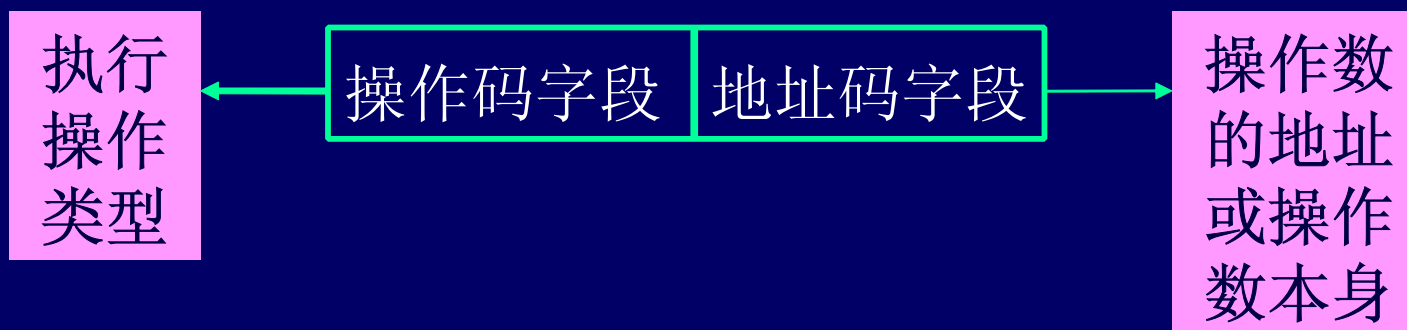


## 第2章 微型计算机指令系统

**机器指令**: 计算机**CPU**执行某种操作的命令编码。

**指令系统**: 某一类型**CPU**中所有机器指令的集合

## 机器指令格式



## 二、地址码

地址码表示操作数的地址。

一般的操作数有被操作数，操作数及操作结果三种类型

三地址指令：

操作码

A1

A2

A3

三操作数指令

二地址指令：

操作码

A1

A2

双操作数指令

一地址指令：

操作码

A1

单操作数指令

零地址指令：

操作码

## 四、指令助记符

机器指令是由**1**，**0**组成的特定的二进制数序列。

为了便于书写和阅读，每条指令通常用**3**个或**4**个英文缩写字母来表示。这种缩写码叫做指令助记符，

表3-1 典型的指令助记符

典型 指令	指令 助记 符	二进制 操作码	典型 指令	指令助 记符	二 进 制 操作码
加法	<b>ADD</b>	<b>001</b>	转移	<b>JSR</b>	<b>101</b>
减法	<b>SUB</b>	<b>010</b>	存储	<b>STR</b>	<b>110</b>
传送	<b>MOV</b>	<b>011</b>	读数	<b>LDA</b>	<b>111</b>
跳转	<b>JMP</b>	<b>100</b>			

## 3.2 寻址方式

操作数的寻址方式：**CPU**指令中规定的寻找操作数所在地址的方式。

操作数所在地址是指——操作数的段内偏移地址（有效地址**EA**）

## 1. 源操作数和目标操作数

**源操作数：** 此类操作数的特点是只参与操作，但不改变原始数据。

例：已知 AX=1000H, BX=2000H

**MOV AX, BX** ; 指令功能: **BX→AX** ;

**BX**为源操作数

指令执行后:

**AX=2000H**

**BX=2000H**

目标操作数： 此类操作数除参与操作外，还要保存操作结果。即指令执行后，目标操作数内容将会随着操作结果而变化，如上例，指令执行后，AX的内容将发生变化。

注意：在一条指令中，源操作数与目标操作数的寻址方式不一定相同

源操作数位于右边，目标操作数位于左边

操作码 目标操作数，源操作数



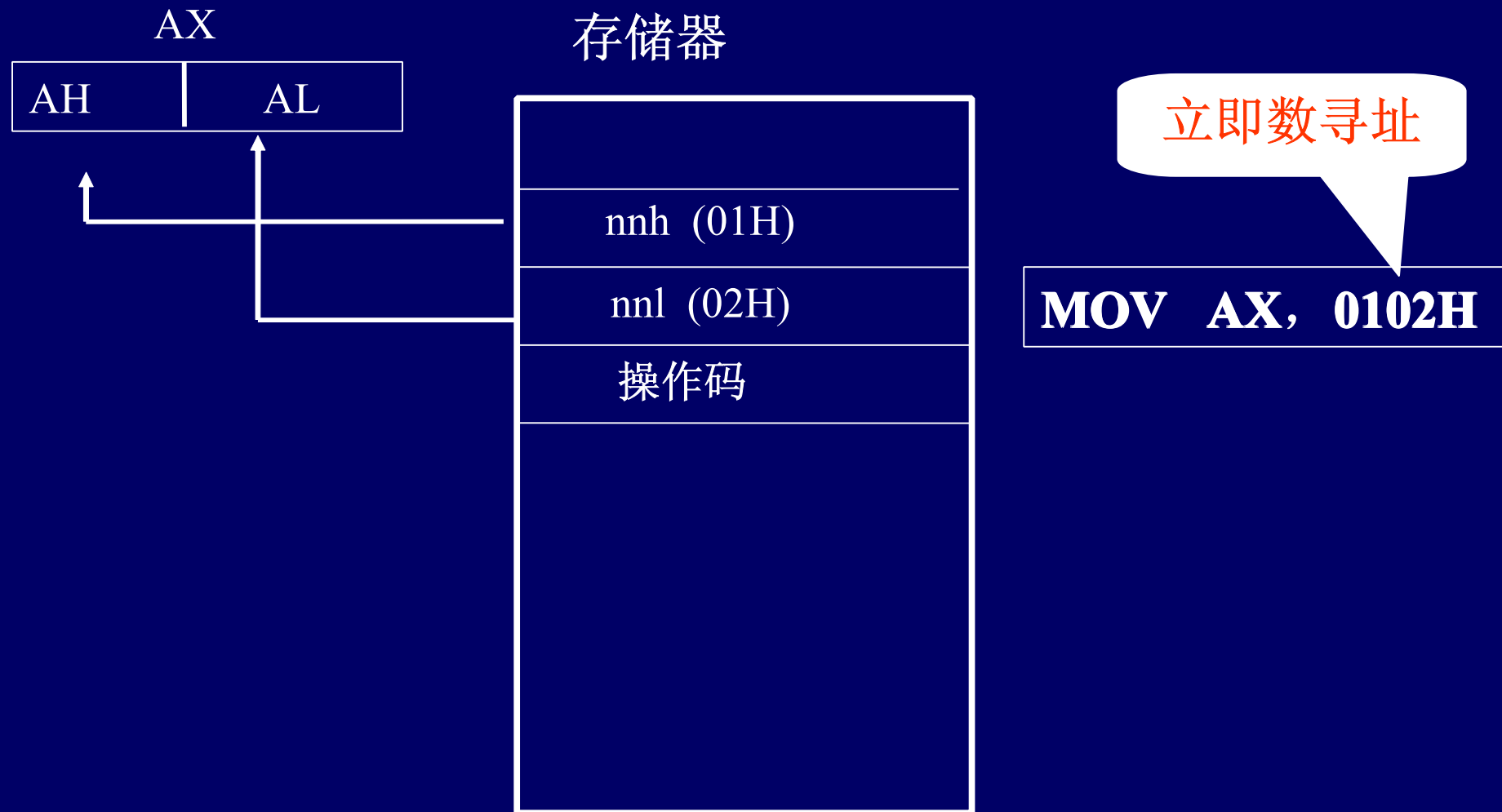
## 2、操作数的寻址方式

操作数，是存储在内存的不同单元或CPU的寄存器中。对于存储在内存中的操作数，指令中的地址码或者直接给出地址，多数并不直接给出操作数的地址，而是说明与这个地址有关的某些信息，计算机根据这个信息，再计算出真正的地址。

