因子）+ 位移量

  例如：**MOV AX, [BX+SI+50]**

$$EA = (BX) + \begin{cases} (SI) \\ (DI) \end{cases} + \begin{cases} 8\text{位偏移量} \\ 16\text{位偏移量} \end{cases}$$

$$EA = (BP) + \begin{cases} (SI) \\ (DI) \end{cases} + \begin{cases} 8\text{位偏移量} \\ 16\text{位偏移量} \end{cases}$$

- 除有段跨越前缀之外，形成物理地址有二种方式：

$$PA=(DS)\times 16+(BX)+\begin{cases}(SI)\\(DI)\end{cases}+\begin{cases}\text{8位偏移量}\\\text{16位偏移量}\end{cases}$$

$$PA=(SS)\times 16+(BP)+\begin{cases}(SI)\\(DI)\end{cases}+\begin{cases}\text{8位偏移量}\\\text{16位偏移量}\end{cases}$$

例：以下三条指令等价

MOV AX, MASK[BX][DI]

MOV AX, MASK [BX+DI]

MOV AX,[MASX+BX+DI]

假设执行指令前：

(DS)=3000H，(BX)=1346H

(DI)=0500H，MASK=1234H

(32A7AH)=4050H

执行指令后：

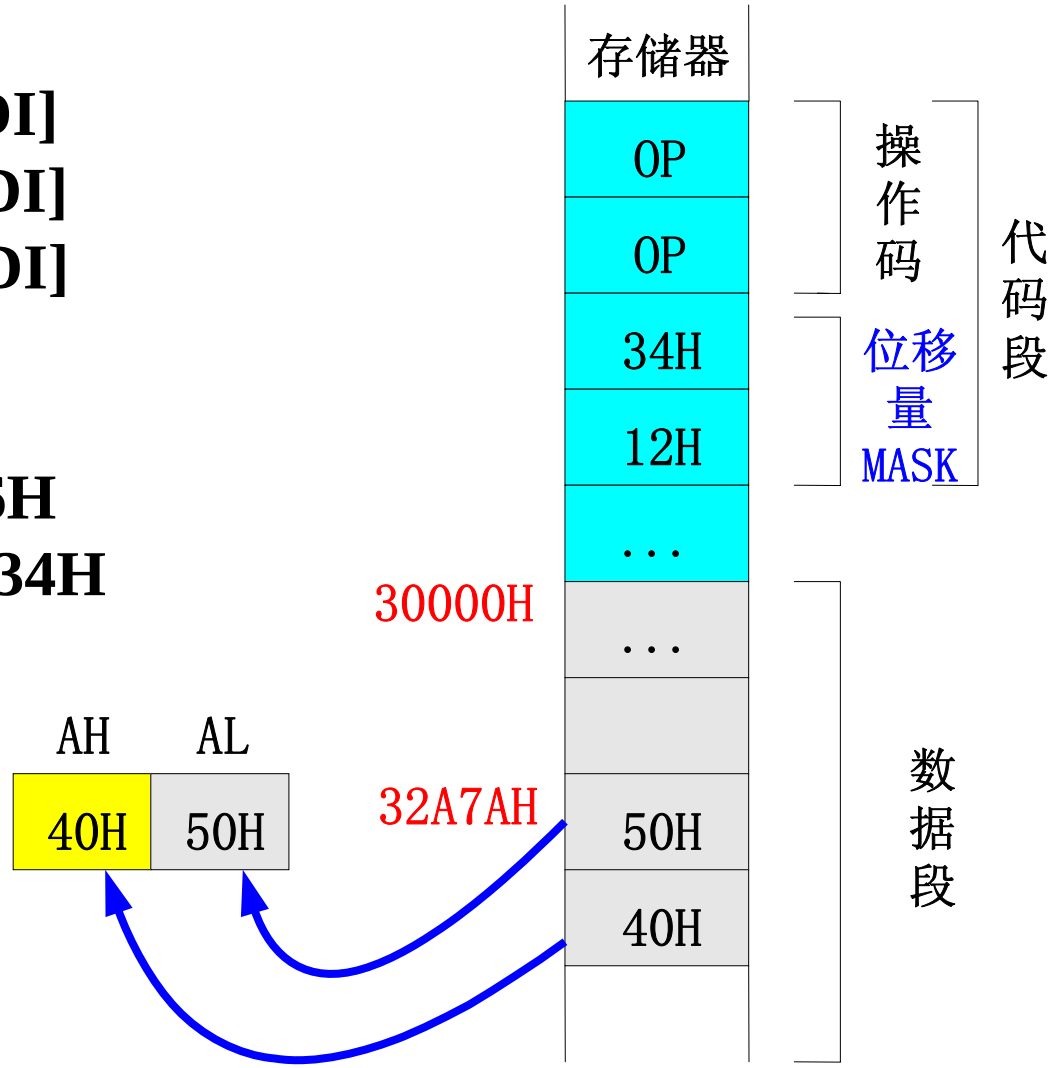
EA = MASK + BX + DI

= 2A7AH

PA = 16 × DS + EA

= 32A7AH

(AX)=4050H



相对基址加变址

MOV AX,MASK[BX+DI]

- 用途：
  - 主要用于二维数组操作，位移量为数组起始地址，适用于**16**位和**32**位寻址。
  - 另外这种寻址方式也为堆栈处理提供方便，主程序常常利用堆栈与子程序传递参数。
    - **(BP)→** 栈顶（一般 **BP**可指向栈顶）
    - 从栈顶到数组的首地址可以用位移量（**MASK**）表示。
    - 变址寄存器（**SI**）或（**DI**）——指向数组中某个元素。

8、比例变址寻址

- 80386以后CPU增加的寻址方式，只适用于32位寻址。
- 操作数的有效地址是变址寄存器的内容乘以指令中指定的比例因子及位移量之和，即

  EA＝基址 + （变址×比例因子） + 位移量

  例：MOV EAX, [ESI*4+50]

- 比例因子可以是1、2、4或8，隐含比例因子是1，位移量为8位或32位立即数。
- 适用于一维数组操作，当数组元素大小为2/4/8字节时，它更方便、有效。

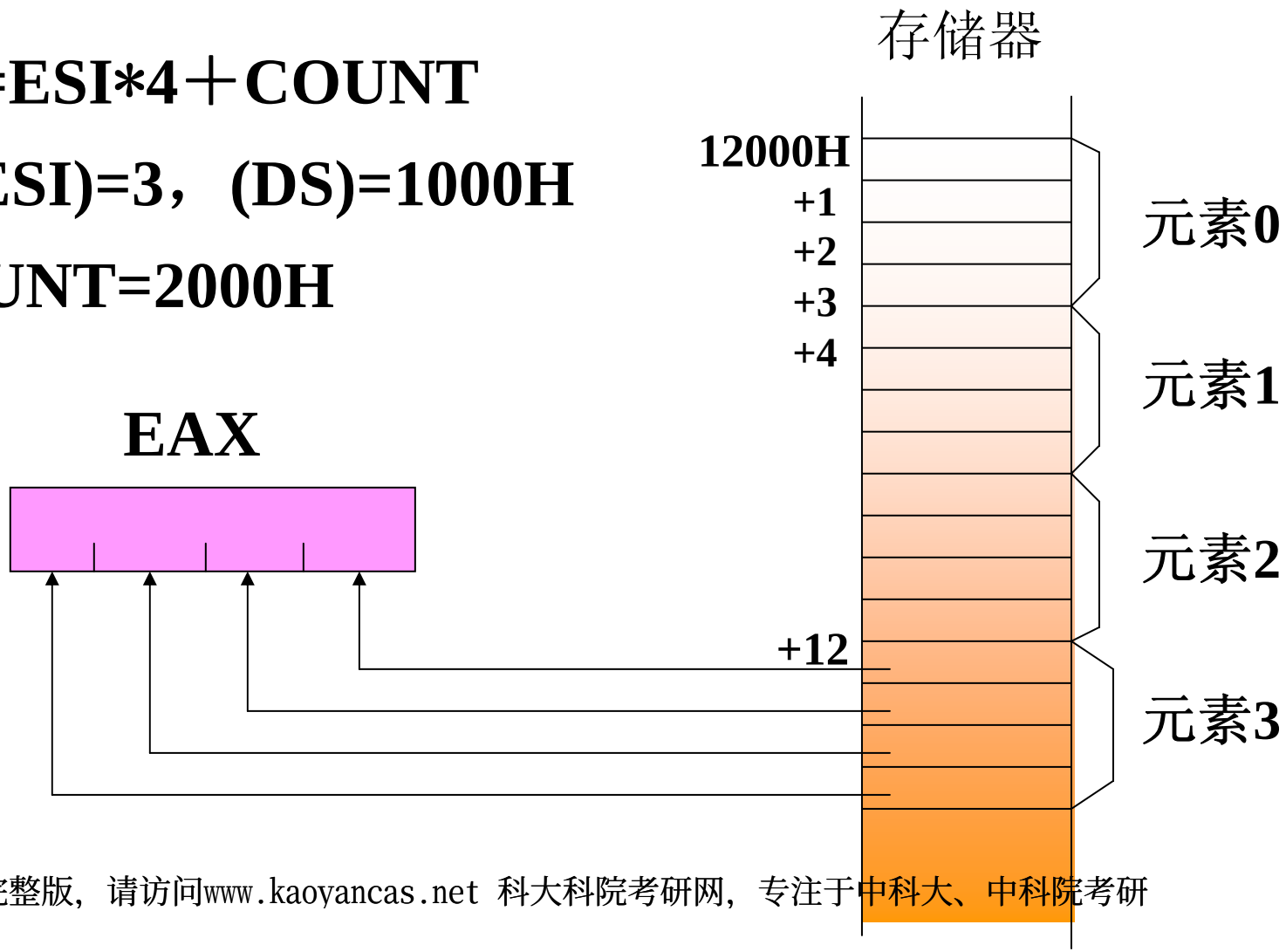
比例变址寻址示例

MOV EAX, COUNT[ESI*4]

EA=ESI*4＋COUNT

设(ESI)=3，(DS)=1000H

COUNT=2000H



9、基址比例变址寻址

- 80386以后CPU增加的寻址方式，只适用于32位寻址。
- 操作数有效地址是变址寄存器的内容乘以指令中指定的比例因子加基址寄存器内容之和，即：

  EA＝基址 +（变址×比例因子）+ 位移量

  如： MOV EAX, [EDX*8+EAX]

  MOV ECX, [EBX+ESI*8]

- 适用于数组元素大小为2/4/8字节时的二维数组操作。

10、相对基址比例变址寻址

- 80386以后CPU增加的寻址方式，只适用于32位寻址。
- 操作数的有效地址是变址寄存器的内容乘以指令中指定的比例因子加基址寄存器的内容、再加上位移量之和，即：

  EA＝基址＋（变址×比例因子）＋ 位移量

  例如：　　MOV EAX, TABLE[EBX+EDI*4]

