

第二章 8088/8086微处理器的指令系统

2.1 8088/8086的寻址方式

2.2 8088/8086的指令系统



2.1 8088/8086的寻址方式

- **指令**是微处理器执行某种操作的命令，微处理器全部指令的集合称为指令系统。
- 指令有两种书写格式：机器指令和符号指令。
- 符号指令是用规定的助记符和规定的书写格式书写的指令。符号指令的书写格式为：

操作码助记符 操作数助记符

MOV AL, 1

10110000 00000001



8086/8088指令格式

- 指令格式

操作码	操作数	...	操作数
-----	-----	-----	-----

例

ADD AL, 10H

(1) 操作码

指明CPU要执行什么样的操作。

是一条指令必不可少的部分，用助记符表示。

按功能

数据传送
算术运算
逻辑运算
串操作
控制转移
处理机控制

(2) 操作数

指明参与操作的数据或数据所在的地方。

了解操作数的来源、个数、类型。

8.16.32 [DX, AX]
字节型 双字型

□ 操作数个数

按指令格式中，操作数个数的多少分为四类：

无操作数：指令只有一个操作码，没有操作数

单操作数：指令中给出一个操作数

双操作数：指令中给出两个操作数。

三操作数：指令中给出三个操作数。



① 无操作数： 指令只有一个操作码，没有操作数。

有两种可能：

▲ 有些操作不需要操作数。

如 HLT, NOP等处理机控制指令。

▲ 操作数隐含在指令中。

如 AAA , DAA等调整指令。



② 单操作数： 指令中给出一个操作数。

有两种可能：

▲有些操作只需要一个操作数

如 INC AL ; (AL) ← (AL) + 1

▲有些操作将另一个操作数隐含在指令中


如 MUL BL ; (AX) ← (AL) × (BL)



③ 双操作数： 指令中给出两个操作数。

如 MOV AL, BL ; $(AL) \leftarrow (BL)$
如 ADD AL, BL ; $(AL) \leftarrow (AL) + (BL)$

目的操作数 源操作数



The diagram shows two arrows pointing from the labels '目的操作数' (Destination Operand) and '源操作数' (Source Operand) to the 'AL' and 'BL' registers in the 'ADD AL, BL' instruction. The 'AL' register is the destination operand, and the 'BL' register is the source operand.