### 三、 Intel 8086/8088操作数的寻址方式

**8086**操作数的寻址方式包括：立即数寻址、寄存器寻址、直接寻址、寄存器间接寻址、变址寻址、基址寻址、基址-变址寻址

**1、立即数寻址：**这种寻址方式所提供的操作数直接放在指令中，紧跟在操作码的后面，即**地址码字段就是操作数本身**，而不是操作数地址。如下图所示。



立即数寻址用于给寄存器赋初值

## 立即数寻址方式

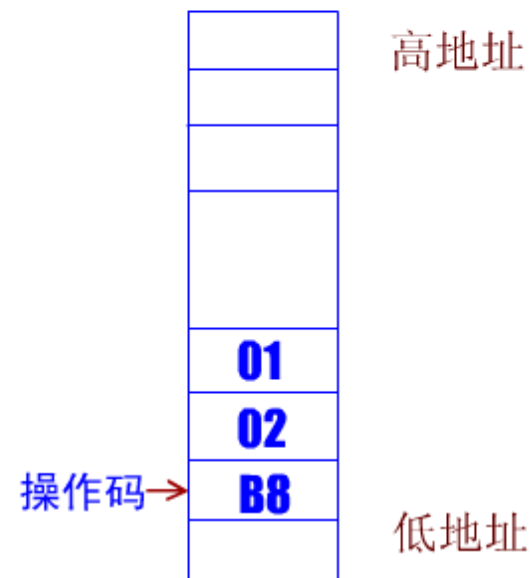
**AX**

--	--

  
**AH AL**

**MOV AX,0102H**

存储器



**MOV AX, 0102H ; AX ← 0102H**

## 2、直接寻址

地址码字段的**16**位二进制数据是操作数地址的段内偏移地址。

操作数的物理内存地址等于数据段段寄存器**DS**中的值左移**4**位后与**16**位偏移地址之和。

例：MOV AX, [2000H] ;

直接寻址

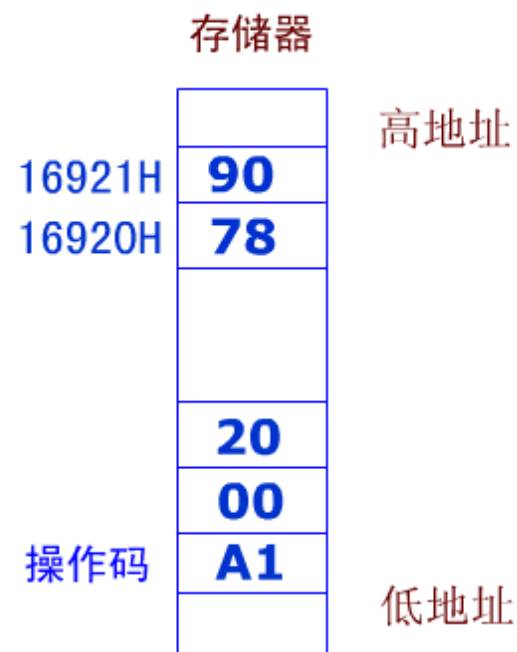
DS: 2000H



AX

3012H

## 直接寻址方式



**MOV AX, [2000H]**



**MOV AX, [2000H] ; AX ← DS:[2000H]**

### 3、寄存器寻址

操作数存放在**CPU**的内部寄存器（**AH**、**AL**、**BH**、**BL**、**AX**、**BX**、**CX**、**DI**、**SI**等）中。

例如

例： MOV CL, AH

源操作数与目标操作  
数均为寄存器寻址

## 寄存器寻址方式

**AX**

--	--

  
**AH AL**

**BX**

12	34
----	----

  
**BH BL**

**MOV AX, BX**



**MOV AX, BX ; AX ← BX**



#### 4、寄存器间接寻址

操作数存放在内存中，其内存单元的**段内偏移地址**（又称有效地址**EA**）可存于且只能存于下面**4**个寄存器之一

**[SI]、[DI]、[BX]、[BP]**

因寄存器中的内容不是操作数本身，而是操作数的有效地址**EA**，故称寄存器间接寻址。

注意：如果一个**寄存器用[]括起来**（当然，不是所有的寄存器都可用[]括起来），则说明这是**寄存器间接寻址**方式，这时**寄存器的内容不是操作数，而是操作数所在内存单元的段内偏移地址**。

EA =  $\left\{ \begin{array}{l} \text{SI} \\ \text{DI} \\ \text{BX} \\ \text{BP} \end{array} \right\} \begin{array}{l} \text{DS} \\ \text{SS} \end{array}$

当间址寄存器为**SI**、**DI**、**BX**中任一个时，寻址默认的段寄存器为**DS**。

当间址寄存器为**BP**时，寻址默认的段寄存器为**SS**。

例：**MOV AX, [SI]** ； 此指令源操作数为寄存器间接寻址。

寄存器  
间接寻址

比较下面二条指令    **MOV AX, SI;**

**MOV AX, [SI];**

若**SI=2100H**，则第一条指令执行后，**AX**的内容为**2100H**;

第二条指令执行后**AX**的内容要取决于**DS:2100H**单元的内容

指出下列指令的错误:

**MOV    AL, BX**

错!

**MOV    AL, [BX]**

对!

**MOV AX, [BX] ; AX ← DS:[BX]**

## 5、变址寻址

操作数在内存中，其有效地址**EA**由**2**个变址寄存器之一的内容，也可加上指令中给出的**8位/16位**偏移量得出。

$$EA = \begin{pmatrix} SI \\ DI \end{pmatrix} + \begin{pmatrix} \text{8位disp或16位disp} \end{pmatrix}$$

同样，寻址时若使用**SI**、**DI**寄存器，则默认的段寄存器为**DS**

**MOV AX, [SI+06H] ; AX ← DS:[SI+06H]**



**MOV AX, 06H[SI] ; AX ← DS:[SI+06H]**





**MOV AX, [SI+06H] ; AX ← DS:[SI+06H]**

**MOV AX, 06H[SI] ; AX ← DS:[SI+06H]**

## 6、基址寻址

操作数在内存中，其有效地址**EA**由**2个基址寄存器**之一的内容，也可加上指令中给出的**8位/16位偏移量**得出。

$$EA = \begin{pmatrix} BX \\ BP \end{pmatrix} + \begin{pmatrix} 8\text{位disp或}16\text{位disp} \end{pmatrix}$$

同样，寻址时若使用**BX**寄存器，则默认的段寄存器为**DS**

寻址时若使用**BP**寄存器，则默认的段寄存器为**SS**



**MOV AX, [BX+06H] ; AX ← DS:[BX+06H]**



**MOV AX, 06H[BP] ; AX ← SS:[BP+06H]**

**MOV BLOCK[BP], AX ; SS:[BP+BLOCK] ← AX**



## 7、基址—变址的寻址方式

操作数在内存中，其有效地址是一个**基址寄存器**与一个**变址寄存器**的内容之和，即有效地址**EA**表示为

$$EA = \begin{array}{|c|} \hline B \\ \hline X \\ \hline \end{array} + \begin{array}{|c|} \hline SI \\ \hline DI \\ \hline \end{array}$$

**注意：**基址加变址的寻址方式中必须是一个基址寄存器 + 一个变址寄存器。

寻址时使用的寄存器中，只要有**BP**，则默认的段寄存器为**SS**，否则默认的段寄存器为**DS**。

**MOV CX, [BP+BX] 或 MOV [SI+DI], AX**

以上指令都是错误的，原因二个不能都是变址寄存器或基址寄存器

**MOV AX, [BX+SI] ; AX ← DS:[BX+SI]**

**MOV AX, [BX+SI] ; AX ← DS:[BX+SI]**



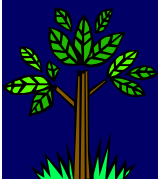
**MOV AX, [BX][SI] ; AX ← DS:[BX+SI]**

例:    **MOV    [SI+10H],    BL**  
         **AND    CX, [DI+BP]**  
         **MOV    WORD PTR [SI+BX], 00H**

## 相对的基址加变址寻址方式

操作数在内存中，其有效地址是由指令中给出的一个带符号的**8/16**位的偏移量**disp**，一个基址寄存器及一个变址寄存器内容之和，即有效地址**EA**的形式为：

$$EA = \begin{array}{|c|} \hline BX \\ \hline BP \\ \hline \end{array} + \begin{array}{|c|} \hline SI \\ \hline DI \\ \hline \end{array} + \begin{array}{|c|} \hline 8\text{位disp} \\ \hline 16\text{位disp} \\ \hline \end{array}$$