- 适用于数组元素大小为2/4/8字节时二维数组操作，位移量为数组起始地址

三、其它寻址方式

1）隐含寻址

- 指令不指明操作数，但是有隐含规定。如指令DAA，指定对AL中的数据进行十进制调整，结果仍然保留在AL中。

2）I/O端口寻址（只有两种方式）

1)直接端口寻址。端口地址在指令中给出，是一个8位立即数。能访问的端口号为00～0FFH；

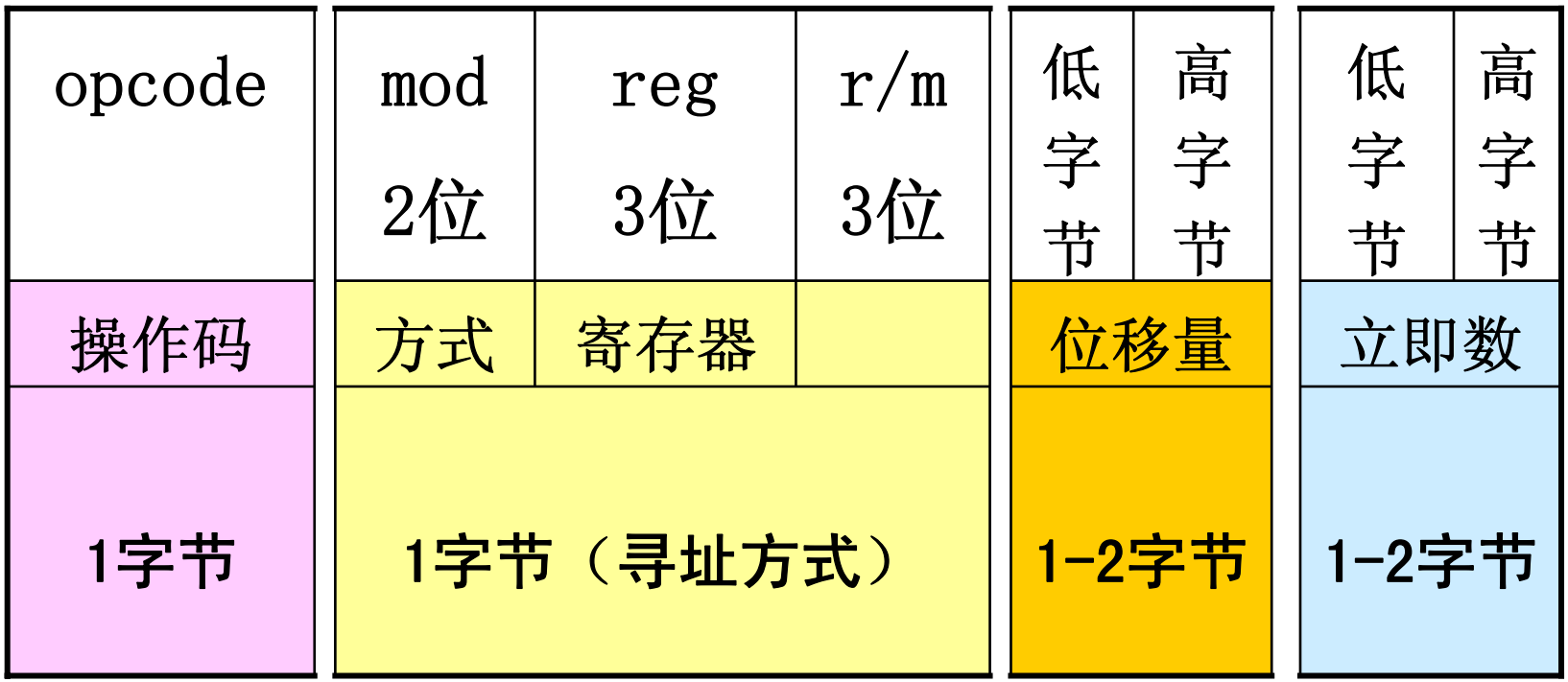
2)间接端口寻址。IO端口地址由DX寄存器提供，能访问的端口地址范围为0000～0FFFFH。

3-2 8086指令的机器码表示方法（#）

一、8086机器语言指令的特点

- 结构：8086指令由操作码域和操作数（地址码）域组成。
- 指令长度：8086的指令长度是可变的，一条指令一般由1－6个字节组成（加上前缀字节，最长可为7字节）。
- 指令机器码的设计思路：对每种基本指令类型设计了一个编码格式，其中规定了各相关含义，对照格式的定义填上不同的寻址方式、数据类型等就得到指令机器码。

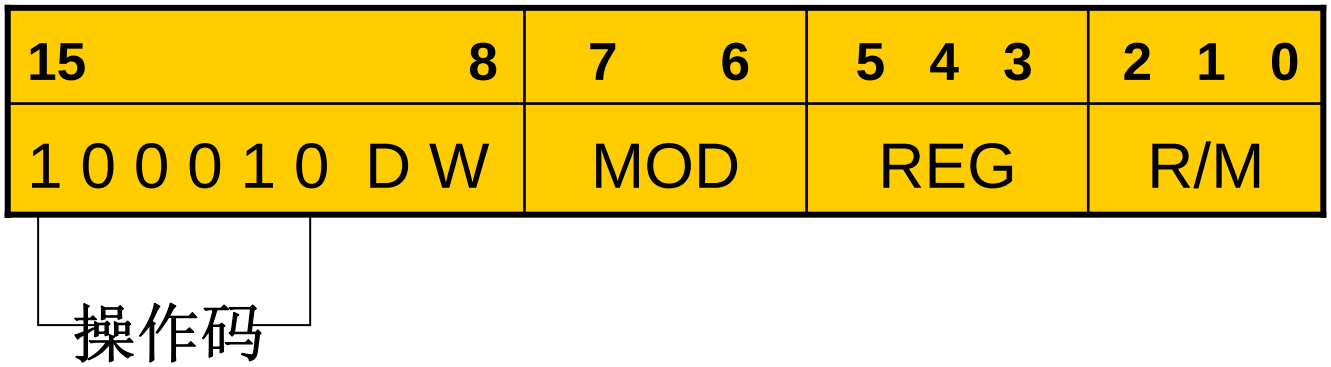
8086 CPU指令格式



二　机器语言指令代码的编制

1、编码格式说明（以数据传送指令为例）：

MOV指令格式：MOV　reg/m，reg/m



注：80x86指令系统中凡是双操作数的指令，两个操作数不能全为存储器，必须有一个是寄存器。

- 操作数1：R(register)
  - REG — R(reg)，参见教材P67表3-1。
  - W — 字长。W=1为字，W=0为字节
  - D —传送方向。D=1，→R；D=0，R→

- 操作数2：R或M(memory)
  - MOD和R/W编码确定寻址方式，由寻址方式明确R和有无位移量，参见教材P67表3-2。

15 8	7 6	5 4 3	2 1 0
1 0 0 0 1 0 D W	MOD	REG	R/M

教材P67表3-1：REG字段对应的寄存器编码

表 3-1　8086 寄存器编码表

REG	W＝1(字)	W＝0(字节)
000	AX	AL
011	BX	BL
001	CX	CL
010	DX	DL
100	SP	AH
111	DI	BH
101	BP	CH
110	SI	DH

REG	段寄存器
01	CS
11	DS
00	ES
10	SS

对于使用段寄存器的指令，**REG**字段只有两位。上右侧是表是**8086CPU**对应的段寄存器编码。

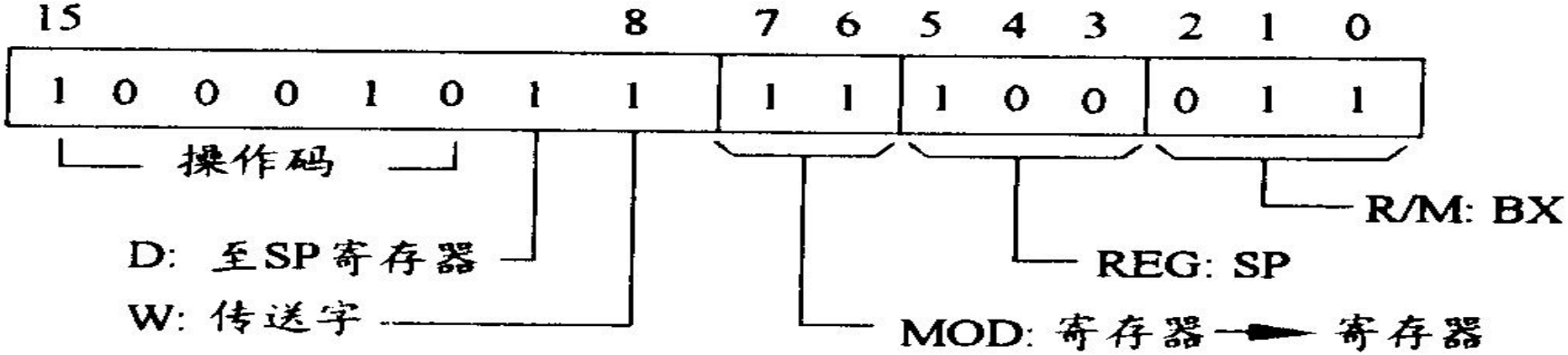
教材P67表3-2。寻址方式以及寄存器和有无位移量

表 3-2　MOD 和 R/M 的编码

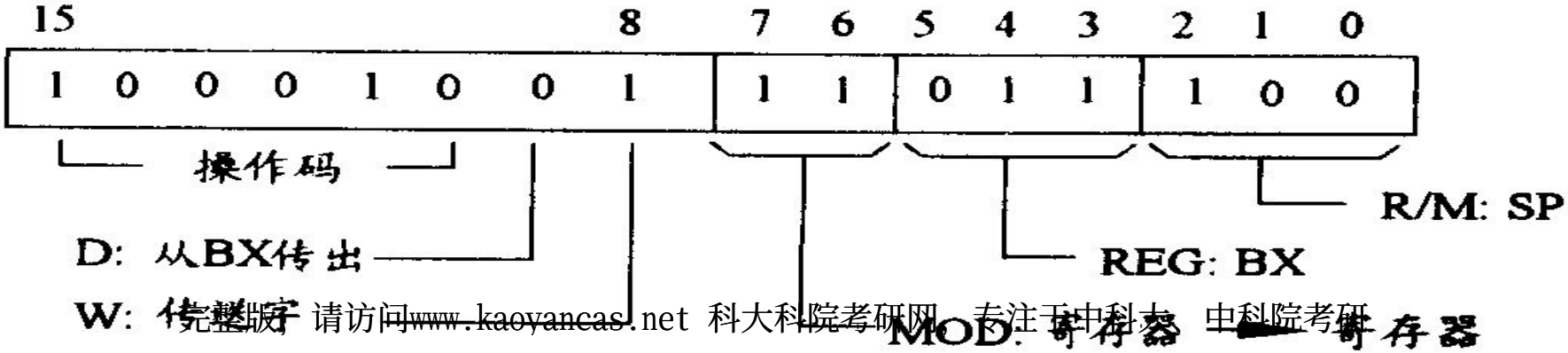
MOD / R/M	00	01	10	11	
				W=0	W=1
000	[BX]+[SI]	[BX]+[SI]+D8	[BX]+[SI]+D16	AL	AX
001	[BX]+[DI]	[BX]+[DI]+D8	[BX]+[DI]+D16	CL	CX
010	[BP]+[SI]	[BP]+[SI]+D8	[BP]+[SI]+D16	DL	DX
011	[BP]+[DI]	[BP]+[DI]+D8	[BP]+[DI]+D16	BL	BX
100	[SI]	[SI]+D8	[SI]+D16	AH	SP
101	[DI]	[DI]+D8	[DI]+D16	CH	BP
110	D16(直接地址)	[BP]+D8	[BP]+D16	DH	SI
111	[BX]	[BX]+D8	[BX]+D16	BH	DI

A）寄存器间传送指令编码示例

例3-18 求指令 MOV SP，BX 的机器码。

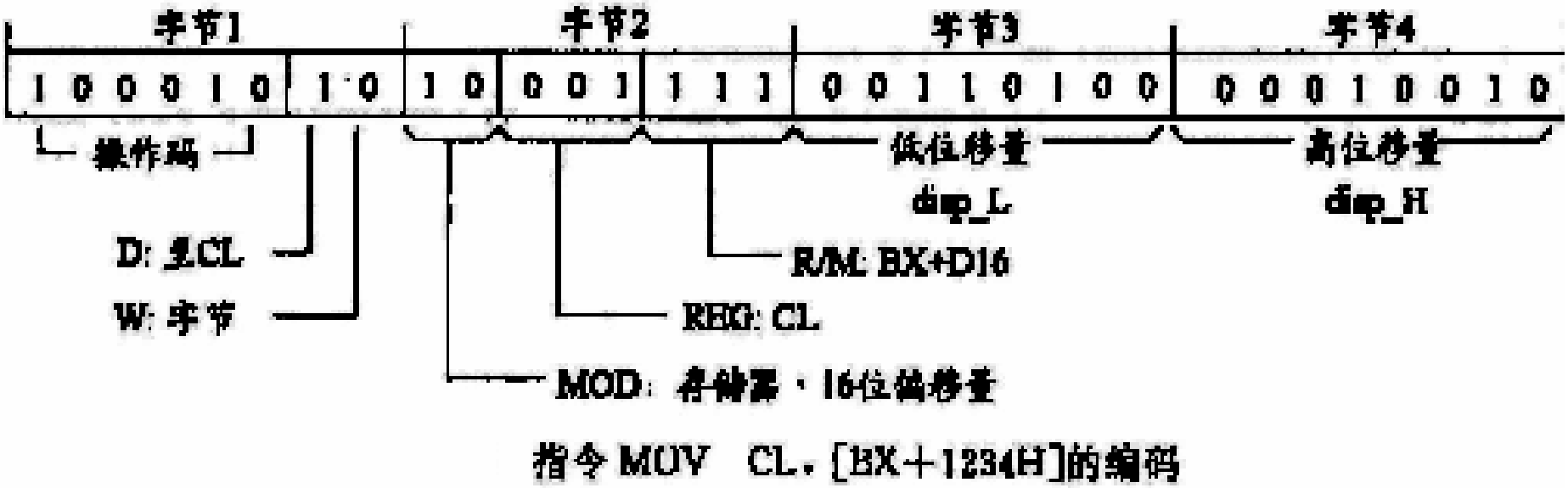


另一种形式



B）寄存器与存储器间传送指令编码示例

例3-19：求指令MOV CL，[BX+1234H]的机器码



指令 MOV CL，[BX+1234H]的编码

3-3 8086指令系统

——IA-32体系结构中的通用指令

- 指令系统：指令系统是一台计算机(μP)能识别和执行的全部指令的集合。
- 每种μP都有独立的指令系统。8086共有6大类87条指令，这些指令系统均向上兼容。

指　令	基本条数	共　计
1）数据传送	14	87
2）算术运算	20	
3）逻辑运算	13	
4）串操作	5	
5）控制转移	28	
6）处理器控制	7	

一、数据传送指令

- 数据传送指令又可以分成4种：

  1）通用数据传送

  2）累加器专用传送（输入/输出数据传送）

  3）目的地址传送

  4）标志寄存器转送

  4种指令总共有14条，见下表… …

数据传送指令（14条）

通用数据传送指令（5条）
MOV、PUSH、POP、XCHG、XLAT
输入输出指令（2条）
IN、OUT
地址目标传送指令（3条）
LEA、LDS、LES
标志传送指令（4条）
LAHF、SAHF、PUSHF、POPF