操作后的结果通常存放在目的操作数中。

④ 三操作数： 指令中给出三个操作数。

如 **IMUL BX, DX, 6** ; $(BX) \leftarrow (DX) * 6$

目的操作数 **源操作数** **立即数**

操作后的结果通常存放在目的操作数中。



操作数的来源

寻址方式：寻找指令中操作数地址的方式。

操作数有三种可能的存放方式：

- 直接包含在指令中

立即数

立即数寻址

- 包含在某个寄存器中

寄存器操作数

寄存器寻址

- 在内存中

存储器操作数（内存操作数）

存储器寻址

内存实际地址由两部分组成：存储单元所在段的基地址/段内偏移地址（偏移量）

MOV ES:[3000H], AL

段内偏移地址可以由如下四个部分组成（称为偏移地址四元素）：

- 基址寄存器内容
- 变址寄存器内容
- 比例因子（**Pentium**处理器才有,8086为1）
- 位移量

由四元素组合形成的**偏移地址**称为有效地址EA：

EA=基址+(变址×比例因子)+位移量

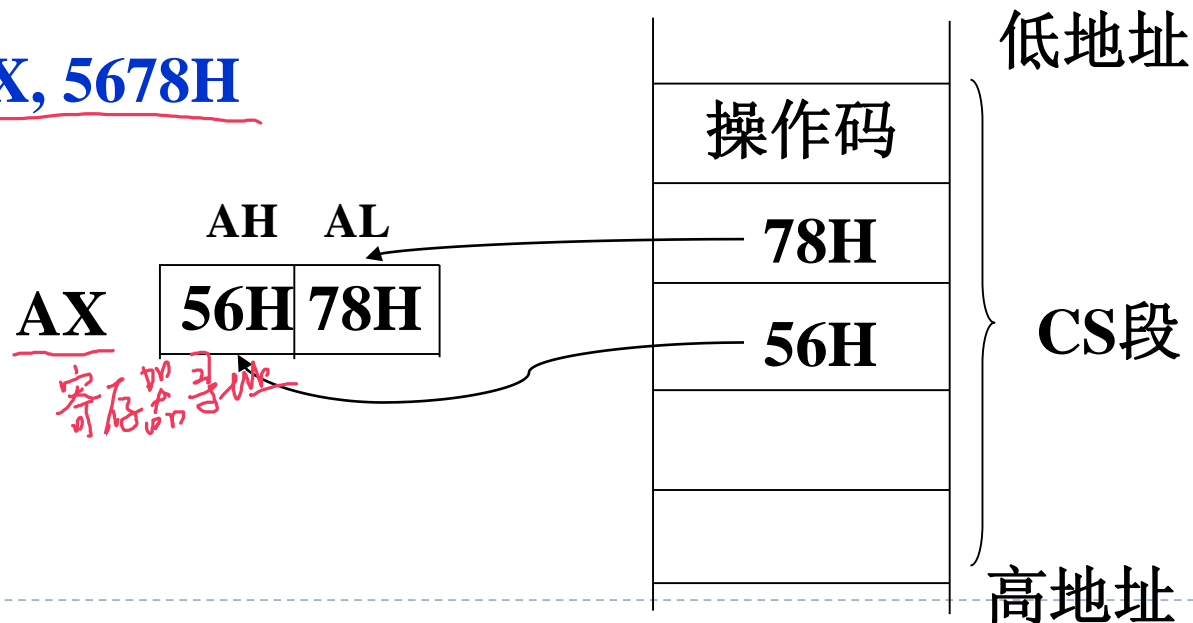
► **EA_{8088/8086}=基址+变址+位移量**

由四元素可组合出多种寻址方式。8088/8086共有7种寻址方式

1. 立即寻址

操作数作为立即数直接存在指令中，紧跟在操作码后，放在代码段。立即寻址方式只能出现在源操作数的寻址中，目的地操作数不能采用此种方式。

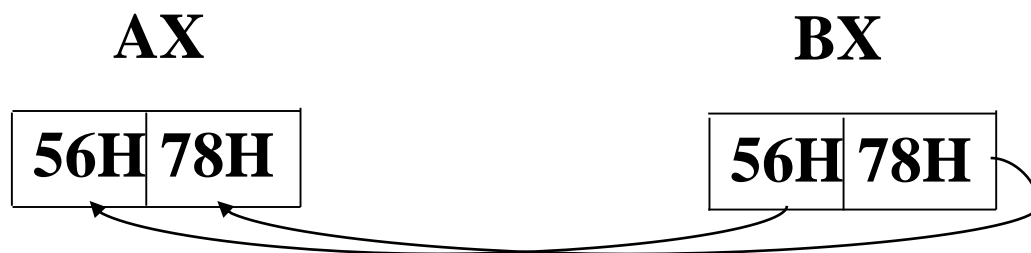
MOV AX, 5678H



2. 寄存器寻址

这种寻址方式的操作数在CPU的内部寄存器中。

MOV AX, BX



寄存器寻址由于无需从存储器中取操作数，故执行速度快。

3. 直接寻址

指令中的操作数部分直接给出操作数的有效地址EA，操作数是16位整数，操作数默认在DS段中。

MOV AX, [3000H]