同样，当使用BP时，默认的段寄存器为SS。

MOV AX, [BP+SI+0124H] → 相对基址加变址寻址

此种寻址方式仍要注意二个基址寄存器（**BP**、**BX**）或二个变址寄存器（**SI**、**DI**）不能同时作为间址寄存器。

练习：分别指出下列各指令源操作数与目标操作数的寻址方式

1. **MOV AL, 20H**

2. **MOV [200H],BX**

3. **AND CX, [BX]**

4. **ADD BYTE PTR[SI+30H], 30H**

5. **MOV DS,[DI+BP+1000H]**

或 **MOV DS,[DI][BP+1000H]**

### 3.1.3

### 8086/8088的指令系统

8086/8088的指令按功能可分为六大类，分别是：

数据传送类、

算术运算类、

逻辑运算与移位类、

字符串处理类、

控制转移类

及处理器控制类。



## 一、 指令系统符号说明

<b>AH、AL、BH、BL、CH、CL、DH、DL</b>	八位通用寄存器
<b>AX、BX、CX、DX、SP、BP、DI、SI</b>	十六位通用寄存器
<b>SP</b>	堆栈指针
<b>IP</b>	指令指针
<b>FLAGS</b>	标志寄存器
<b>DI、SI</b>	目的和源变址寄存器
<b>CS、DS、SS、ES</b>	段寄存器
<b>SEG</b>	段寄存器通用符号
<b>REG</b>	通用寄存器组

<b>AC</b>	<b>AX或AL/AH</b> （取决于操作数长度）
<b>SRC</b>	源操作数
<b>DST</b>	目的操作数
<b>MEM</b>	存储器操作数
<b>MEM/REG</b>	存储器或通用寄存器操作数
<b>DATA</b>	立即数， <b>8位或16位</b>
<b>OPRD</b>	操作数
<b>n</b>	<b>8位立即数</b>
<b>nn</b>	<b>16位立即数</b>
<b>nnnn</b>	<b>32位立即数</b>

## 一、数据传送类指令

分为四种：

通用数据传送、

累加器专用数据传送

地址传送

和标志传送。

### （一）通用数据传送指令

此类指令包括最基本的传送指令**MOV**，

堆栈操作指令**PUSH**和**POP**，

数据交换指令**XCHG**和查表指令**XLAT**。

## 1. 基本的传送指令MOV

指令一般形式: **MOV DST, SRC**

指令功能: **SRC**  **DST** (字节或字) ;

指令执行后, 源操作数不变, 目标操作数发生变化且与源操作数相同。

例如: 指令 **MOV AL, BL;**

若该指令执行前, **AL=25H, BL=86H,**

则指令执行后, **AL=BL=86H。**

# MOV指令的功能



- 把一个字节或字的操作数从源地址传送至目的地址

MOV reg/mem, imm

； 立即数送寄存器或主存单元

MOV reg/mem/seg, reg

； 寄存器送（段）寄存器或主存单元

MOV reg/seg, mem

； 主存送（段）寄存器

MOV reg/mem, seg

； 段寄存器送寄存器或主存

在应用此类指令时，应注意以下几点：

- ① **CS**和**IP**两个寄存器不能作为目的操作数，也就是说这两个寄存器的值是不能用**MOV** 指令来修改的。
- ② 当用**BX**、**DI**、**SI**进行寄存器间接寻址时，默认的段寄存器为**DS**，当用**BP**来间址时默认的段寄存器为**SS**。
- ③ 当将一个立即数传送到内存时，必须指明内存单元的类型属性。内存单元的类型有**3**种：

**DWORD PTR**——双字类型，**4byte**

**WORD PTR**——字类型，**2byte**

**BYTE PTR**——字节类型。

例: **MOV BYTE PTR [1000H], 00H;**

此指令功能是将内存数据区地址是**1000H**字节单元清零,

**MOV WORD PTR [1000H], 00H**

指令的功能是将内存数据区地址是**1000H**字单元清零,

即将**1000H**、**1001H**二个字节单元清零

④两个操作数长度必须相同

⑤符合正确规定的寻址方式

⑥所有通用传送指令都不影响状态标志。

注意!

⑦不存在以下指令:

存储器向存储器的传送指令; 立即数至段寄存器; 段寄存器之间的传送指令



例：指出下列指令的功能 （已知 **AX=2A35H**, **BX=1000H**）

1) **MOV AX, 2100H;**    AX=?                    **2100H**

2) **MOV [BX], AX;**    AX=? , BX=?            **2A35H, 1000H**

3) **MOV CL, AL;**        AX=? , CL=?            **2A35H, 35H**

4) **MOV [2000H], AL;**        AX=?                    **2A35H**

## 2. 堆栈操作指令

在实际程序中，把需要重复执行的操作编成子程序。  
所以一个应用程序常分为主程序和子程序两部分

主程序调用子程序时，要暂停主程序的执行，转去执行子程序，为了保证在子程序执行完正确返回主程序继续执行，系统必须把主程序中调用子程序指令的下一条指令地址（称断点地址，由段基址及偏移地址两部分组成）保存下来。

如果子程序再次调用子程序，也需要将嵌套子程序的断点地址保留。在子程序返回时，要求后保存的地址先取出来，先保存的地址后取出来。

- 即要求后保存的值先取出来，就是说数据要按照【后进先出】的原则保存。
- 能实现数据按照【后进先出】的原则保存的内存区域，称为堆栈。

- 堆栈是一个“后进先出FILO”（或说“先进后出FILO”）的主存区域，位于堆栈段中；**SS段寄存器**记录其段基地址
- 堆栈只有一个进口或出口，即当前栈顶；用**堆栈指针寄存器SP**指定
- 堆栈只有两种基本操作：**进栈和出栈**，对应两条指令**PUSH**和**POP**

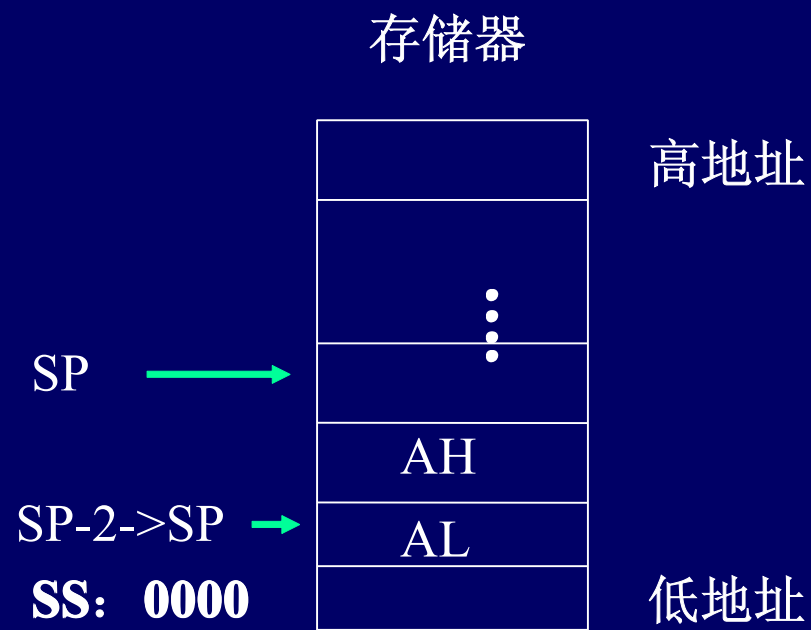
一种叫压入操作（**PUSH**），另一种称弹出（**POP**）操作。在**8086/8088**系统中，无论压入或弹出操作，都是以字为单位。

### （1） 入栈指令

指令格式：**PUSH OPRD**

指令功能：先修改 $SP-2 \rightarrow SP$ ，然后把一个源操作数（1个字）传送到由**SP**所指向的堆栈的顶部

# **PUSH**指令的功能



例: **PUSH AX**

**PUSH [2000H]**

因为是字操作，  
每执行一次**PUSH**，  
则使**SP-2→SP**

**PUSH MEM;** 把内存单元的一个字压入堆栈

**PUSH REG;** 把**16**位寄存器内容压入堆栈

**PUSH SEG;** 把段寄存器内容压入堆栈



## (2) 出栈指令

指令格式: **POP OPRD**

指令功能: 是先将栈顶的一个字弹出送往目的操作数, 再修改**SP+2→SP**。

出栈指令有如下几种形式:

**POP MEM/REG**

**POP REG**

**POP SEG**

例: **POP CX**

**POP [BX]**

# POP指令的功能

## 堆栈操作的原则

- 压入堆栈时，操作数的长度是字，**16位**
- 压入堆栈时，先修改栈顶地址**SP-2**，后压入堆栈
- 压入堆栈时，地址从高地址向低地址变化
- 出堆栈时，先弹出堆栈，后修改栈顶地址**SP+2**

### 3. 交换指令

指令格式: **XCHG OPRD1, OPRD2**

指令功能: 把二个字节或字的操作数相互交换。这二个操作数不能是立即数, 也不能同时为存储器操作数。

可有如下几种形式:

**XCHG REG, MEM/REG**

**XCHG AC, REG;** 若**REG**为8位寄存器, **AC=AL**或**AH**

若 **REG**为16位寄存器, **AC=AX**

# XCHG指令的功能

## （二） I/O数据传送指令

**I/O**（输入/输出）指令完成累加器**AL**（**AX**）与**I/O**端口间的数据传送功能。此类指令中，一个操作数为**AX**（16位）或**AL**（8位），另一个是**I/O**端口。

● **I/O**端口的地址范围总共**64K**，**0000H—FFFFH**

**I/O**端口地址的表示方式:

**直接方式:** 若端口地址 $\leq \mathbf{FFH}$ , 端口地址用立即数直接给出;

**间接方式:** 若端口地址 $> \mathbf{FFH}$ , 需要将**I/O**端口地址存入**DX**中。用**DX**可寻址**100H—0FFFFH**的端口。

## 1) 端口输入指令IN

指令一般格式: **IN AC, PORT**

指令功能: 把1个字节或1个字, 由输入端口传送给**AL**或**AX**。又分以下几种形式:

直接方式: 地址 $\leq$ **FFH**

**IN AL, n**

**IN AX, n** (n为端口地址)

例: **IN AL,**  
**20H;**

间接方式: 地址 $>$ **FFH**

**IN AL, DX**

**IN AX, DX**

**MOV DX, 2A01H**  
**IN AL, DX**



## 2) 端口输出指令OUT

指令一般格式: **OUT PORT, AC**

指令功能: 把**AL** (**AX**) 中的**1**个字节 (字), 传送到某个输出端口。

### 直接方式

**OUT n, AL**

**OUT n, AX** (**n**为端口地址)

### 间接方式

**OUT DX, AL**

**OUT DX, AX**

**OUT 20H, AL**

**MOV DX, 3100H**

**OUT DX, AL**

### （三） 地址传送指令

地址传送指令有**3**条：①取有效地址指令**LEA**；②地址指针装入**DS**指令**LDS**；③地址指针装入**ES**指令**LES**。

#### 1. 取有效地址指令：

指令一般格式：**LEA REG, MEM**

指令功能：将源操作数的**段内偏移地址**传送给目的操作数。

**注意：**源操作数必须是一个内存操作数；目的操作数必须是一个**16**位的通用寄存器。

**LEA BX, [DI+110H], 设DI=0500H**

此指令中，源操作数的段内偏移地址由**DI**寄存器内容加上偏移量**110H**决定，因此指令执行后，**BX**中的内容为**0610H**。

**MOV BX, [DI+110H], 设DI=0500H**

则是把数据段中的段内偏移地址为**0610H**中的**1**个字送往**BX**。

## 2. 地址指针装入**DS**指令

指令一般格式: **LDS REG, MEM**

指令功能: 源操作数必须是内存操作数, 把源操作数 (内存中的双字数据—**32位逻辑地址**) 的高字部分 (段基址) 传送给**DS**, 低字部分 (段内偏移地址) 送指令规定的寄存器。

**LDS SI, [0010H]**

设对应地址单元的内容

**(DS: 10) = 80H**

**(DS: 11) = 01H**

**(DS: 12) = 00H**

**(DS: 13) = 20H**

则指令执行后, **SI=0180H, DS=2000H**

### 3. 地址指针装入ES指令

指令一般格式: **LES REG, MEM**

指令功能: 把源操作数（内存中的双字数据）的高位字传送给**ES**（**16**位段基址），低位字传送给指令规定的**16**位寄存器中。

地址传送指令不影响标志位。

## （四）标志传送指令

**8086/8088**有四条标志传送指令。

### 1. 标志装入**AH**指令

指令格式：**LAHF**

指令功能：把标志寄存器的低**8**传送给**AH**。

这样，相应的符号标志**SF**、零标志**ZF**、辅助进位标志**AF**、奇偶标志**PF**和进位标志**CF**被传送至**AH**的对应位。

**FLAGS**

				OF	DF	IF	TF	SF	ZF		AF		PF		CF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

**AH**

--	--	--	--	--	--	--	--

## 2、 设置标志指令

指令格式: **SAHF**

此指令功能与**LAHF**相反，是把**AH**内容传给标志寄存器**FLAGS**的低**8**位，高**8**位不受影响。



### 3、 标志压入堆栈指令

指令格式: **PUSHF**

**PUSHF**指令先修改堆栈指针, 即 $SP-2 \rightarrow SP$ , 把整个标志寄存器内容压入堆栈。指令本身的执行不影响标志位。

### 4、 标志弹出堆栈指令

指令格式: **POPF**

这条指令把当前堆栈指针所指的一个字, 传送给标志寄存器, 同时修改堆栈指针, 即 $SP+2 \rightarrow SP$ 。

### 三. 算术运算指令

**8086/8088**提供加、减、乘、除四种基本的算术操作。操作数可是带符号数的字或字节，也可是不带符号数的字或字节。若是带符号数，则用补码表示。

**8086/8088**还提供了各种校正操作指令，可以进行BCD码或ASCII码表示的十进制数的算术运算。

## 1. 二进制加减法指令

### (1) 不带进位的加减法指令 **ADD** 和 **SUB**

指令格式及功能: **ADD** 目标, 源; 目标+源  $\longrightarrow$  目标

**SUB** 目标, 源; 目标-源  $\longrightarrow$  目标

指令用于无符号或带符号数的字节或字的加减运算。

源操作数可以为寄存器、存储器或**8/16**位立即数;

目的操作数可以为寄存器或存储器;

但两个操作数不能同时为存储器操作数。

指令的执行结果影响六个状态标志。 **OF/SF/ZF/AF/PF/CF**

**例5.2 ① ADD BL, 3AH;**

**指令功能为: BL+3AH→BL**

**设BL = 04H, 则此加法指令执行后, 寄存器BL  
中的内容为3EH。**

无符号数和带符号数是由编程者区分的。

实际上，无符号数的加法与带符号数的补码加法是完全一致的，**CPU**没有必要去区分。

关键在于程序员如何去看待。**CPU**的运算结果将同时影响进位标志与溢出标志。如果程序员认为这是无符号数运算，则只考虑进位标志而忽略溢出标志。反之，如果程序员认为这是带符号数运算，则只考虑溢出标志而忽略进位标志。

**MOV AL, 7EH**

**MOV BL, 5BH**

**ADD AL, BL**

执行后，结果（**AL**）=**D9H**，此时各标志位状态：**SF=1**，**ZF=0**，**AF=1**，**PF=0**，**CF=0**，**OF=1**。

认为是无符号数相加，**CF=0**，没有进位，结果正确

认为是带符号数，**OF=1**，产生溢出，结果错误。

标志位	1	0
<b>OF</b>	<b>OV</b>	<b>NV</b>
<b>DF</b>	<b>DN</b>	<b>UP</b>
<b>IF</b>	<b>EI</b>	<b>DI</b>
<b>SF</b>	<b>NG</b>	<b>PL</b>
<b>ZF</b>	<b>ZR</b>	<b>NZ</b>
<b>AF</b>	<b>AC</b>	<b>NA</b>
<b>PF</b>	<b>PE</b>	<b>PO</b>
<b>CF</b>	<b>CY</b>	<b>NC</b>