（一）通用数据传送

1. MOV传送指令

格式：MOV　DST(目的)，SRC(源)

操作：　DST ← SRC

例：　MOV　DS, AX　　;DS ←AX

　　MOV　[DX], AX　　;[DX]←AX

　　MOV　AX, [BX+0060H]

注意：

1、源和目的操作数不能都是存储器

2、目的操作数不能是立即数，也不能是CS寄存器

3、不允许两个段寄存器之间传送数据

4、立即数不能直接送段寄存器

5、不影响标志位

MOV指令允许传送数据的途径（P71）

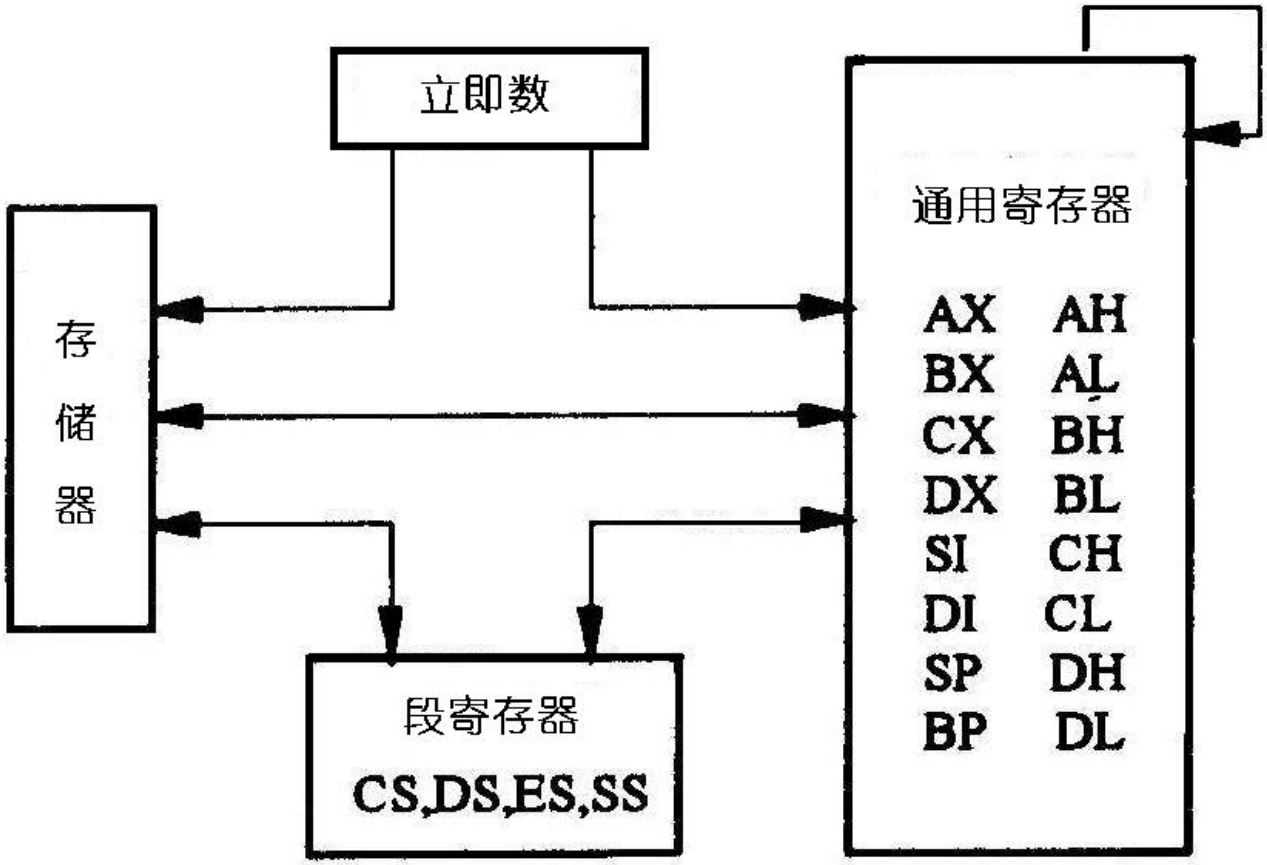


图 3-12 MOV 指令允许传送数据的途径

例：以下各条指令是否正确？

1）MOV ES，1234H

X 立即数不能直接送入段寄存器！

2）MOV 1AH，CL

X 1AH是立即数，不能作为DST！

3）MOV CS，BX

X CS不能作为目的操作数！

4）MOV W1，［BX］

X 两个内存操作数之间不能传送！

5）MOV ES，DS

X 两个段寄存器之间不能传送！

6）MOV AX, BL

X 两个操作数的位数（类型）不相同！

2. 堆栈操作指令

- 格式：PUSH　SRC(源)；
- 操作过程：

  a、SP－1，指示堆栈存放数据位置，存源SRC高8位；

  b、SP再减1，存源SRC的低8位，完成压栈操作。

- 格式：POP　　DST(目的)；
- 操作过程：

  a、将SS:SP指示的栈顶处两个字节弹出到DST操作数中；

  b、SP＋2，指示当前栈顶位置，完成出栈操作。

- 注意：
  - 1、堆栈操作只能以字为单位进行。
  - 2、不能 **POP CS**（即**CS**不能作为**POP**的目的操作数）
  - 3、**SRC**不能是立即数

- 例：设**(SS)=2000H**，**(SP)=00C0H**，执行下述指令后，**SP**的值＝？ 物理地址**PA**＝？

1）PUSH AX ； SP＝SP－2＝00BEH，PA＝200BEH

2）PUSH BX ； SP＝00BE－2＝00BCH，PA＝200BCH

3）POP CX ； SP＝00BC＋2＝00BEH，PA=200BEH

3. 交换指令

- 格式: XCHG　DST, SRC
- 注意：1、DST 与 SRC不能同时为内存单元；
  2、操作数不能为立即数，也不能为段寄存器；
  3、不影响标志位。
- 例：读下列程序段，写出AX，BX的内容。

```
MOV     AX,1234H    ；(AX)＝1234H
MOV     BX,5678H    ；(BX)＝5678H
PUSH    AX          ；
PUCH    BX          ；
POP     AX          ；  (AX)＝？
POP     BX          ；  (BX)＝？
XCHG    AX，BX      ；  (AX)＝？      (BX)＝？
```

4. 表转换指令（换码）

- 格式：XLAT
- 操作：AL←[AL+BX]
- 步骤：

  1) 准备：表首地址 → BX，序号n（表中第n项）→AL

  2) 执行：XLAT；（BX+AL）→ AL

- 用途：

  实现不规则代码转换，如：

  字符扫描码 → ASCII码

  0~9数字 → LED显示器的七段点亮码

  数字 → 控制码

表转换指令应用举例：

利用查表转换方式，写出求5的平方的指令片断：

```
… …
MOV  BX , 2000H  ；  BX指向平方
                 ；  表的首地址
MOV  AL , 5      ；  求5的平方
XLAT             ；  执行换码指令，
                 ；  AL中是结果
```

地址	平方值
2000H	0
2001H	1
2002H	4
2003H	9
	…
	…
2009H	81

（二）输入输出(I/O)指令

1）直接寻址

- 格式:

IN AL，n	OUT n，AL	；传送一个字节
IN AX，N	OUT N，AX	；传送一个字

  - 寻址空间为：0 ~ 255

2）间接寻址

- 格式:

IN AL，DX	OUT DX，AL	；传送一个字节
IN AX，DX	OUT DX，AX	；传送一个字

  - 寻址空间为：0000H ~ FFFFH

（三）地址目标传送指令

1. 取有效地址

- 格式：**LEA　reg16, SRC**
- 功能：取**SRC**的有效地址（偏移量）送到目的寄存器（**reg16**）
- 说明：**LEA**的源操作数必须是存储单元；目的操作数是一个除了段寄存器以外的**16**位寄存器。
- 例：
  - **LEA　AX，[1000H]　；　AX＝1000H**
  - **LEA　SI，　DCD　　；　将DCD的偏移地址送到SI**
  - **LEA　BX，[BP+SI]　；　BX＝(BP)＋(SI)**

2. 传送（双字）指针到DS

- 格式： LDS DST，SRC
- 功能：从SRC指定的存储单元，取出4个字节（通常是2个字节的偏移量和2个字节的段址），前2个字节（低地址）送到DST指定的目的寄存器；后2个字节（高地址）送到DS寄存器。
- 说明：LDS的源操作数必须是存储单元；目的操作数必须是一个除了段寄存器以外的16位寄存器，常用SI寄存器。
- 例：

  DS＝1200H，(12450H)＝F346H，(12452H)＝0A90H

  执行LDS SI，[450H]后，SI＝F346H，DS＝0A90H

3. 传送（双字）指针到ES

- 格式： LES DST，SRC
- 功能：与LDS指令类似。不同之处是后2个字节（段址）送到ES寄存器；目的操作数常用DI寄存器。
- 例：

  DS＝0100H，BX＝0020H，(01020H)＝0300H，(01022H)＝0500H

  执行：LES DI，[BX]后，DI＝0300H，ES＝0500H

（四）标志传送指令

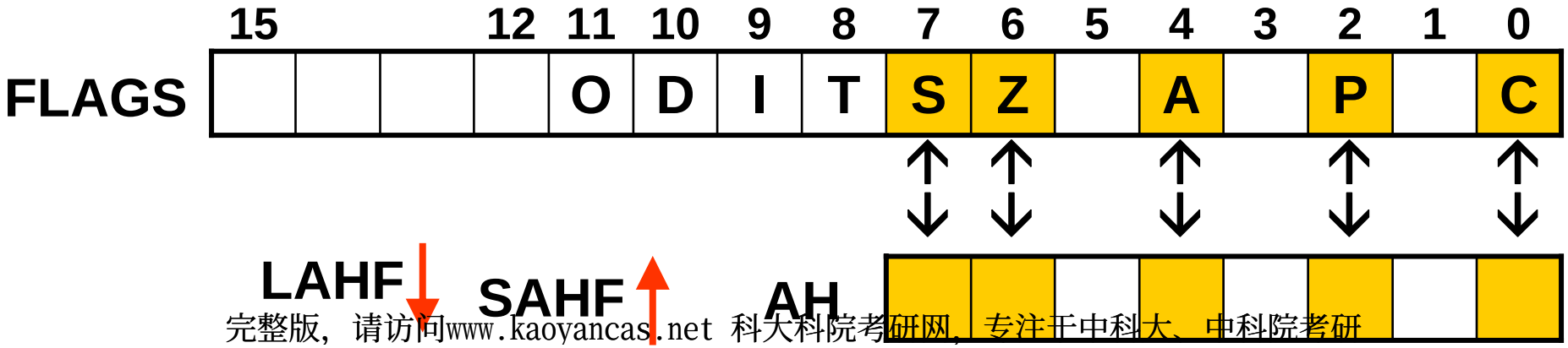
1. 读标志（标志寄存器送到AH）指令

- 格式：LAHF

2. 置标志（AH送到标志寄存器）指令

- 格式：SAHF

- LAHF与SAHF为互逆操作。如下图所示。



3.标志寄存器入栈指令

- 格式：**PUSHF**
- 功能：把标志推入堆栈。
- 步骤：1）**SP ← SP-2**，　2）**PSW(FLAGS)**入栈

4.标志寄存器出栈指令

- 格式：**POPF**
- 功能：从堆栈顶端中弹出标志字到FLAGS中
- 步骤：1）PSW（FLAG）←（[SP+1]、[SP]）
  2）SP ← SP＋2
- 说明：
  1）PUSHF和POPF要成对使用，以保护恢复PSW
  2）修改TF方法，先压栈PSW，再修改堆栈单元中的D8，再出栈。