MOV AX, 3000H → 立即寻址

DS

5000 0

+

3000

53000

50000

53000

AX

AH AL
12H 34H

操作码

00H

30H

34H

12H

低地址

CS段

DS段

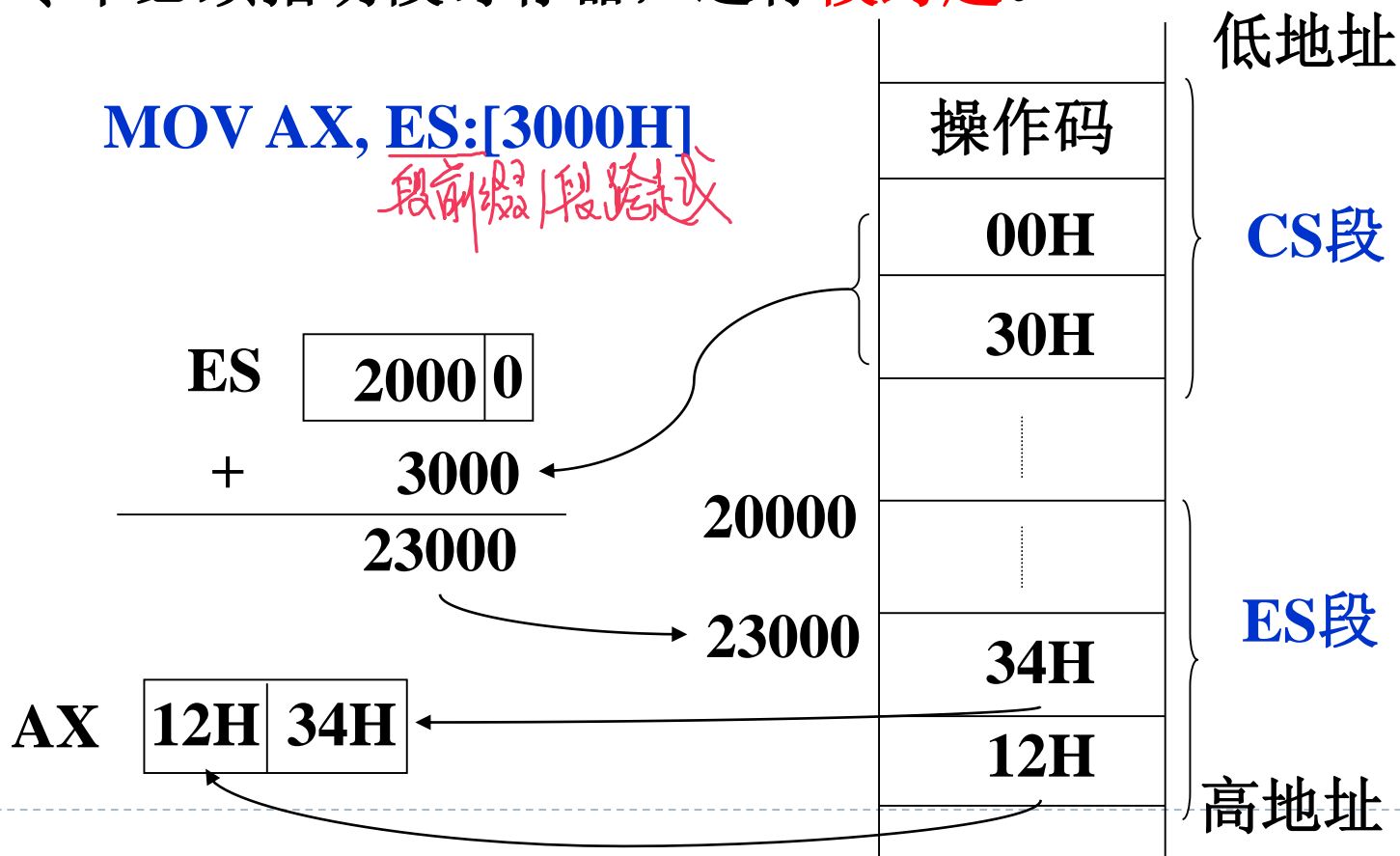
高地址

*低地址单元 - 低字节
高地址单元 - 高字节*

*AX = 1234H (v)
AX = 3412H (x)*

① 类型一致原则 ② 高对高, 低对低原则

如果操作数在DS以外的其他段（CS,SS,ES）中，指令中必须指明段寄存器，这称**段跨越**。



4. 寄存器间接寻址

操作数地址的有效地址EA存放在**寄存器**中
16位偏移地址放在SI,DI,BP,BX中

以**SI,DI, BX**间接寻址，默认操作数在**DS**段中

MOV AX, [SI]