## (2) 带进（借）位的加减法指令**ADC**及**SBB**

此类指令通常用来实现多字节、多字的加/减运算。

除了在加法运算时须在最低位加上进位位**CF**值，或在减法运算时在最低位减去借位**CF**值外，其它与**ADD**，**SUB**指令相同。

指令格式及功能：

**ADC**    目标， 源； 目标+源+**CF**     $\longrightarrow$     目标

**SBB**    目标， 源； 目标-源-**CF**     $\longrightarrow$     目标



### (3) 加法和减法的ASCII码调整指令

加法调整: { 非压缩BCD码-----AAA  
                  压缩BCD码-----DAA

减法调整: { 非压缩BCD码-----AAS  
                  压缩BCD码-----DAS

**AAA指令**: 先用**ADD**（或**ADC**）指令进行**8**位数加法，相加结果存在**AL**；执行**AAA**后，对**AL**中的结果进行调整，并将非压缩**BCD**码的低位存于**AL**、高位存于**AH**中



例：计算十进制数的和 **7+8=?**

首先将 **7、8**以非压缩**BCD**码 存于**AL**、**BL**中，**AH=0**

用以下指令段实现：

**MOV AX, 0007H**

**MOV BL, 08H**

**ADD AL, BL**       **AL=0FH**

**AAA**                       **AX=0105H**

结果：**1**和**5**分别存于**AH**、**AL**中

**AH 01H**

**AL 05H, CF=AF=1**

例：计算十进制数的和 **7+8=?**

首先将 **7、8**以非压缩**BCD**码 存于**AL**、**BL**中，**AH=0**

用**DAA**指令实现：

**MOV AX, 0007H**

**MOV BL, 08H**

**ADD AL, BL**       **AL=0FH**

**DAA**                       **AL=15H**

**AH = 00H**

**CF=0, AF=1**

#### (4) 加1/减1指令**INC/DEC**

指令格式及功能: **INC** 目标; 目标+1  $\longrightarrow$  目标

**DEC** 目标; 目标-1  $\longrightarrow$  目标

指令字节较短, 运行速度快, 主要用于在循环程序中修改地址指针或循环次数。

**INC**及**DEC**指令运算结果不影响**CF**标志, 对其他标志位的影响与加减法指令**ADD**、**SUB**相同。

例: **INC SI** ; **SI+1**  $\longrightarrow$  **SI**

**DEC CX** ; **CX-1**  $\longrightarrow$  **CX**

### (5) 求补及比较指令**NEG**、**CMP**

求补**NEG**及比较**CMP**指令都属于特殊的二进制减法运算。

指令格式及功能: **NEG** 目标; 0-目标  目标

**CMP** 目标, 源; 目标-源  目标  
 状态标志

利用**NEG**指令可计算负数的绝对值。

比较指令主要用于比较两个数之间的关系, 可以判断两者是否相等, 或两个中哪一个大。在比较指令之后, 根据**ZF**标志判断二数是否相等。

例：已知有如下指令段：(设**DS=1000H**)

**MOV SI, 3000H**

**MOV AL, 3CH**

**MOV [SI], AL**

**ADD [SI], AL**

问：指令执行后,存储单元地址是**13000H**中  
内容变为多少？

**78H**

## 2、二进制乘除法指令

### (1)无符号数乘法指令:

指令格式及功能: **MUL SRC**;

字节乘法:**AX** ← **AL\*SRC**

字乘法:**DX和AX** ← **AX\*SRC**

**SRC: REG/MEM**

### (2)带符号数乘法指令

指令格式及功能: **IMUL SRC**;

字节乘法:**AX** ← **AL\*SRC**

字乘法:**DX和AX** ← **AX\*SRC**

**SRC: REG/MEM**

对ZF、SF、AF、PF影响不确定

### (3) 无符号数除法指令:

指令格式及功能: **DIV**

**SRC: REG/MEM**

**SRC ;**

字节除法:  $AL \leftarrow AX / SRC$   
 $AH \leftarrow AX \% SRC$   
字除法:  $AX \leftarrow DX\_AX / SRC$   
 $DX \leftarrow DX\_AX \% SRC$

### (4) 带符号数除法指令:

指令格式及功能: **IDIV**

**SRC: REG/MEM**

**SRC ;**

字节除法:  $AL \leftarrow AX / SRC$   
 $AH \leftarrow AX \% SRC$   
字除法:  $AX \leftarrow DX\_AX / SRC$   
 $DX \leftarrow DX\_AX \% SRC$

对ZF、SF、AF、PF、OF、CF影响不确定



### (5) 转换指令（符号扩展）

**CWB:** 将**AL**的符号位扩展到**AH**中;

**CWD:** 将**AX**的符号位扩展到**DX**中;

对无符号数，符号扩展时只是将AH（DX）清零！

对带符号数，扩展如下：

CWB:	AL < 80H	AH = 0
	AL ≥ 80H	AH = FFH

CWD:	AX < 8000H	DX = 0
	AX ≥ 8000H	DX = FFFFH

## 四、逻辑运算与移位类指令

为了处理字节或字中各位信息，**8086/8088**提供了三种位处理指令：**逻辑运算指令、移位类指令和循环移位类指令。**

### 1. 逻辑运算指令

此类指令包括**逻辑与、逻辑或、逻辑非、逻辑异或和逻辑测试。**

所有的指令都对其操作数**按每一位**进行逻辑操作；

操作数可以是**字节或字**；

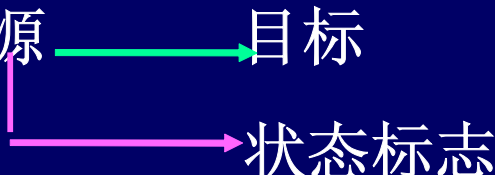
①逻辑非指令**NOT**: 逻辑非指令主要用来使某数变反。

指令格式及功能: **NOT** 目标;  $\overline{\text{目标}}$   $\longrightarrow$  目标

逻辑非指令不影响状态标志。

例如 **NOT AL**; 若原**AL**=**01000111B**, 则指令执行后,  
**AL**=**10111000B**,

## ②逻辑与指令AND:

指令格式及功能: **AND** 目标, 源; 目标  $\wedge$  源 

指令对状态标志的影响: 执行后将使**CF**、**OF**标志复位  
按结果影响**PF**、**ZF**、**SF**标志  
**AF**标志不定!

---

利用**AND**指令, 可以屏蔽操作数的某些位。

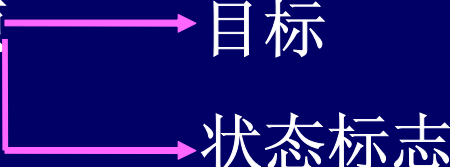
**AND AL, 0FH;**                      **AL**高四位清零, 低四位不变

**AND CX, 0FFFEH;**                  **CX**最低位清零, 其余位不变

**AND AL, AL;**                        **AL**自身相与, 其值不变

### ③逻辑或指令**OR**

指令格式及功能: **OR** 目标, 源; 目标  $\vee$  源



目标  
状态标志

或指令对状态标志的影响与**AND** 指令相似。

或操作常用来使目标操作数某位置位。

例: **OR AL, 80H**; 此指令执行后, 将使**AL** 的最高位置**1**, 其余位不变。

**OR AX, AX**; 执行后, **AX**中内容不变, 但可设置标志位。  
**OR BX, 0FH**; 低四位置**1**, 其余位不变

#### ④逻辑异或指令**XOR**

此类指令对状态标志的影响也与**AND**指令相似。

**XOR**指令可使目标操作数某些位取反

指令格式及功能: **XOR** 目标, 源; 目标  $\oplus$  源



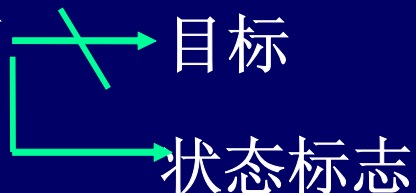
The diagram shows a green arrow pointing from the '目标' (target) part of the instruction format to the word '目标' (target), and another green arrow pointing from the same '目标' part to the words '状态标志' (status flags).