习题一：P121～

- 1　2）、4）、6）、8）、10）
- 2　1）～6）
- 3　2）、4）、6）、8）、10）
- 5
- 6　1）～14）
- 7
- 8

二、算术运算指令

- 按加减乘除分为四类，每类包括数制调整指令：
  - 加法：**ADD、ADC、INC、AAA、DAA**
  - 减法：**SUB、SBB、DEC、NEG、CMP、AAS、DAS**
  - 乘法：**MUL、IMUL、AAM**
  - 除法：**DIV、IDIV、AAD、CDW、CWD**
- 算术运算指令可处理的数据类型：
  - 无符号二进制数
  - 带符号二进制数（补码）
  - 压缩**BCD**码（**1**字节表示**2**位十进制数）
  - 非压缩**BCD**码（**1**字节只含**1**位十进制数，高**4**位全**0**）

预备知识：关于溢出的讨论

一、回顾：FLAGS中的6个状态标志位（ACOPSZ）

- CF，进位标志。本次运算最高位有进位或借位发生，则CF=1。
- AF… …
- ZF… …
- … …
- OF，溢出标志。本次运算结果产生溢出（对于有符号数，8位表示范围为-128～+127，16位表示范围为-32768～+32767，超出此范围为溢出），或者运算过程中，最高位已无法再向前进位或借位，则OF=1。

二、以8位加法运算为例，观察CF和OF

1、无符号数，有符号数都不溢出

	无符号数	有符号数
04H	4	4
+ 0BH	+ 11	+ 11
0FH	15	15
	CF=0	OF=0

2、无符号数溢出，有符号数不溢出

	无符号数	有符号数
07H	7	7
+FBH	+251	+(-5)
1 02H	258	+2
	(>255)	
	CF=1	OF=0

3、有符号数溢出，无符号数不溢出

	无符号数	有符号数
09H	9	9
+7CH	+124	+124
85H	133	133
		(>127)
	CF=0	OF=1

4、无符号数，有符号数都溢出

	无符号数	有符号数
87H	135	-121
+F5H	+245	+(-11)
1 7CH	380	-132
	(>255)	(<-128)
	CF=1	OF=1

结论：OF——表示有符号数的溢出

CF——表示无符号数的溢出（进位）

（一）加法指令

1.不带进位位的加法

格式：ADD DST，SRC ；DST←DST+SRC

例：ADD AL，50H ；AL ←AL+50H

ADD [BX+DI]，AX

注意：影响标志AF、OF、PF、SF、ZF，CF（参见P80）。

2.带进位位的加法

格式：ADC DST，SRC ;DST←DST+SRC+**CF**

例：ADC AX，SI ;AX ←AX+SI+**CF**

ADC DX，[SI]

注意：影响标志AF、OF、PF、SF、ZF，CF（参见P80）。

- 说明：

  1）ADD和ADC的SRC和DST不能都是内存单元

  2）SRC和DST的数据类型必须一致，即都是字节或字。

3. 加一指令

- 格式：　INC　DST　；DST ← DST+1
- 例：　INC　AX　；AX ← AX+1

  INC　BL　；BL ← BL+1
- 该指令主要用于修改地址指针和循环计数器。
- 注意：影响AF、OF、PF、SF、ZF，但不影响CF。

关于BCD码加法运算以及十进制调整

CPU做加法运算时一律按二进制数规则加。因此两个BCD码相加时会出现两种情况：

① 和≤9

② 和>9 此时和的范围为0AH～12H（18）之间，其中，若和在0AH～0FH之间时，AF=0；若和在10H～12H之间时，AF=1或CF=1。

例：三个加法运算

0101	0101	1001
+ 0011	+ 0111	+ 1001
1000	1100	AF 0010
	②两个结果都是错误的	
①结果正确		

解决办法：十进制调整。

即遇到以上情况②时，对BCD结果进行加6调整。

	5＋7	9＋9
原先	0101	1001
	＋ 0111	＋ 1001
	1100	AF 0010
加6调整后	1100	AF 0010
	＋ 0110	＋ 0110
	10010	AF 1000

4.加法的ASCII调整指令

- 格式：AAA
- 功能：在用ADD或者ADC对两个非压缩BCD码或用ASCII码表示的十进制数作加法后，将AL中的和进行调整，结果放在AL或者AX中 。
- 策略：
  - 若AL低4位的值大于9或AF=1， 则AL＋6→AL，将AL高4位清零，同时置AF＝1、CF＝1、AH＋1→AH。
  - 否则仅将AL高4位清零。
- AAA指令影响标志AF、CF。AAA指令必须紧跟着ADD或ADC指令之后。

AAA指令示例1

计算十进制数9＋4

```
MOV AL, 9H          09H
MOV BL, 4H        +  04H
                  ------
ADD AL, BL          0DH      ；>9
AAA                +  06 H   ；调整
                  ------
                   13H
```

送入AH

结果：

(AH)=01H　(AL)=03H　CF=AF=1

AAA指令示例2：

- 计算两个ASCII码“9”和“5”的加法
- ASCII码‘9’和‘5’的编码分别是39H和35H，一般应先使用0FH分别与其相与（∧0FH），将ASCII码转换成非压缩的BCD码后再相加。利用AAA指令则可以先将两个ASCII直接相加，然后再对结果进行调整。

```
MOV AL, 39H        39H
MOV BL, 35H      + 35H
ADD AL, BL         6EH    ；低4位>9
AAA              + 06H    ；调整，低4位加6，高
        （AX）＝ 0104H    ；4位清零，置CF＝1
                          ；CF＝1，AF＝1
```

送入AH

- 结果调整：将AX的内容转换成ASCII码，只要使用“或”操作指令OR AX，3030H，AX的内容即为ASCII码‘1’‘4’。

5. 加法的十进制调整指令

- 格式：DAA
- 功能：在用ADD或者ADC对两个压缩BCD数作加法后，将AL中的和进行调整，得到一个正确的压缩BCD数并仍然放在AL或者AX中 。
- 策略：
  - 若AL低4位的值大于9或AF=1， 则AL＋6→AL；
  - 然后，若AL高4位的值大于9或CF=1， 则AL＋60H→AL，并置CF=1。否则CF＝0。
- DAA指令影响标志AF、CF、PF、SF、ZF 。DAA指令必须紧跟在ADD或者ADC加法指令之后。

DAA指令示例

例：计算 89 ＋ 75 ＝ 164

```
MOV   AL，89H ；89H看作是两个非压缩BCD码
MOV   BL，75H ；89H看作是两个非压缩BCD码
ADD   AL，BL  ；（AL）=0FEH，AF=0，CF=0
DAA           ；（AL）=64H，CF=1
```

```
    1000 1001 (89)
+   0111 0101 (75)
    1111 1110 (FEH)
+   0110 0110 (低4位＋6调整，高4位再＋60H调整)
 1  0110 0100
```

(二)减法指令

1.不带借位的减法

- 格式：**SUB DST，SRC** ；DST ← DST-SRC

（对比：ADD）

例1： SUB BX，CX ；BX ← BX-CX

例2： SUB WORD PTR [DI]，1000H

；说明：PTR是一条修改属性运算符的伪指令，表示将PTR左边的数据类型属性赋予右边的变量或标号。本例指令的作用是将起始地址为 DS×16＋(DI)的连续两个存储单元看成一个字，将该字减去1000H后结果仍然存在原来位置。

注意：SUB指令影响标志AF、OF、PF、SF、ZF，CF。

2. 带借位的减法

- 格式：**SBB DST，SRC** ； DST ← DST−SRC−CF

（对比**ADC**）

例1： SBB AX，2030H；

功能：执行(AX) ← (AX)－2030H－CF

例2： SBB DX，[BX+20H]

功能：执行(DX)←(DX)－[DS×16＋(BX)＋20H]－CF

注意：SBB影响标志AF、OF、PF、SF、ZF，CF。

3. 减量指令

- 格式：**DEC DST** ； DST ←DST － 1（对比INC）

  注意：影响标志AF、OF、PF、SF、ZF，CF。

4. 取负指令

- 格式：**NEG DST** ； 对DST求补码，DST←0－DST

5. 比较指令

- 格式：**CMP DST，SRC** ； DST－SRC

  注意：CMP指令执行相减，但不回送结果，结果只影响标志位CF、OF、SF、ZF。

关于比较指令CMP的进一步讨论

1. 必须区分无符号数比较与有符号数比较

- 如：比较11111111B与00000000B
  - 无符号数比较：255 > 0
  - 有符号数比较：－1 < 0

2. 比较两数是否相等，应根据标志位ZF判断。若相等，则ZF=1；否则ZF=0

- 问题：如果两数不相等，执行比较指令以后如何判断两数之间的大小关系？

比较两数的大小　CMP DST, SRC

1、无符号数比较

DST≥SRC

DST=80H　SRC=58H

80H

-58H

28H

CF=0　够减

DST<SRC

DST=58H　SRC=80H

58H

-80H

D8H

CF=1　不够减

结论：用标志位CF判断无符号数的大小

CF=0，则 DST ≥ SRC

CF=1，则 DST < SRC

2、有符号数比较（分4种情况）

(1) DST>0, SRC>0 （必不溢出，OF=0）

DST=5AH, SRC=46H

```
  5AH
- 46H
------
  14H
```

SF=0, DST>SRC

DST=46H, SRC=5AH

```
  46H
- 5AH
------
  ECH
```

SF=1, DST<SRC

(2)DST>0, SRC<0 （必有DST>SRC）

DST=10H SRC=95H

```
  0001 0000B（10H）
- 1001 0101B（95H）
-------------------
  0111 1011B（7BH）
```

SF=0, OF=0

DST=62H SRC=95H

```
  0110 0010B（62H）
- 1001 0101B（95H）
-------------------
  1100 1101B（CDH）
```

SF=1, OF=1

(3)DST<0, SRC>0 (必有DST<SRC)

DST=D3H SRC=38H

```
  D3H
- 38H
------
  9BH
```

SF=1, OF=0

DST=BFH SRC=55H

```
  BFH
- 55H
------
  6AH
```

SF=0, OF=1

(4)DST<0, SRC<0 （必不溢出，OF=0）

DST=B5H SRC=9CH

```
  B5H
- 9CH
------
  19H
```

SF=0， DST>SRC

DST=9CH SRC=B5H

```
  9CH
- B5H
------
  E7H
```

SF=1， DST<SRC

结论：用标志位SF和OF判断有符号数的大小

SF、OF值相同，则 DST≥SRC

SF、OF值不同，则 DST < SRC

CMP指令示例1

若(AL)= −64　　(BL)=10

执行：**CMP AL, BL**

```
   − 64
   − 10
  ------
   − 74
```

OF=0　SF=1　　结论：(DST) < (SRC)

CMP指令示例2

若 (CL)= −100　　(DS:100)= −110

执行：**CMP CL, [100H]**

```
    -100
  −(−110)
  -------
     10
```

OF=0　SF=0　　结论：(DST) > (SRC)

6. 减法的ASCII码调整指令

- 格式：**AAS** ；　（对比AAA）
- 功能：在用SUB或者SBB对两个非压缩十进制数或用ASCII码表示的十进制数作减法后，将AL中的和进行调整，正确的结果仍然放在AL或者AX中 。
- 策略：
  - 若**AL**低**4**位的值大于**9**或**AF=1**， 则**AL－6→AL**，将**AL**高**4**位清零，同时置**AF＝1**、**CF＝1**、**AH－1→AH**。
  - 否则不对**AL**的内容做任何调整。
- **AAS**应该紧跟在减法指令之后

注意与AAA的区别

AAS指令示例

计算十进制数3－8

```
MOV AL, 03H
MOV BL, 08H
SUB AL, BL ; 低4位大于9
AAS        ; 调整，低4位减6
           ; 高4位清零
```

```
  03H
- 08H
  FBH
- 06H
  05H
```

高4位清零

结果为AL＝05H，CF＝1，表示有借位

7.减法的十进制调整

- 格式：DAS ；（对比DAA）
- 功能：在用SUB或者SBB对两个压缩BCD数作减法后，将AL中的差进行调整，得到一个正确的压缩BCD数并仍然放在AL中 。
- 策略：
  - 若AL低4位大于9或AF=1，则AL－6→AL，AF置1；否则不需对低4位做调整。
  - 然后，若AL高4位大于9或CF＝1， 则AL－60H →AL，并置CF＝1。否则不需对高4位做调整。
- DAS指令必须紧跟在减法指令之后。