以**BP**间接寻址，默认操作数在**SS**段中

MOV AX, [BP]

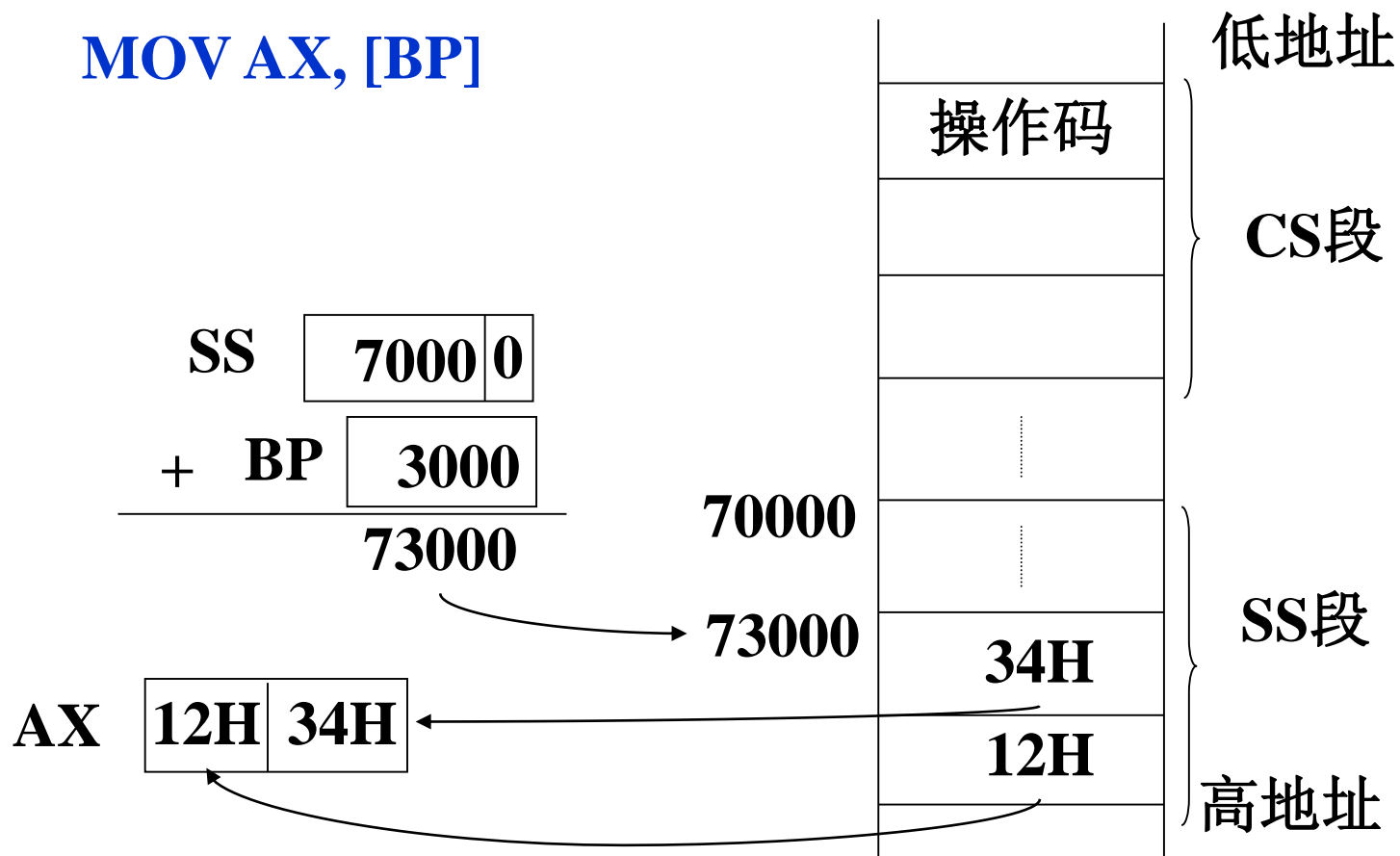
SP

DX (I/O)