例: **XOR BL, 0FH;** **BL**高四位不变, 低四位取反

**XOR CX, CX;** **CX**清零

**XOR AX, BX;** 若结果全零, 说明**AX**与**BX**相等

## ⑤逻辑测试指令**TEST**

指令格式及功能: **TEST** 目标, 源; 目标  $\wedge$  源 

指令的功能是**将两个操作数按位相与**, 但结果不送回目标, 只影响状态标志, 影响情况同**AND**指令。

**TEST**指令**常用来检测操作数的某位是1还是0**。

例如: **TEST CL, 01H**; 此指令执行后, 若**ZF**为**1**, 则说明**CL**最低位为**0**, 否则**CL**最低位为**1**。

## 2. 移位类指令

- 将操作数移动一位或多位，分成逻辑移位和算术移位，分别具有左移或右移操作
- 移位指令的第一个操作数是指定的被移位的操作数，可以是寄存器或存储单元；
- 后一个操作数表示移位位数：
  - 该操作数为1，表示移动一位
  - 该操作数为CL，CL寄存器值表示移位位数（移位位数大于1只能CL表示）
- 按照移出的位设置进位标志CF，根据移位后的结果影响SF、ZF、PF



## (1) 逻辑左移指令SHL

**SHL** reg/mem,1/CL

演示

； **reg/mem**左移**1**或**CL**位

； 最低位补**0**，最高位进入**CF**



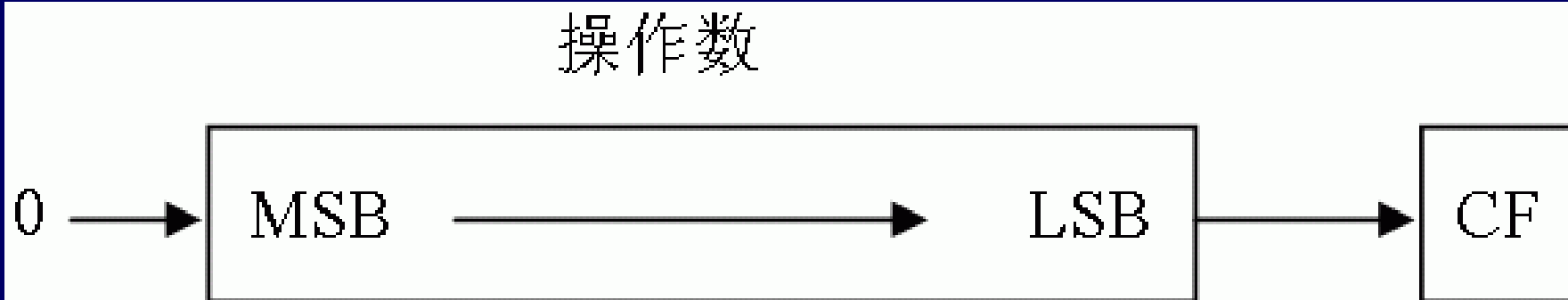
## (2) 逻辑右移指令SHR

**SHR** reg/mem, 1/CL

演示

； **reg/mem**右移**1/CL**位

； 最高位补**0**，最低位进入**CF**



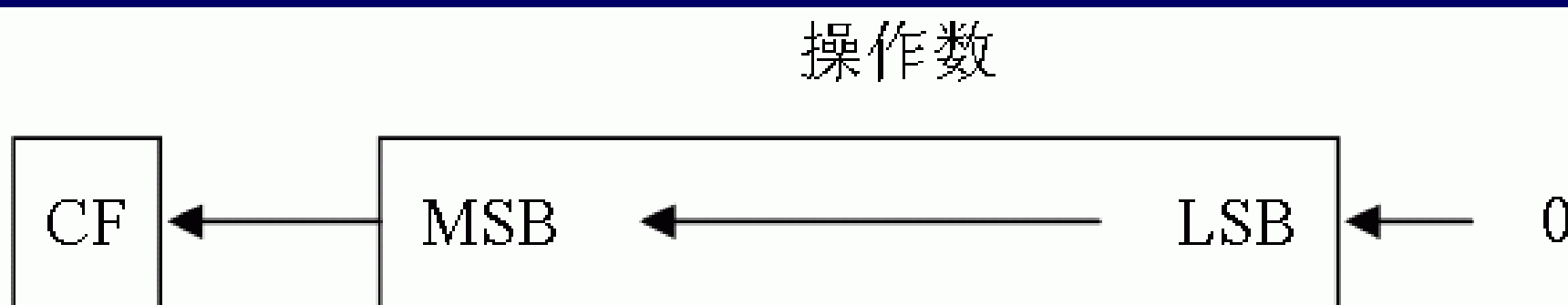
(b) 逻辑右移 SHR

### (3) 算术左移指令 **SAL**

**SAL** reg/mem,1/CL

演示

； 功能与**SHL**相同，同一条指令



(a) 逻辑/算术左移 SHL/SAL

#### (4) 算术右移指令 **SAR**

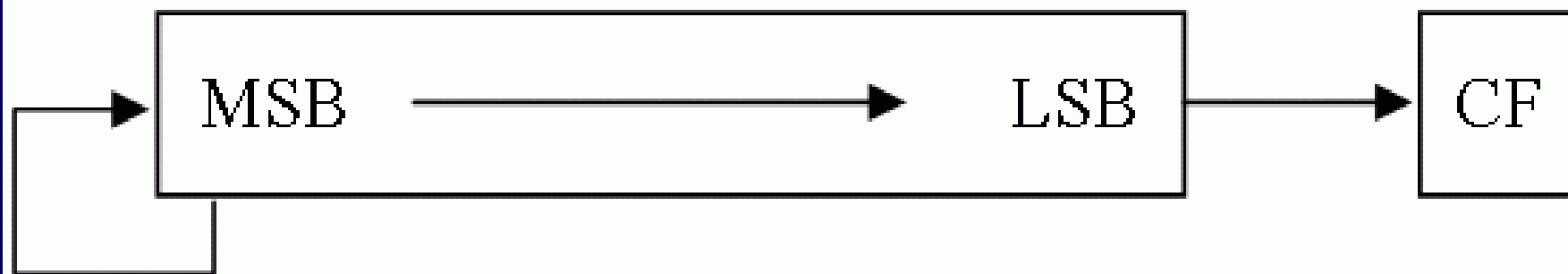
**SAR** reg/mem, 1/CL

演示

； **reg/mem** 右移 **1/CL** 位

； 最高位不变，最低位进入 **CF**

操作数



(c) 算术右移 SAR

# SHL和SAL指令的功能



# SHR指令的功能



# SAR指令的功能



**SHR**适用于无符号数，而**SAR**适用于带符号数。

这四条指令对标志位影响为：

- 1) **CF**总是等于目的操作数最后移出的那一位；移位以后按操作数结果影响**PF**、**SF**和**ZF**，**AF**无意义；
- 2) 在移位次数为**1**时，若移位完成后目的操作数的最高位与**CF**不相等，则溢出标志**OF=1**，否则**OF=0**。

因此**OF**标志用以说明移位后的符号位与移位前的是否相同（若相同，则**OF=0**）。

若移位次数不为**1**，**OF**无定义！！



### 3. 循环移位类指令

**8086/8088**也有四条循环移位类指令。

其格式及功能说明如下：

循环左移    **ROL REG/MEM, CNT;**    **ROL**指令功能



循环右移 **ROR** REG/MEM, CNT; **ROR**指令功能



帶CF循环左移 **RCL** REG/MEM, CNT; **RCL**指令功能



带CF循环右移 **RCR** REG/MEM, CNT; **RCR**指令功能



## 五、串操作指令

指令功能是完成各种基本的**字节串或字串**的操作。

所谓字节串或字串就是字节或字的序列，按顺序存储在内存的多个单元中。

串操作指令，均可以在指令的前面加一个**重复前缀**使它们重复执行。

这组指令有以下几条特点。

(1) 源操作数段地址由**DS**决定，段内偏移地址由**SI**确定；  
目标操作数段地址由**ES**决定，段内偏移地址由**DI**确定。

(2) 每一条串操作指令执行后会自动修改地址指针**SI**、**DI**。  
按增量还是按减量修改地址，取决于方向标志**DF**。

**DF=0**: 字节操作指令执行后，**SI+1→SI**，**DI+1→DI**

字操作指令执行后，**SI+2→SI**，**DI+2→DI**；

**DF=1**: **SI-1（或2）**，**DI-1（或2）**

**(3) CX:** 数据项的个数。

在执行带**重复前缀**指令的字符串指令时，每执行一次，**CX**的内容自动减**1**。当**CX**中的内容减到零时，停止执行字符串指令。所以在重复执行字符串指令前，必须先给**CX**赋值。