注意与DAA的区别

- **DAS指令示例1：**

设AL＝BCD 56，CL＝BCD 28，求两数之差

```
SUB AL, CL ;      0101 0110    BCD 56
           ; －   0010 1001    BCD 28
           ;      0010 1110    低4位大于9，需要调整。
DAS        ; －   0000 0110    减6调整
           ;      0010 1000    高4位小于9，不需调整。

           ;      结果为BCD 28，AF＝1，CF＝0，
           ;      AL没有向AH借位。
```

- **DAS指令示例2：**

设AL＝BCD 19，CL＝BCD 28，求两数之差

```
SUB AL, CL ;      0001 1001    BCD 19
           ; －   0010 1000    BCD 28
           ;      1111 0001    低4位小于9，不需调整
DAS        ; －   0110 0000    高4位大于9，减60调整
           ;      1001 0001    CF置1，AL有借位发生

           ; 结果为91，AF＝0，CF＝1，AL向AH有借位。
```

(三) 乘法指令

- 二进制乘法特点：
  - 两个8位数相乘，结果为16位数。
  - 两个16位数相乘，结果为32位数。
- 8086乘法指令特点：
  - 乘数总是放在AL（8位）或AX（16位）中；
  - 将DX看成是AX的扩展。

1、无符号数乘法指令

- 格式：MUL SRC；AX←AL×SRC，字节

  ；DX，AX←AX × SRC，字

- 说明：1）SRC不能是立即数；

  2）若CF=OF=0，结果高半部分全为零，可去除。

  3）不能用于有符号数的相乘。压缩BCD码没有乘法指令。

8位乘法

	AL	（8位）
×	操作数	（8位）
AH	AL	（16位）

16位乘法

	AX	（16位）
×	操作数	（16位）
DX	AX	（32位）

2. 带符号数的乘法指令

- 格式：**IMUL SRC** ；AX ← AL×SRC，字节。

  ；DX、AX ← AX×SRC，字。

- 说明：1）在**8086**中，**SRC**不能是立即数；

  2）若**CF＝OF＝0**，结果的高半部分是低半部分符号位的扩展，可去除。

例1：**IMUL BL** ；**AL**的内容乘以**BL**的内容，

：结果送**AX**。

例2：**IMUL DI** ；**AX**的内容乘以**DI**的内容，

：结果送**DX**、**AX**。

80286以后，增加了以下两种类型IMUL指令：

格式：IMUL REG, SRC或IMM
操作：$REG_{16} \leftarrow (REG_{16}) \times (SRC或IMM)$
$REG_{32} \leftarrow (REG_{32}) \times (SRC或IMM)$

格式：IMUL REG, SRC, IMM
操作：$REG_{16} \leftarrow (SRC) \times IMM$
$REG_{32} \leftarrow (SRC) \times IMM$

说明：1）乘积字长和源、目的操作数的字长一致
2）可能溢出
3）SRC可用寄存器或存储器寻址方式；
REG只能是寄存器寻址方式；
IMM是立即数。

MUL, IMUL指令示例

MUL BL

(AL)=0B4H=180
(BL)=11H=17

```
      1011 0100
    × 0001 0001
    ------------
      1011 0100
 1011 0100
-------------------
101111110100
```

(AX)=0BF4H=3060

IMUL BL

(AL)=0B4H= -76,
(BL)=11H=17
$-76_{补}$=4CH

参见教材P87：IMUL实现方法

```
      0100 1100
    × 0001 0001
    ------------
      0100 1100
 0100 1100
-------------------
0101 0000 1100
```

(AX)=$(-050C)_{补}$ =FAF4H= -1292

3. 乘法的ASCII调整指令

- 格式：AAM
- 功能：

  1)将两个非压缩BCD码使用MUL指令后存放在AL中的积进行调整，使得在AX中得到正确的非压缩十进制数的乘积，高位放在AH，低位放在AL。

  2)两个ASCII码相乘之前，先屏蔽每个数字的高4位，使其先转换成非压缩BCD码，然后相乘，再用AAN对结果进行调整。
- 调整方法：（P87例3-60）

  1) AH ← AL/10 (0AH)；将AL除以10的商存放在AH中

  2) AL ← AL/10 (0AH)；将AL除以10的余数存放在AL中
- 注1：AAM必须紧跟在MUL指令之后。
- 注2：有符号乘法IMUL没有对应的十进制调整指令。

(四) 除法指令

1. 无符号数除法

- 格式：DIV SRC ;
  - 节节除：AX÷SRC，AL← 商，AH←余数
  - 字　除: (DX.AX)÷SRC，AX←商，DX←余数
- 说明：
  - SRC是除数，被除数在累加器中，且必须是除数的两倍字长。
  - 对于无符号除法，当被除数只有8位时，可将AH清零，把被除数扩展到16位；若除数是16位，被除数只有16位，可将DX清零，把把被除数扩展到32位。
  - SRC可以是除立即数以外所有寻址方式。

2. 有符号数除法

- 格式：IDIV SRC ；
  - 节节除：AX÷SRC，AL← 商，AH←余数
  - 字　除：(DX.AX)÷SRC，AX←商，DX←余数
- 说明：
  - 操作数、商和余数都是带符号数，并且余数的符号与被除数相同。
  - 无/有符号数除法都有除法溢出的问题。原因是如果被除数高位的绝对值大于除数的绝对值，而存放商的寄存器字长只有被除数字长的一半，商“装不下”。产生类型号为0的除法错中断，相当于执行了除0运算。
  - IDIV执行后，所有的标志位没有意义。
  - 被除数字长也必须是除数的两倍。对于有符号除法，不能简单地通过将高位置零的方式进行字长扩展。

3. 扩展字节为字

- 格式：CBW ；　　AX ←AL
- 功能：将AL的符号位扩展到AH中。将一个字节的有符号数扩展为一个字。

4. 扩展字为双字

- 格式：CWD ；　　DX,AX ←AX
- 功能：将AX的符号位扩展到DX寄存器中。将一个字的有符号数扩展为一个双字。

- 说明：CBW和CWD执行后不影响标志位。

6. 除法的ASCII调整

- 格式：AAD ；
- 功能：在除法运算之前，将BCD码转换成二进制数。
- 策略：(AL)←(AH)×10＋AL，(AH)←0
- 注意：AAD指令应该在除法指令之前。
- AAD指令示例：计算压缩BCD码28÷4

```
MOV  AX, 0208H      ；AX<--被除数
MOV  CL, 4          ；除数
AAD                 ；调整，(AX)=28
DIV  CL             ；结果(AL)＝7，(AH)＝0
```

习题二：P122～

- 9
- 10

三、逻辑运算和移位指令

1. 逻辑运算指令
2. 非循环移位指令
3. 循环移位指令

（一）逻辑运算指令

① NOT DST

② AND DST，SRC

③ OR DST，SRC

④ XOR DST，SRC

⑤ TEST DST，SRC

⑥ 位扫描和位测试（386后新增）

①、逻辑非指令

- 格式： NOT DST ； DST按位取反
- 说明：本指令不影响标志位

NOT指令示例1

```
MOV AL, 52H        ；执行前AL＝01010010
NOT AL             ；执行后AL＝10101101
```

NOT指令示例2

```
NOT  BYTE PTR[BX]
```

对逻辑地址为DS:BX的存储单元内容取反后送回该单元。

②、逻辑与指令

- 格式： AND DST，SRC ；
- 功能： (DST) ← (DST)∧(SRC)
- 说明：影响标志位PF、SF、ZF，使CF=0、OF=0

AND指令示例1

```
MOV AL, 32H ；执行前AL为ASCII码’2’
AND AL, 0FH ；屏蔽高4位，执行后AL＝02H
```

AND指令示例2

```
AND  AX, AX    ；执行后 AX内容不变， CF=0
```

③、逻辑或指令

- 格式： OR DST，SRC ；
- 功能： (DST) ← (DST)∨(SRC)
- 说明：影响标志位PF、SF、ZF，使CF=0、OF=0

OR指令示例

```
MOV     AX,  0508H
OR      AX,  3030H
```

执行后AX＝3538H，将AX中原先存放的两位非压缩BCD码转换成两个ASCII码’5’和’8’。

④、异或指令

- 格式： XOR DST，SRC ；
- 功能： (DST) ← (DST) ⊕ (SRC)，按位异或。
- 说明：影响标志位PF、SF、ZF，使CF=0、OF=0

XOR指令示例1

```
MOV AL, 0B6H ;
XOR AL, 0FH  ;
```

```
   1011 0110
⊕  0000 1111
-------------
AL=1011 1001
   不变 变反
```

与0/1异或不变/求反

XOR指令示例2

```
XOR AL, AL      ; 清零操作，且CF=0
```

思考：用一条指令使AX清零，有几种方法？

⑤、测试指令

- 格式：TEST DST，SRC ；
- 功能：(DST)∧(SRC)，但是不回送结果
- 说明：影响标志位PF、SF、ZF，使CF=0、OF=0

TEST指令示例1

测试AL最低位是否为1，若是1则转移

```
TEST AL, 01H
JNZ  NEXT      ；是1非0 (ZF=0)，转NEXT
```

AND指令示例2

测试AX最高位是否为1，若是1则转移

```
TEST AX, 8000H
JNZ  THERE     ；是1非0 (ZF=0)，转THERE
```

⑥位测试和位扫描指令

- 80386以后CPU新增的命令
- 格式：OP DST，SRC ； OP为以下操作码
- OP：
  - BT (bit test) 位测试
  - BTS(bit test and set) 位测试并置1
  - BTR(bit test and reset) 位测试并清0
  - BTC(bit test and complement) 位测试并变反
  - BSF(bit scan forward) 正向位扫描
  - BSR(bit scan reverse) 反向位扫描

指令	操作
BT	BT测试DST中由SRC指定的位，将测试位送CF。
BTS	测试DST中由SRC指定的位，将测试位送CF，并将DST中该位置1。
BTR	测试DST中由SRC指定的位，将测试位送CF，并将DST中该位置0。
BTC	测试DST中由SRC指定的位，将测试位送CF，并将DST中该位变反。
BSF	从低位至高位扫描SRC第一个为“1”的位，并将位号送DST。
BSR	从高位至低位扫描SRC第一个为“1”的位，并且将位号送DST。

位测试指令示例1

BT AX, 4

若执行前 **(AX)=1234H**，则执行后， **CF=1**

若执行前 **(AX)=1224H**，则执行后， **CF=0**

位测试指令示例2

若执行前 **(AX)=0000H**

BTS AX, 4

执行后，**CF=0**，**(AX)=0010H**

位扫描指令示例

BSF ECX, EAX ； 从右至左扫描

BSR EDX, EAX ； 从左至右扫描

若执行前 **(EAX)=60000000H**，

则执行后， **(ECX)=29d**，**(EDX)=30d**

(二) 非循环移位指令

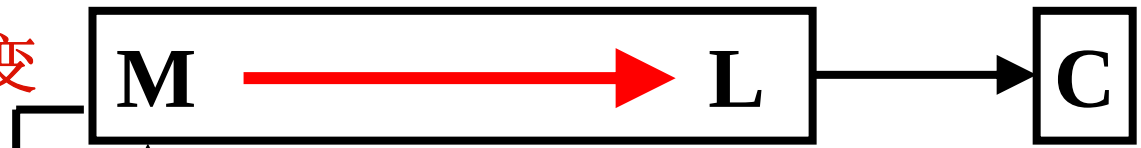
① SHL DST，count；逻辑左移

② SHR DST，count；逻辑右移

③ SAL DST，count；算数左移

④ SAR DST，count；算数右移

最高位不变

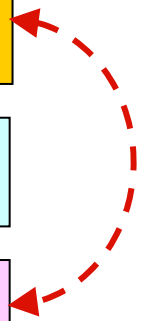


- 注意：
  - 移位指令影响标志位CF、OF、PF、SF和ZF。
  - 如果只移一位，指令中可用立即数'1'指出移位的位数，如果超过1位，移位的位数必须事先放在CL中。
  - CL可以表示的移位次数最大为255，80286以后的CPU规定最大的移位次数为31（00011111），超过此范围也仅截取低5位，

- 例：
```
MOV   CL，4
SAL   DX，1    ；DX中的数左移1位
SAL   AX，CL ；AX中的数左移4位
SHL   AL，CL ；AL中的数左移4位（同SAL AL，CL）
SHR   AL，CL ；AL中的数右移4位
```