数据寄存器

放偏移地址

MOV AX, [BP]



5. 基址寻址

$EA = [\text{基址寄存器}] + \text{位移量}$
方括号表示寄存器中的内容是偏移地址。

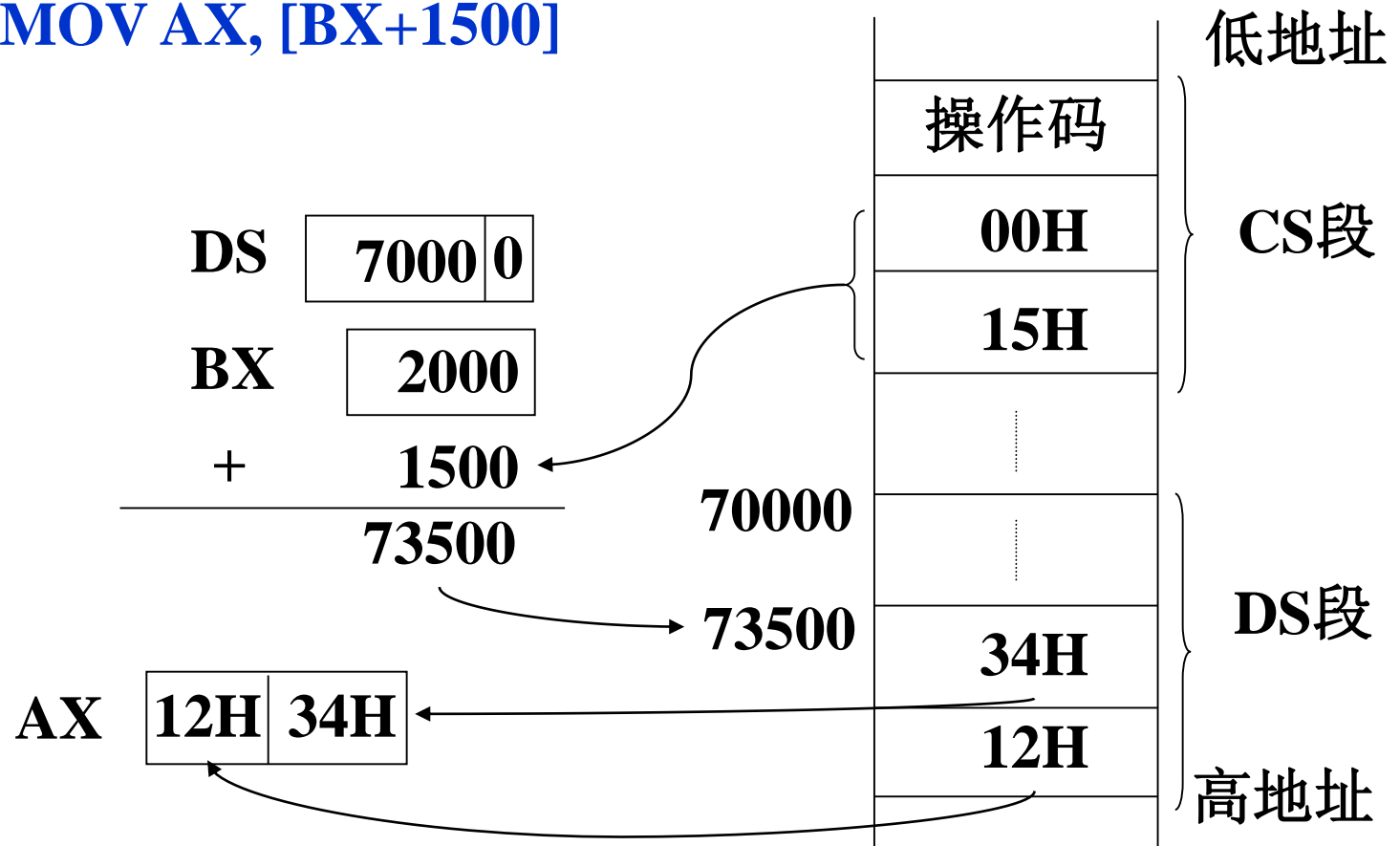
BP, BX 为基址寄存器

BX, DS 为默认段寄存器

BP, SS 为默认段寄存器



MOV AX, [BX+1500]