**(4)** 重复的字符串指令可以被中断。



串操作指令包括:

**MOVSB ; SCASB ; LODSB**  
**CMPSB ; STOSB**

字节操作

**MOVSW ; SCASW ; LODSW;**  
**CMPSW ; STOSW**

字操作

**8086/8088**指令系统有五条字符串操作指令，三条重复前缀。

1. 字符串传送指令格式：

**MOVSB** ; 字节传送

$[DS: SI] \longrightarrow [ES: DI]$

$SI=SI+1, DI=DI+1 \quad (DF=0)$

$SI=SI-1, DI=DI-1 \quad (DF=1)$

**MOVSW** ; 字传送

$[DS: SI] \longrightarrow [ES: DI]$

$SI=SI+2, DI=DI+2 \quad (DF=0)$

$SI=SI-2, DI=DI-2 \quad (DF=1)$

例：编程将内存数据区地址自**12000H**开始的**20**个字节数据，传送到地址自**16000H**开始的存储区。

分析：字符串传送

源串：            **DS: 1000H**            **SI=2000H**

目标串：        **ES: 1000H**            **DI=6000H**

数据个数：    **20**                    **CX=20**

**ES=DS**

程序片段如下：

```
MOV AX, 1000H
MOV DS, AX
MOV ES, DS
MOV SI, 2000H
MOV DI, 6000H
MOV CX, 20
CLD      ;
AGAIN: MOVSB
      DEC CX
      JNZ AGAIN
```

置初值

置方向标志位  
**DF=0**

## 2. 字符串比较指令格式:

**CMPSB** ; 字节串比较

$[ES: DI] - [DS: SI] \xrightarrow{\text{FLAGS}}$

$SI=SI+1, DI=DI+1$  (DF=0)

$SI=SI-1, DI=DI-1$  (DF=1)

**CMPSW** ; 字串比较

$[ES: DI] - [DS: SI] \longrightarrow \text{FLAGS}$

$SI=SI+2, DI=DI+2$  (DF=0)

$SI=SI-2, DI=DI-2$  (DF=1)

### 3. 字符串搜索

指令格式: **SCAS** 目的串

一般形式: 搜索关键字放在  
**AL或AX**

**SCASB** ; 字节串搜索

**SCASW** ; 字串搜索

AL-[ES: DI]  FLAGS

SI=SI+1 , DI=DI+1 (DF=0)

SI=SI-1 , DI=DI-1 (DF=1)

AX-[ES: DI]  FLAGS

SI=SI+2 , DI=DI+2 (DF=0)

SI=SI-2 , DI=DI-2 (DF=1)

如何判断是否搜索到某个字符?

**ZF=?**

**ZF=0**, 没找到!!

**ZF=1**, 找到!!

#### 4. 字符串装入

指令格式: LODS 源串

一般形式:

**LODSB** ; 字节串装入

**LODSW** ; 字串装入

$[DS: SI] \longrightarrow AL$

$SI=SI+1, DI=DI+1$  (DF=0)

$SI=SI-1, DI=DI-1$  (DF=1)

$[DS: SI] \longrightarrow AX$

$SI=SI+2, DI=DI+2$  (DF=0)

$SI=SI-2, DI=DI-2$  (DF=1)

此指令属传送指令，对状态标志无影响，当需要将一批数据不断送往**AL** (**AX**) 中处理时，用此指令非常方便。

## 5. 字符串转储

指令格式: **STOS**      目的串;  
一般形式:

**STOSB**    ; 字节转储

**STOSW**    ; 字转储

$[ES: DI] \leftarrow AL$   
 $DI = DI + 1 \quad (DF = 0)$   
 $DI = DI - 1 \quad (DF = 1)$

$[ES: DI] \leftarrow AX$   
 $DI = DI + 2 \quad (DF = 0)$   
 $DI = DI - 2 \quad (DF = 1)$

此指令对标志  
位无影响。



## 6. 重复前缀指令

重复前缀指令用来控制紧跟其后的字符串指令是否重复执行。此类指令共有3条。指令助记符及功能说明如下：

<b>REP</b>	重复执行其后的字符串指令，直到CX=0
<b>REPE/ERPZ</b>	当相等/为零时重复执行其后的字符串指令
<b>REPNE/REPNZ</b>	当不相等/不为零时重复执行其后的字符串指令

REP常与MOVS及STOS指令联合使用，可以将由CX中内容规定元素个数的一串字符进行传送或转储。

REPE（相等时重复）和REPZ（等于零时重复）两个重复前缀指令实际上是相同的。它们与CMPS及SCAS指令联合使用，根据零标志ZF（由CMPS及SCAS指令影响）及CX内容决定是否重复。若**ZF=1**，且**CX≠0**则重复；否则停止字符串操作，转而执行下一条指令。

REPNE（不相等时重复）和REPNZ（不等于0时重复）两个重复前缀指令意义相同。它们也与CMPS及SCAS指令联合使用，根据CMPS及SCAS所设置的零标志ZF及CX的内容，决定是否重复。若**ZF=0**且**CX≠0**，则重复执行其后的字符串指令，否则停止重复，转而执行下一条指令。

**例5.13** 若要对内存某一缓冲区清零，缓冲区地址为2000H—2100H，缓冲区长度为100个字节。编程实现：

```
CLD  
MOV DI, 2000H  
MOV CX, 100  
MOV AL, 00  
REP STOSB  
:
```

重复把AL=00送到DI所指的单元，并自动修正DI和CX内容，直至CX=0为止。

## 六、控制转移类指令

**8086/8088**指令系统有**4**种控制转移类指令，分别为：转移指令、循环指令、子程序调用及返回指令和中断及中断返回指令。

### 1. 转移指令

转移指令又分成**无条件转移**与**有条件转移**两种类型的指令。

#### (1) 无条件转移指令**JMP**

指令格式：**JMP   XUL**

指令功能：程序无条件转移到 **XUL**(语句标号 )处执行

```
    jmp xul
```


```
    .....
```

```
xul:  mov ax, bx
```

## (2) 条件转移指令

根据对状态标志位测试的结果决定是否将程序转移到新的地址，当测试结果满足指令的条件时，程序转移到目标地址；否则不发生转移，依然按原顺序向下执行。

所有条件转移指令的目标地址必须在当前代码段内，相对位移只能在**-128——+127**字节范围内。



实际应用时，当要求转移的范围超过上述规定，必须借助于**JMP**指令。

**8086/8088**指令系统中，共有**18**条条件转移指令。

## 两个无符号数比较后根据其比较结果形成的4个条件转移指令

	助记符	测试条件	转移条件	
对 无 符 号 数	<b>JA/JNBE</b>	<b>CF=0</b>	此类指 令一般	若目的操作数 > 源 操作数则转移
	<b>JAE/JNB</b>	<b>CF=0</b> 或 <b>ZF=1</b>	用于比 较指令	若目的操作数 $\geq$ 源 操作数则转移
	<b>JB/JNAE</b>	<b>CF=1</b>	及减法 指令之	若目的操作数 < 源 操作数则转移
	<b>JBE/JNA</b>	<b>CF=1</b> 或 <b>ZF=1</b>	后	若目的操作数 $\leq$ 源 操作数则转移

## 两个带符号数比较后根据其比较结果形成的4条条件转移指令

对 带 符 号 数	<b>JG/JNLE</b>	<b>SF <math>\oplus</math> OFVZF=0</b>	同样 根据	若目的操作数 > 源 操作数则转移
	<b>JGE/JNL</b>	<b>SF <math>\oplus</math> OF=0</b>	二个 数比	若目的操作数 $\geq$ 源 操作数则转移
	<b>JL/JNGE</b>	<b>SF <math>\oplus</math> OF=1</b>	较或 相减	若目的操作数 < 源操作数则转移
	<b>JLE/JNG</b>	<b>SF <math>\oplus</math> OFVZF=1</b>	的结 果	若目的操作数 $\leq$ 源 操作数则转移