- 例：已知(AL)=0B4H, (CF)=1，分析下列指令执行后的结果

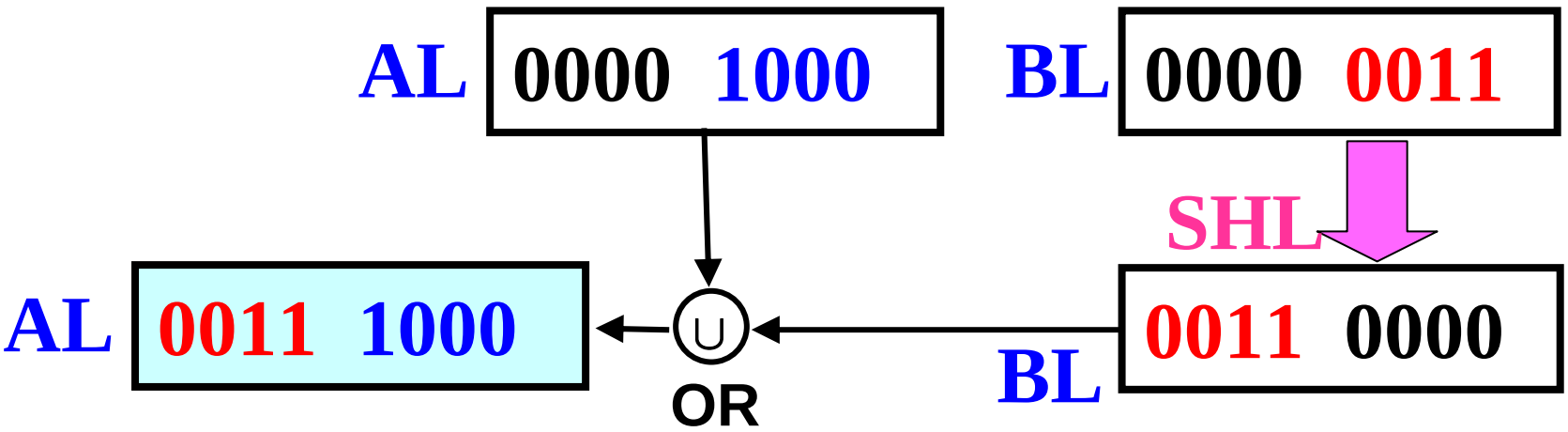
	CF	AL
执行前	1	10110100
(1) SAL AL, 1	1	01101000
(2) SAR AL, 1	0	11011010
(3) SHL AL, 1	1	01101000
(4) SHR AL, 1	0	01011010



- 算术左移与逻辑左移的结果相同
- 算术右移与逻辑右移的结果不同

例：分析下列程序片段，求执行后(AL)＝？

```
PRO1:   MOV   AL，04H
        MOV   BL，03H
        MOV   CL，4
        SHL   BL，CL
        OR    AL，BL
        HLT
```



- 通常：
  - 算术右移N位相当于有符号数除以2^N(有例外)
  - 逻辑右移N位相当于无符号数除以2^N
  - 算术/逻辑左移N位相当于无符号数乘以2^N

- 例：已知 AL＝ 01100100（64H＝100）

  MOV CL, 5

  SAR AL, CL ；AL＝00000011（03H＝3）

  当最右边的“1”移出后，除法运算结果不对

- 例题：已知变量Y中为一字节无符号数，计算(Y)×10，积放在AX中。

变换：(Y)×10＝(Y)×(8＋2)＝(Y)×2＋(Y)×8

提问：为什么要这样变换？

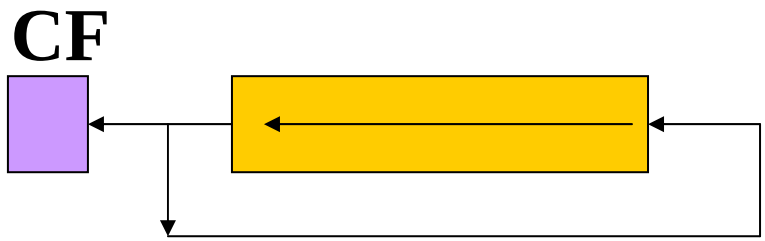
```
MOV   AL, Y      ; AL<--(Y)
MOV   AH, 0
SHL   AX, 1      ; (Y)×2
MOV   BX, AX
SHL   AX, 1      ; (Y)×4
SHL   AX, 1      ; (Y)×8
ADD   AX, BX     ; (Y)×10
```

思考题：利用移位指令计算 DX＝3×AX＋7×BX

(三) 循环移位指令

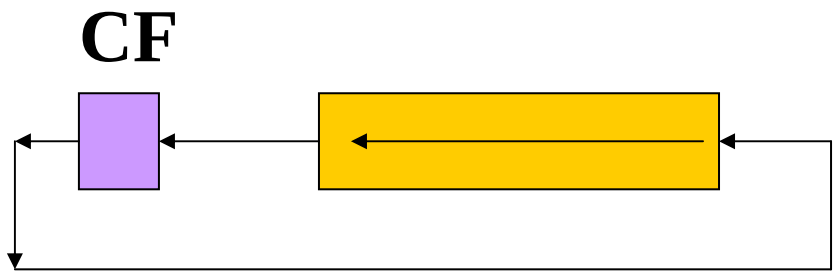
1) 循环左移

ROL DST，count



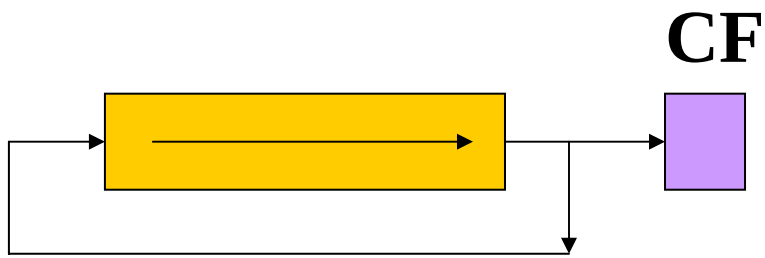
2) 带进位循环左移

RCL DST，count



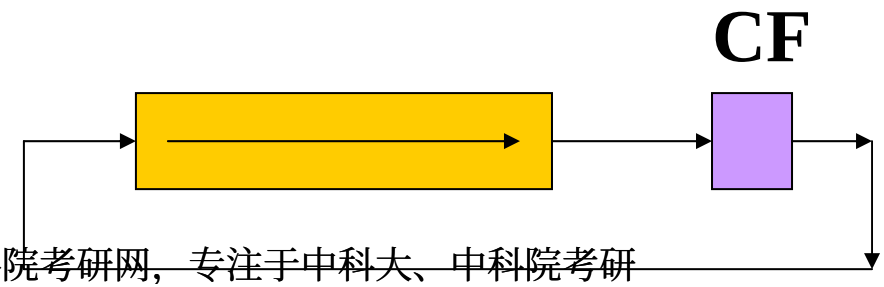
3) 循环右移

ROR DST，count



4) 带进位循环右移

RCR DST，count



- 说明
  - 不带CF位的循环移位有时也称为小循环；带CF位的称为大循环。
  - 如果只移一位，指令中可用立即数'1'指出移位的位数，如果超过1位，移位的位数必须事先放在CL中。
  - 循环移位指令操作结果只对CF和OF有影响
    - CF由最后一次移入CF的内容决定；
    - OF只在移位次数为1时有效，若移位后的符号位发生改变，则OF=1，否则为OF＝0。多位循环移位时，OF的值不确定。

四、字符串操作指令

- 字符串－存放在存储器中的一系列字或字节数据。
- 8086提供5条串操作指令
  - 1、MOVS——串传送指令
  - 2、CMPS——串比较指令
  - 3、SCAS——串扫描指令
  - 4、LODS——装入串指令
  - 5、STOS——存储串指令
- 80386以后增加了
  - 6、INS——串输入
  - 7、OUTS——串输出

串操作的隐含规定

1. 源串地址——DS:SI，允许使用段超越
2. 目的串地址——ES:DI，不许使用段超越。若源串和目的串在同一段，应将ES和DS相同。
3. 每执行一次串指令，SI和DI自动修改指针。
   - DF＝0/1，SI和DI自动＋n/－n
   - 当操作数类型为字节或字或双字时，n为1或2或4。
4. 串长度（字数或字节数或者双字数）放在CX中。每传送一次CX←(CX－1)；当CX＝0时，串操作结束。

5. 串操作的重复前缀

- **REP** 重复，直到CX=0
- **REPE/REPZ** 相等/为零则重复，直到CX=0
- **REPNE/REPNZ** 不相等/不为零则重复，直到CX=0

6. 串操作指令的不同形式

- 目的：表明操作对象是字节/字/双字
- 方法1：用指令中的操作数类型表明
- 方法2：指令助记符后面加B/W/D表明

①串传送指令

- 格式：MOVS 目的串，源串

  或者：MOVSB、MOVSW、MOVSD

- 操作：目的串ES:DI←源串(DS:SI)

  SI←(SI+n/-n) ；修改源指针

  DI←(DI+n/-n) ；修改目的指针

  CX←(CX-1) ；重复次数－1

- 重复条件：一般配合前缀REP，直到CX＝0

- 例：将100个字节从开始地址为AR1单元区传送到开始地址为AR2的单元区。

```
LEA    SI，AR1  ；SI指向源串地址
LEA    DI，AR2  ；DI指向目的串地址
MOV    CX，100  ；置计数器
CLD             ；DF＝0（地址自动加1）
REP   MOVSB     ；开始传送，无需再指明
HLT             ；源串和目的串。
```

②串比较指令

- 格式：CMPS 目的串，源串
  或者：CMPSB、CMPSW、CMPSD
- 操作：　源串(DS:SI) －目的串ES:DI
  SI←(SI+n/-n)，　DI←(DI+n/-n)
  CX←CX-1
- 重复条件：
  - 根据前缀REPE/REPZ，或者REPNE/REPNZ；
  - 或者遇到CX＝0
- 影响标志位： A、C、O、P、S， Z！

- 例：串String1和String2分别定义在DS段和ES段中，比较两串，如相等则转移到标号NEXT处。

```
String1  DB ‘HELP’          ；定义String1
String2  DB ‘HEPP’          ；定义String2
……
CLD                          ；DF=0
LEA   SI, String1            ；源串地址→SI
LEA   DI, String2            ；目的串地址→DI
MOV   CX, 4                  ；重复次数→CX
REPZ  CMPSB                  ；重复比较，直到发现
                             ；不同或者CX＝0
JZ    NEXT                   ；串相同则转移
……
NEXT:
```

③ 字符串扫描指令

- 格式：SCAS 目的串

  或者：SCASB、SCASW、SCASD

- 操作：“关键字”－目的串ES:DI、DI←(DI+n/-n)。
- 功能：从一个字串中查找一个与AL/AX/EAX中的内容（即所谓的“关键字”）相同或不同的字符。
- 重复条件：
  - 根据前缀REPE/REPZ或者REPNE/REPNZ
  - 或者遇到CX＝0
- 影响标志位： A、C、O、P、S， Z!