6. 变址寻址

$EA = [\text{变址寄存器}] + \text{位移量}$

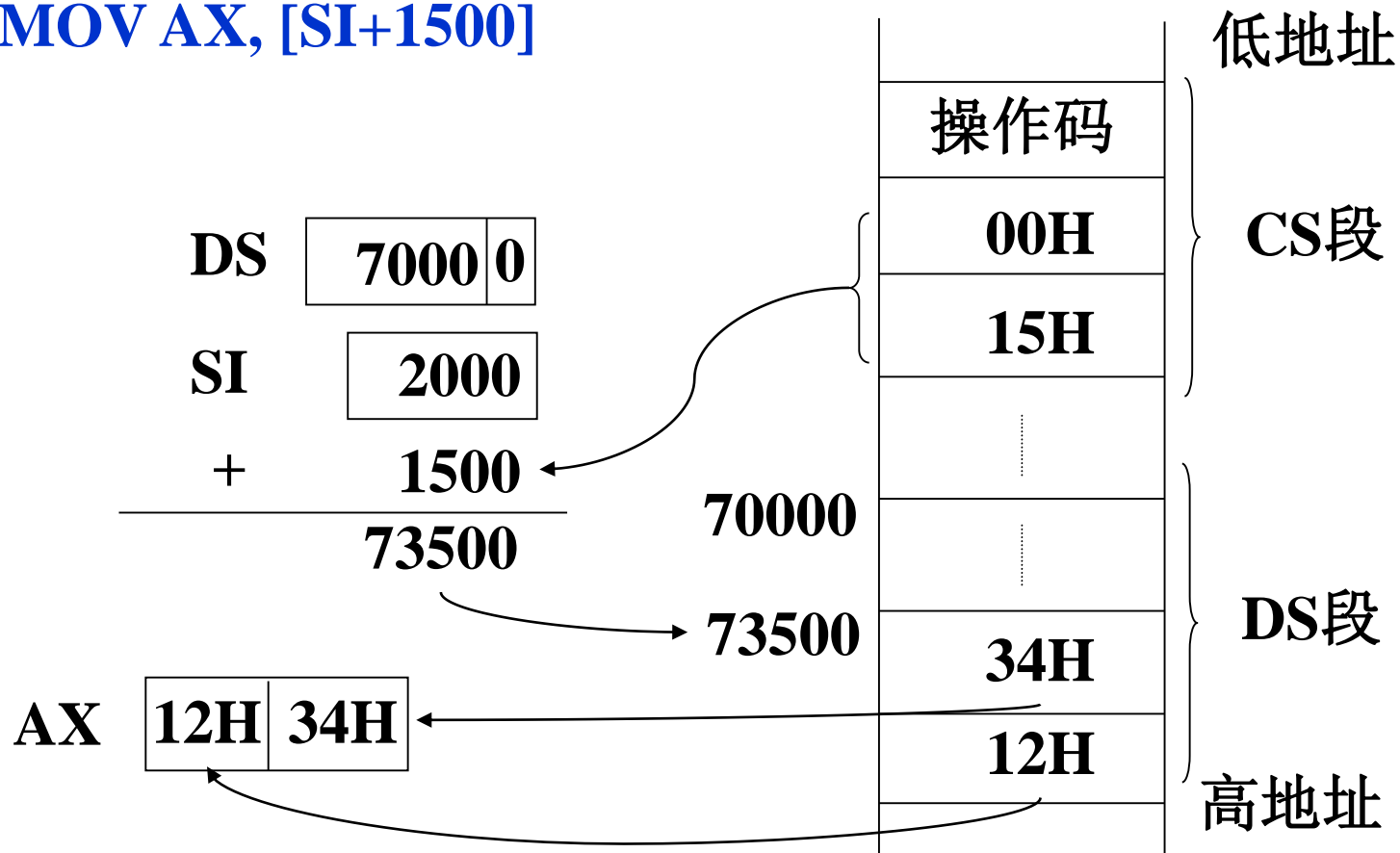
SI, DI 为变址寄存器, DS 为默认段寄存器

MOV AH, [SI+5]

mov AH, [SI] + 5

变址寻址适用于对一维数组的元素进行操作。

MOV AX, [SI+1500]