根据**CF**、**ZF**、**SF**、**OF**、**PF**的状态形成的**10**条条件转移指令

<b>JE/JZ</b>	<b>ZF=1</b>	当结果为零时，转移
<b>JNE/JNZ</b>	<b>ZF=0</b>	当结果不为零时，转移
<b>JC</b>	<b>CF=1</b>	有借（进）位，转移
<b>JNC</b>	<b>CF=0</b>	无进（借）位，转移
<b>JO</b>	<b>OF=1</b>	有溢出（带符号数）转移
<b>JNO</b>	<b>OF=0</b>	无溢出（带符号数）转移
<b>JP/JPE</b>	<b>PF=1</b>	结果为偶数个 <b>1</b> 转移
<b>JNP/JPO</b>	<b>PF=0</b>	结果为奇数个 <b>1</b> 转移
<b>JS</b>	<b>SF=1</b>	最高位为 <b>1</b> 转移
<b>JNS</b>	<b>SF=0</b>	最高位为 <b>0</b> 转移



**例5.4** 设**AL**、**BL**中分别存有无符号数，找出大值存入数据区**100AH**单元。

分析：要判断**AL**、**BL**中哪个值大，首先要运用比较指令将二数作减法（注意结果并不送回），由于为无符号数，故可从**CF**标志判断其大小，**CF=1**，说明作比较（相减）时有借位，因而被减数<减数；**CF=0**，说明相减时无错位，被减数 $\geq$ 减数。

程序片断如下：

```
CMP    AL, BL  
JNC    NEXT      ; CF=0, 转NEXT  
XCHG   AL, BL    ; 交换AL, BL  
NEXT:  MOV [100AH], AL
```

## 2. 循环指令

循环指令是一组特殊的条件转移指令，具有下述几个特点：

- ①循环指令对**CX**寄存器的内容进行测试，根据**CX**中内容是否为零；或者根据**CX**中的内容是否为零以及零标志**ZF**的状态作为转移条件。
- ②采用段内相对寻址方式，即满足条件时，将实现程序转移。程序转移的范围，只能在**-128~+127**字节内。
- ③除**JCXZ**指令外，其余循环指令执行时，都先自动修改**CX**内容（**CX-1→CX**），再根据**CX**内容及**ZF**状态来决定是否循环转移。

助记符	测试内容	指令功能	示例
<b>LOOP</b>	<b>CX</b> 内容	<b>CX-1→CX</b> 若 <b>CX≠0</b> 则循环 (转移)	<b>LOOP MULT</b>
<b>LOOPZ/LOOPE</b>	<b>CX</b> 内容 <b>ZF</b> 状态	<b>CX-1→CX</b> 若 <b>CX≠0</b> 且 <b>ZF=1</b> 则循环 (转移)	<b>LOOPZ MULT</b>
<b>LOOPNZ/LOOPNE</b>	<b>CX</b> 内容 <b>ZF</b> 状态	<b>CX-1→CX</b> 若 <b>CX≠0</b> 且 <b>ZF=0</b> 则循环 (转移)	<b>LOOPNZ XLT</b>
<b>JCXZ</b>	<b>CX</b> 内容	若 <b>CX=0</b> 则转移	<b>JCXZ NEXT</b>

**例5.15** 设有**100**个无符号字节数据，存放在自**1000H**开始的数据区中，试编程求取其中零的个数，并存于**3000H**单元中。

分析：在100个数据中找零的个数，方法如下：

把100个数据一一取出，判其是否为零；

典型的计数循环

步骤如下：

1) 设循环初态：{ CX=100  
SI=1000H  
DL=0

2) 取出的数存于AL，

MOV AL, [SI]

3) 判断是否为0，设标志位：AND AL, AL

4) 若 ZF=1，则为零，DL=DL+1

6) 将DL内容存于3000H中

若ZF=0，则不为零，DL不加1

5) CX=CX-1，没减到零，返回2)

程序段如下：

	<b>MOV SI, 1000H</b>	； 首地址→SI	
	<b>MOV DL, 00H</b>	； 清DL，放零的个数	
	<b>MOV CX, 100</b>	； 数据长度→CX	
<b>LOOP1:</b>	<b>MOV AL, [SI]</b>	； 取一个数→AL	
	<b>INC SI</b>	； 修改地址指针	
	<b>AND AL, AL</b>	； 设标志	
	<b>JNZ NEXT</b>	； 若数不为零，转移	
	<b>INC DL</b>	； 否则，数为零DL+1→DL	
<b>NEXT:</b>	<b>DEC CX</b>	； 计数器减1	} <b>LOOP LOOP1</b>
	<b>JNZ LOOP1</b>	； 不为零重复	
	<b>MOV BX, 3000H</b>	； 将零的个数送入	
	<b>MOV [BX], DL</b>	； 3000H单元	

### 3. 子程序调用及返回指令

**8086/8088**系统把重复执行的一系列操作指令编成一独立的程序段，并定义名字，称为子程序。

子程序可被其它程序多次调用，可以用**CALL**指令来实现。

#### 1)子程序调用指令

常用形式： **CALL SUB1**

SUB1 -----子程序名

**表5-11** 调用与返回指令的助记符及功能

指令助记符	操作数	指令功能
CALL	disp16	段内相对调用 ①SP-2→SP IP →堆栈 ②IP+disp16 →IP
CALL	REG/MEM	段内间接调用 ①SP-2→SP IP →堆栈 ②(EA) →IP
CALL	Addr32 [FAR]	段间直接调用 ①SP-2→SP CS →堆栈 ②SP-2→SP IP →堆栈 ③addr 偏移地址→IP ④addr 段地址→CS
CALL	MEM [FAR]	段间间接调用 ①SP-2→SP CS →堆栈 ②SP-2→SP IP →堆栈 ③(EA) →SP ④(EA+2) →CS

RET		段内子程序返回 ①堆栈→IP ②SP+2→SP
RET	disp16	段内子程序返回并修改 SP ①堆栈→IP ②SP+2→SP ③SP+disp→SP
RETF		段间子程序返回 ①堆栈→IP    SP+2→SP ②堆栈→CS    SP+2→SP
RETF	disp16	段间子程序返回并修改 SP ①堆栈→IP    SP+2→SP ②堆栈→CS    SP+2→SP ③SP+disp→SP



#### 4、中断及中断返回指令

8086/8088有三种中断操作指令：**INT n**、**INTO**、**IRET**

**INT n**; **n**——中断类型号，**0**——**255**

**INTO**; ——溢出中断，**OF=1**启动某类型中断，否则无操作

**IRET**; ——中断返回

## 六 处理器控制类指令

处理器控制类指令可分为三组。

一组用于修改标志位；

另一组主要用于使8086/8088 CPU与外部事件同步；

第三组仅一条空操作指令。

如表3-2所示。

表3-2 处理器控制类指令

类型	操作码助记符	操作功能
标志操作	<b>CLC</b>	<b>0→CF</b> ; 进位标志清零
	<b>STC</b>	<b>1→CF</b> ; 进位标志置 1
	<b>CMC</b>	<b>CF→CF</b> ; 进位标志取反
	<b>CLD</b>	<b>0→DF</b> ; 方向标志清零
	<b>STD</b>	<b>1→DF</b> ; 方向标志置 1
	<b>CLI</b>	<b>0→IF</b> ; 中断标志清零
	<b>STI</b>	<b>1→IF</b> ; 中断标志置 1
外部同步	<b>HLT</b>	暂停
	<b>WAIT</b>	等待
	<b>ESC</b>	交权
	<b>LOCK</b>	总线封锁
空操作	<b>NOP</b>	空操作

## 外部事件同步类指令

(1) 处理器暂停指令HLT

(2) 处理器脱离指令ESC

(3) 处理器等待指令WAIT

(4) 总线锁定指令LOCK

(5) 空操作指令NOP

### 学习重点:

- 1) 掌握操作数的各种寻址方式的概念
- 2) 掌握各种指令的助记符、功能及应用方法
- 3) 会用指令编写程序
- 4) 会分析程序段的运行结果