- 例:在串“That is CAI”中查找字符‘a’，找到，则转到标号FOUND处。

```
String  DB  ‘That is CAI’   ；定义串
 ……
CLD                          ；DF＝0
LEA    DI, String            ；串地址→DI
MOV    AL, ‘a’               ；关键字符’a’→ AL
MOV    CX, 11                ；重复次数→ CX
REPNZ SCASB                  ；重复扫描
JZ       FOUND               ；找到目的串元素转移
……
FOUND:
```

④ 字符串装入指令

- 格式：LODS 源串

  或者：LODSB、LODSW、LODSD

- 操作：AL/AX/EAX←源串DS:SI

  SI←(SI+n/-n)

  CX←(CX-1)

- 功能：将DS:SI的内容取到AL/AX/EAX中。
- 注意：由于AL/AX/EAX仅能保留一次装入的数据，所以该指令一般不使用重复前缀。

⑤ 字符串存储指令

- 格式：**STOS 目的串**

  或者：**STOSB、STOSW、STOSD**

- 操作：**AL/AX/EAX → 目的串ES:DI**

  DI←(DI+n/-n)

  CX←(CX-1)

- 功能：将**AL/AX/EAX**的内容存储到**ES:DI**指示的目的串单元中。

- **STOS**可以使用重复前缀吗？

  √ **STOS**可以与**REP**前缀连用，用一个放在**AL/AX/EAX**中的常数对一个数据块进行初始化。

⑥ 串输入指令INS

- 格式：INSB、INSW、INSD
- 操作：目的串ES:DI←[DX]

  DI←(DI+n/-n)

  CX←(CX-1)
- 功能：从[DX]寻址的端口输入一串数据到ES:DI为起址的一连串存储单元中。
- 可以使用重复前缀。

（386新增指令，参见教材P480）

⑦ 串输出指令OUTS

- 格式：OUTSB、OUTSW、OUTSD
- 操作：源串DS:SI→[DX]

  SI←(SI+n/-n)

  CX←(CX-1)
- 功能：将以DS:SI为起址的一连串存储单元中的内容输出到[DX]指明（寻址）的IO端口。
- 可以使用重复前缀。

（386新增指令，参见教材P480）

五 控制转移指令

- 作用：控制程序流程。可分为以下几类：

(一) 无条件转移和过程调用指令

(二) 条件转移指令

(三) 条件循环控制指令

(四) 中断指令以及中断返回指令

- 学习难点
  - 转移或者调用目的地址的寻址方式

(一) 无条件转移和过程调用指令

- JMP ；无条件转移指令
- CALL ；过程调用指令
- RET ；过程返回指令

1. JMP无条件转移指令

- 格式：JMP 目的
- 功能：使程序无条件的转移到目的地址去执行。

- 两种类型
  - 段内转移或近转移：JMP指令与转移的目的地址在同一代码段，转移仅需改变IP。
  - 段间转移或远转移： JMP指令与转移的目的地址不在同一段中，转移时CS和IP都要改变。
- 两种方式
  - 直接方式：指令代码中直接给出目的地址。
  - 间接方式：目的地址包含在某个寄存器或存储单元中。

- 两种类型＋两种方式 ＝ 四种组合

（1）段内直接转移

（2）段内间接转移

（3）段间直接转移

（4）段间间接转移

(1) 段内直接转移

① 段内直接短转移：

JMP SHORT 标号 ； 2字节指令

② 段内直接近转移：

JMP NEAR PTR 标号 ； 3字节指令

或 JMP 标号 ；

- 操作：CS不变；IP＝IP＋2（或＋3）＋disp
- 说明：
  - 编程只需确定短转还是近转，地址由汇编自动计算。
  - 短转是近转的特例，位移量disp为8位，转移范围-128字节至+127字节(以下条指令为基准)，偏移量用补码表示。
  - 近转位移量disp为16位，转移范围-32768字节至+32767字节(以下条指令为基准)，NEAR PTR可省略不写。

(2) 段内间接转移

- 格式：JMP WORD PTR OPR
- 操作：CS不变，IP←(EA)，EA由OPR除了立即数以外的任何寻址方式确定，如：JMP REG16
- 例一：JMP BX ；

  假设BX＝4500H，执行后程序转到CS:4500H处。
- 例二：

```
ADDRESS DW 2000H      ；定义转移地址
  …
LEA  SI, ADDRESS      ；偏移量→SI
  …
JMP  WORD PTR[SI]     ；转移到CS:2000H
```

(3) 段间直接(远)转移

- 格式：**JMP FAR PTR OPR(标号)** ；
- 操作：IP← 标号的段内偏移量

  CS← 标号所在的段基址

  其中：**FAR PTR**为段间转移操作符

- 例：假设标号**PROG-F**所在的段基址为**3500H**，偏移量为**1234H**。则执行

  JMP FAR PTR PROG-F

  之后，程序转移到**3500H:1234H**处继续执行。

(4) 段间间接转移

- 格式：JMP DWORD PTR［SI＋DISP］

- 操作：由(DS)×16＋SI＋DISP寻址连续4个字节的内存单元，其中前两个单元的内容→IP，后两个单元内容→CS。即：

  IP←[(DS)×16＋SI＋DISP]

  CS←[(DS)×16＋SI＋DISP＋2]

- 例：假设DS＝2500H，SI＝1300H，内存单元

  (26425H)＝4500H，(26427H)＝32F0H。则执行在指令

  JMP DWORD PTR［SI＋125H］

  以后，程序转移到32F0H:4500H处继续执行。

* 为什么不是从偶地址开始？有许多指令的字长是奇数字节。

2. 过程调用与过程返回指令

- 过程（Procedure）——为完成某些特定功能而编写的相对独立的程序模块。也常被称为子程序（Subroutine）。
- 在主程序中使用过程调用指令CALL对过程进行调用。
- 过程以语句PROC开头，以语句ENDP结尾。在ENDP前要安排一条过程返回语句RET，使过程结束后能够正确返回主程序。
- 过程嵌套——在过程中又调用了另外一个过程。

- 近过程调用——过程调用指令和被调用的过程在同一代码段。
- 远过程调用——过程调用指令和被调用的过程不在同一代码段。
- **CALL**指令执行的两个步骤

  ① 将返回地址（紧跟**CALL**指令的下一条指令）**IP**压栈。近调用只需压栈**IP**；远调用需先将**CS**压栈，再将**IP**压栈。（教材**P105～P106**）

  ② 转到过程入口地址去执行被调用的过程。
- 与无条件转移指令相比，**CALL**指令增加了返回地址的压栈和弹出操作。调用过程入口地址的寻址方法与无条件转移指令基本相同

- 过程的调用方式
  - 根据被调用过程与CALL指令的相对位置关系以及过程入口地址的寻址方法，过程调用有以下4种方式

(1) 段内直接调用

(2) 段内间接调用

(3) 段间直接调用

(4) 段间间接调用

- 注意：没有段内短调用指令，因为考虑到被调过程不太可能在CALL指令的－128字节～127字节范围内。

- 过程的返回
  - 近调用返回：堆栈弹出一个字→IP，SP+2→SP
  - 远调用返回：堆栈先弹出一个字→IP，SP+2→SP；再弹出一个字→CS，SP+2→SP。

(1) 段内直接调用与返回

- 调用格式：CALL PROG_N
- 说明：PROG_N是近标号，也是子程序名。
- 操作：

  SP←SP－2

  返回地址IP压栈

  IP←IP＋DISP（汇编程序自动计算）

- 返回格式：RET
- 操作：IP出栈，IP ←（SP和SP＋1）单元内容，

  SP ← SP＋2

- 段内直接调用示例：（P106例3-83）
  - 调用前：CS＝2000H，IP＝1050H，SS＝5000H，SP＝0100H，PROG_F与CALL指令的之间的字节距离DISP为1234H。
  - 调用指令： CALL PROG_F ；（3字节）
  - 执行后：

    1）返回地址为1050H＋3＝1053H，因此应将1053H压栈

    2）过程入口地址应为1053H＋1234H＝2287H，所以程序转到CS:2287H处执行。

(2) 段内间接调用与返回

- 调用格式：CALL WORD PTR OPR
- 操作：　SP←SP－2
  返回地址IP压栈
  IP←（EA）
- 说明：EA是由OPR的寻址方式确定的16位有效地址。
  例如以下两种形式：

```
CALL  BX                    ；地址在寄存器中
CALL WORD PTR (BX＋SI)      ；地址被寻址的内存中
```

- 返回格式：RET
- 操作：与段内直接调用相同。

(3) 段间直接调用与返回—I

- 调用格式：CALL FAR PTR PROG_F
- 说明：
  - 被调用的过程与CALL指令不在同一个代码段。
  - PROG_F是远标号，也是子程序名。
  - 在程序中给出子程序名，就可以得到子程序的入口偏移地址(EA)及段基址。
- 操作：1）返回地址压栈

  SP←SP－2，CS压栈

  SP←SP－2，IP压栈

  2）转到过程入口地址

  IP ←子程序对应的偏移量（EA）

  CS←子程序所在的段基址

(3) 段间直接调用与返回—II

- 段间返回格式：RET
- 操作：

1) 返回地址IP出栈：IP ←[(SP)＋1，(SP)]

SP ←(SP)＋2

2) 返回地址CS出栈：CS ←[(SP)＋1，(SP)]

SP ←(SP)＋2

- 段间直接调用示例：（P107例3-85）
  - 调用指令：CALL FAR PTR PROG_F
  - 假设调用前： CS＝1000H，IP＝205AH；SS＝2500H，SP＝0050H；标号PROG_F所在单元的地址指针： CS＝3000H，IP＝0050H
  - 该指令长度为5个字节，下条需要返回的指令地址为：CS＝1000H，IP＝205AH＋5＝205FH。因此堆栈段2500H:004EH←1000H，2500H:004DH←205FH。
  - 调用后程序转移到3000H:0050H处执行。
  - 过程执行完毕，RET指令顺序将IP和CS从堆栈中弹出，程序返回1000H:205FH继续执行，堆栈指针最后为2500H:0050H。

(4) 段间间接调用与返回—I

- 调用格式：CALL DWORD PTR OPR

- 说明：
  - 被调用的过程与CALL指令不在同一个代码段。
  - 被调用的过程的入口地址（CS和IP）放在内存某处连续4个存储单元中。内存单元由OPR表明的寻址方式进行寻址。
  - 段间调用必须保存返回地址的IP和CS。
- 操作：1）返回地址压栈

  SP←SP－2，CS压栈

  SP←SP－2，IP压栈

- **(4) 段间间接调用与返回—II**

2）转到过程入口地址

IP ← （EA）

CS ← （EA＋2）

EA为OPR寻址方式确定的内存单元有效地址。

- **段间返回格式：RET**
- 操作：

1) 返回地址**IP**出栈：　IP ← [(SP)＋1，(SP)]

SP ← (SP)＋2

2) 返回地址**CS**出栈：　CS ← [(SP)＋1，(SP)]

SP ← (SP)＋2

- 段间间接调用示例：（P109例3-86）
  - 调用前：DS＝1000H，BX＝200H，
    (10200H)＝31F4H，(10202H)＝5200H
  - 调用指令：　　CALL DWORD PTR [BX]
  - 执行后：1）顺序将返回地址CS和IP压栈；
    2）转到过程入口地址
  - 根据调用指令的寻址方式可计算出存放过程入口地址的内存单元首地址为：DS×16＋BX＝10200H
  - 从10200H单元取得2个字节31F4H→IP
  - 从10200H＋2＝10202H取得2个字节5200H→CS
  - 执行完毕，RET指令顺序将IP和CS从堆栈中弹出，程序返回主程序继续执行。

返回的特别情况——带参数的返回指令

- 段内带参数返回：**RET n**
- 段间带参数返回：**RET n**
- 说明：
  - **n**称为弹出值。它使得**CPU**在弹出返回地址之后再从堆栈中弹出**n**个字节。**n**用表达式表示，其值可以是**0000H～FFFFH**范围内的任意一个偶数。
- 作用：
  - 主程序可以通过堆栈向被调用的过程传递传输，这些传输必须在调用前压入堆栈，过程在运行中通过堆栈指针取出这些参数。
  - 过程运行完毕，这些参数已失去作用，利用**RET n**可以顺便将这些传输从堆栈中弹出。

(二) 条件转移指令

- 根据上条指令执行后CPU的状态标志决定是否转移。
- 8086的所有条件转移都是段内短转移。转移目的地址与转移指令都在同一代码段内，并且相对地址仅用一字节表示，目的地址与紧跟转移指令后的那条指令的距离在-128到+127字节范围内。80386相对转移地址为4个字节，范围大多了。
- 条件转移指令通常位于比较指令或者算术逻辑运算指令之后。
- 条件转移指令的目的地址均用标号表示。
- 条件转移指令的格式：**Jcc（条件） 标号**

- 条件转移指令可以分为两大类：

  (1) 直接标志转移

  (2) 间接标志转移

(1) 直接标志转移（10条）

- 8086 CPU提供了10条直接标志转移指令
- 所有直接标志转移指令见下表。

（参见教材P110表3-11）

序号	指令助记符	测试条件	指令功能
1	JC	CF＝1	有进位 转移
2	JNC	CF＝0	无进位 转移
3	JZ/JE	ZF＝1	结果为零/相等 转移
4	JNZ/JNE	ZF＝0	结果不为零/相等 转移
5	JS	SF＝1	符号为负 转移
6	JNS	SF＝0	符号为正 转移
7	JO	OF＝1	溢出 转移
8	JNO	OF＝0	不溢出 转移
9	JP/JPE	PF＝1	奇偶为1/偶数 转移
10	JNP/JPO	PF＝0	奇偶为0/奇数 转移

- 思考题：如果8086 CPU中的条件转移语句——例如：JC Next——中的转移目标地址超出-128～+127的范围，会怎样？
- 如果超出范围，汇编时则会引起编译错误：“Jump out of range by xxx byte(s)”。那么该如何解决？
- 如果汇编程序编译时遇到上述情况，应将条件转移语句配合无条件转移语句一起使用。原代码可改为：

```
        JNC  L1
        JMP  NEXT
L1:     ……
```