7. 基址加变址寻址

$EA = [\text{基址寄存器}] + [\text{变址寄存器}] + \text{偏移量}$

通常把BX和BP作为基址寄存器，把SI和DI作为变址寄存器，一共有四种组合。

BX DS为默认段寄存器

BP SS为默认段寄存器

MOV AX, COUNT[BX+^{DI}SI]

MOV AX, [BP+SI]

BX DX
BP DI

$[BP] + [SI] + 7$

$[BP + SI] + 7$

$[BP + SI + 7]$

基址加变址寻址主要用于二维数组操作和二重循环

练习题

现有 (DS)=2000H , (BX)=0100H , (SI)=0002H ,
(20100H)=12H , (20101H)=34H , (20102H)=56H ,
(20103H)=78H , (21200H)=2AH , (21201H)=4CH ,
(21202H)=B7H, (21203H)=65H, 试说明下列各条指令执行完后AX寄存器的内容。

- ▶ (1) MOV AX, 1200H
- ▶ (2) MOV AX, BX
- ▶ (3) MOV AX, [1200H]
- ▶ (4) MOV AX, [BX]
- ▶ (5) MOV AX, 1100[BX]
- ▶ (6) MOV AX, [BX][SI]
- ▶ (7) MOV AX, 1100[BX][SI]

2.2 8088/8086的指令系统

8088/8086指令系统（共130多种）可分为以下6组：

1. 数据传送（Data transfer）指令 (14)
2. 算术运算（Arithmetic）指令 ($3+5+2+2+2+6=20$)
3. 逻辑运算（Logic）指令 ($4+8=12$)
4. 串操作（String manipulation）指令 ($5*2=10$)
5. 控制转移（Control transfer）指令 ($1+8+10+2+2+2=23$)
6. 处理器控制（Processor control）指令



注意一： 学习指令的要点

从以下几个方面来掌握一条指令：

- 指令的助记符
- 指令的格式：操作数的个数、类型 (B, W, DW) 8 16 32
- 执行的操作：指令执行后的结果 字节 双字节

包括：哪些寄存器、内存单元的值发生了变化

对标志位有无影响，哪些受影响

- 特殊要求及注意事项

只介绍常用的指令，其他需要时可自学。



注意二： 利用DEBUG学习指令系统（示例）

编程完成 B5h + 8Fh = ? 学习加法ADD指令及其对状态标志位的影响。

D:\>DEBUG ↵
- A ↵ ;汇编指令
0AF8:0100 MOV AL,B5 ↵
0AF8:0102 ADD AL, 8F ↵
0AF8:0104 ↵
- R ↵ ;显示指令执行前各寄存器的值
AX=0000 BX=0000 CX=0000 DX=0000 、 、 、 、 、 、
CS=0AF8 IP=0100
NV UP EI PL NZ NA PO NC
- T=100 2↵ ;执行指令，查看结果
AX=0044 BX=0000 CX=0000 DX=0000 、 、 、 、 、 、
CS=0AF8 IP=0104
OV UP EI PL NZ AC PE CY
0AF8:0104 2080FC01 AND [BX+SI+01FC], AL
- ▶

10110101
+ 10001111
进位 1 111111
01000100

注意三： 书写指令注意事项：

- 不区分字母的大小写。

下列写法表示同一条指令：

MOV AX, 1ABDH

mov ax, 1abdh

- 不添加指令系统没有的指令，即不自创助记符。

将 MOV AL, 0 写成 MOVE AL, 0

JMP lable 写成 JUMP lable



注意四：注意操作数的范围

80H ~ 7FH

对字节操作指令 0 ~ FFH 0 ~ 255

对字操作指令 0 ~ FFFFH 0 ~ 65535

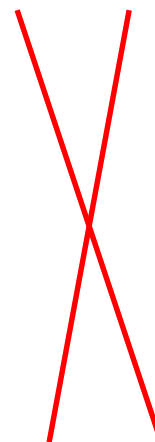
1000 0000 0000 0000B ~ 0111 1111 1111 1111B