(2) 间接标志转移（8条）

- 间接标志转移通常位于CMP指令之后，通过测试某个或某些状态位比较两数大小。
- 指令中不直接给出状态标志位的测试条件，但是仍然以某个标志的状态或几个标志的状态组合作为测试条件。
- 间接标志转移共有8条指令，每条指令均有两种不同的助记符，两者之间用“/”分割。
- 必须严格区分有符号数和无符号数比较测试指令。
- 间接标志转移参见下表（P110表3-12）

类别	指令助记符	测试条件	指令功能
无符号数比较测试	JA/JNBE	CF∨ZF=0	高于/不低于等于 转移
	JAE/JNB	CF=0	高于等于/不低于 转移
	JB/JNAE	CF=1	低于/不高于等于 转移
	JBE/JNA	CF∨ZF=1	低于等于/不高于 转移
有符号数比较测试	JG/JNLE	(SF ⊕ OF)∨ZF=0	大于/不小于等于 转移
	JGE/JNL	SF ⊕ OF=0	大于等于/不小于 转移
	JL/JNGE	SF ⊕ OF=1	小于/不大于等于 转移
	JLE/JNG	(SF ⊕ OF)∨ZF=1	小于等于/不大于 转移

注：A－Above，B－Below，E－Equal to，G－Great than，L－Less than

(三) 条件循环控制指令

- 实质上是一组增强型的条件转移指令，用于控制一段程序的重复运行。当条件不满足或者重复次数没有达到预定次数时，重复运行该段程序；否则脱离重复程序段转而执行其它指令。
- 循环条件的控制与条件转移指令类似，也是段内短转移。转移偏移量为8位，并且都是补码表示的负数。
- 重复次数由CX内容决定，每循环一次CX减1，直到CX－1＝0。循环控制指令不影响任何标志位。
- 循环控制指令共有4条。

(1) LOOP 标号 ；

CX－1≠0则转移到标号处循环。

(2) LOOPZ/LOOPE 标号 ；

ZF＝1(相等或结果为零)且CX－1≠0则转到标号处循环。

(3) LOOPNZ/LOOPNE 标号 ；

ZF＝0(不等或结果不为零)且CX－1≠0则转到标号处循环。

(4) JCXZ 标号 ；

CX＝0则转到标号处。该指令通常放在循环开始处。为使程序跳出循环，只需将CX清零。

* 也可看成一条件转移指令，80386还增加了JECXZ指令，ECX=0则转移。

循环指令示例1

求长度为10的字节数组ARRAY之和，并将和存入TOTAL中。

```
        LEA  SI, ARRAY     ；数组首地址→SI
        MOV CX, 10         ；数组长度→CX
        MOV AX, 0
AGAIN:  ADD  AL, [SI]      ；求数组和
        ADC  AH, 0
        INC  SI            ；修改指针
        LOOP  AGAIN
        MOV TOTAL, AX      ；存和
```

其中语句 LOOP AGAIN 相当于：

```
        DEC  CX
        JNZ  AGAIN
```

循环指令示例2

在某一字节串中寻找第一个非0字节。设串首地址在DI中，串末地址在BX中。

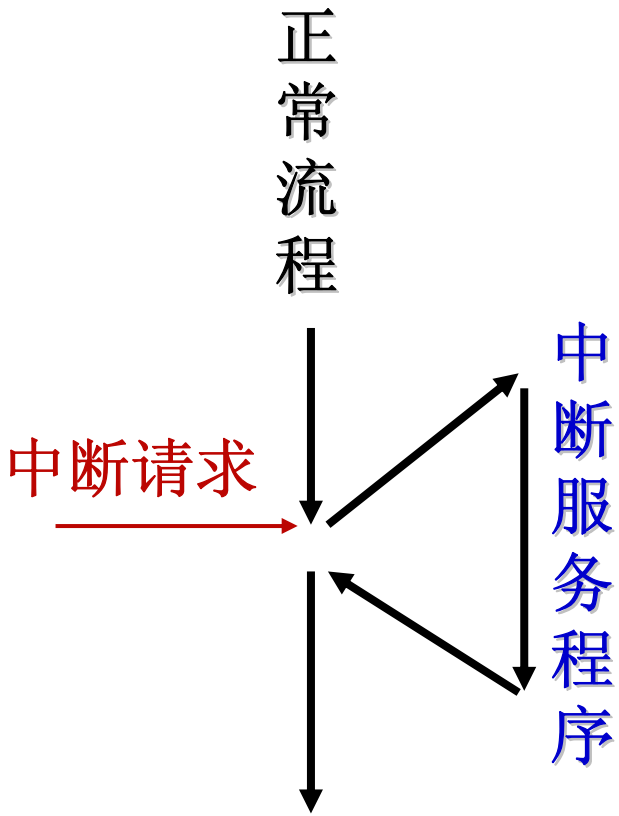
思考：还有更好的方法实现吗？

```
        SUB  BX, DI              ；计算串长度
        INC   BX                 ；串长度在BX中
        MOV  CX, BX              ；串字节数→CX
        DEC DI
AGAIN:  INC DI                   ；修改指针
        CMP BYTE PTR [DI], 0     ；串元素=0？
        LOOPZ  AGAIN             ；循环查找
        JNZ  FOUND               ；找到非0字节跳转
        ……
FOUND:
```

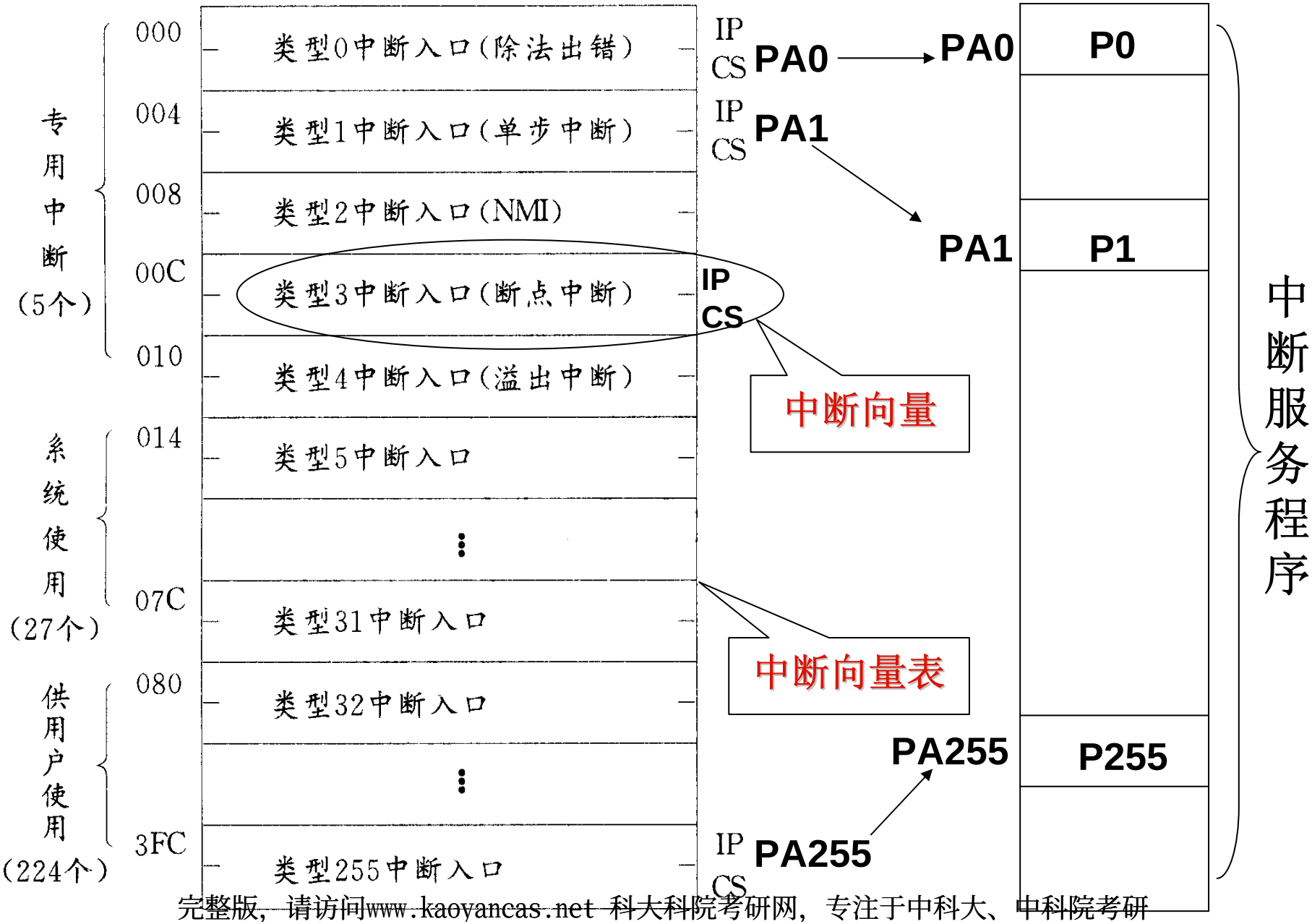
(四) 中断指令以及中断返回指令

(1) 中断概念

- 中断：由于某些事件计算机中止当前程序的运行，转去执行中断服务程序，中断服务程序完毕后，又返回继续运行正常程序的过程称为中断。
- 中断源：引起中断的原因。8086的中断源有两种，外部中断和内部中断。



- 中断响应：若中断请求得到CPU的应允(响应)，CPU保护断点、保护标志寄存器值，然后找到中断服务程序的入口地址，转相应的中断服务程序。
- 中断返回：中断服务程序结束时，通过执行中断返回指令IRET，从堆栈中恢复中断前CPU的状态和断点，返回正常程序继续执行。
- 中断向量：中断服务程序的入口地址。存储CS和IP共需要4个字节。
- 中断向量表：位于内存00000～003FFH的区域中存储256个中断向量地址的连续空间称为中断向量表。



(2) 中断指令

思考：为什么要清除这两个标志？

① INT n 软件中断

- 操作：

  1）SP←SP-2， (SP+1,SP)←PSW， TF←0，IF←0

  2）SP←SP-2， (SP+1,SP)←CS

  3）SP←SP-2， (SP+1,SP)←IP

  4）IP←(n×4)， CS←(n×4+2)

② INTO （＝INT 4）

- 说明：溢出中断，也是类型4中断，OF＝1则执行该指令。在带符号数进行加减运算后，必须安排一条INTO指令，以便对运行结果是否溢出进行监测。

③ IRET　中断返回

- 说明：总是被安排在中断服务程序的出口处，执行后从堆栈依次弹出程序断点和标志寄存器，以使CPU继续原来被中断的程序。
- 操作：

```
IP←(SP+1,SP)      ；IP出栈
SP←SP+2           ；修改栈顶指针
CS←(SP+1,SP)      ；CS出栈
SP←SP+2           ；修改栈顶指针
PSW←(SP+1,SP)     ；标志寄存器
SP←SP+2           ；修改指针
```

六、处理器控制指令

8086只提供对C、D、I 3个标志的处理指令。

思考：如何对T标志操作？

(1) 标志处理指令

CLC —— 进位标志CF置0

CMC —— 进位标志CF求反

STC —— 进位标志CF置1

CLD —— 方向标志DF置0

STD —— 方向标志DF置1

CLI —— 中断标志 IF置0

STI —— 中断标志 IF置1

(2) 其它处理器控制指令

NOP—— 空操作（增加延时）

HLT—— 停机（暂停，RESET或中断请求才唤醒）

WAIT——等待（实现8086对8087的控制）

ESC—— 换码（实现8086对8087的控制）

LOCK——封锁（指令前缀，实现总线封锁）

BOUND——界限（286后新增，检查数组下标是否越界）

ENTER——建立堆栈帧（286后新增）

LEAVE——释放堆栈帧（286后新增）

* 堆栈帧是在堆栈段中开辟的一个存储区，专门用于过程调用（包括多层嵌套调用）时的参数传递。

七、指令的执行时间和软件延时

- 8086指令执行所需的时钟周期不同。
- 寄存器之间的数据传递以及寄存器移位指令执行时间最短，只需2个时钟周期。
- 有些指令的执行时间不确定。例如：条件转移指令在发生转移时需要16个时钟周期，不发生转移时仅需4个时钟周期。
- 最长的指令执行时间接近50个时钟周期。例如：段间间接调用，被调用的过程入口地址采用相对基址变址寻址时（参见P119～P120）。

- 当程序需要延时或者定时时，应该仔细计算每条指令的执行时间。
- 可以使用**NOP**指令以及循环控制指令构成延时程序。
- 延时时间较长时，可以采用嵌套的多重循环程序来实现。

习题三：P123～

- 11　2）～8）、10）、12）
- 12
- 13
- 14
- 15
- 17
- 18