MOV AL, 260

MOV AX, 70000

MOV AL, 1FFH

MOV AL, 2ABCDH



注意五：注意操作数的格式

- 对无操作数指令，不添加操作数。

STC AL X

- 对单操作数指令，操作数不能是立即数。

IMUL 6 X



● 对双操作数指令

① 不能两个同为存储器操作数

MOV [DI], [SI] X

② 目的操作数不能是立即数

ADD 3, AL X

③ 两个操作数的类型应相同

SUB AX, BL X

若 value 定义为字类型存储器变量：

MOV CL, value [BX] X

1. 一致性原则
2. 高对高低对低原则
3. 类型必须确定 $6 = 06 = 0006$

● 内存操作数的类型属性应明确。

MOV [BX], 0

类型都不确定

MOV byte ptr [BX], 0

字节



MOV word ptr [BX], 0

字



MOV [BX], AL



MOV [BX], AX



-
-
- A、B、C、D、E、F开头的十六进制数前面加0，与H结尾的标识符区别。

如 寄存器名： AH、BH、CH、DH

变量名： abcdH 等

例

mov AL, 0AH



mov AL, AH



mov BX, 0abcdH



注意六：个别寄存器的特殊性

- CS 和 IP的值只在**控制转移指令**中修改。
- 对非控制转移指令，取完指令后IP值自动 指向下条指令。
- 段寄存器CS的值，只在MOV、PUSH中可作操作数，且这两条指令执行结果不改变CS值。

MOV AX, CS

PUSH CS

- IP、PSW两个寄存器不作为操作数在指令中出现。

mov IP, 1234H

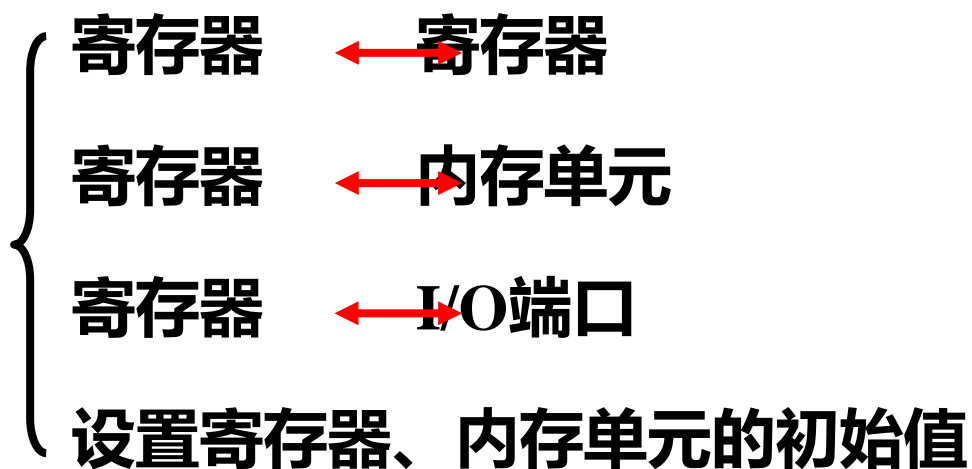
mov PSW, 0F0FH

X

- PSW状态寄存器的值由指令执行后确定，
不同的指令对各标志的影响不同。

一、数据传送指令

- ▲ 数据传送是最基本、最重要的一种操作
实际程序中，使用的比例最高



执行后不影响标志位，源操作数不变，有四类：

通用传送指令

地址传送指令

累加器专用传送指令

标志传送指令

（一）通用数据传送指令

包括MOV PUSH POP XCHG

- **MOV 目标操作数（OPRD1），源操作数(OPRD2)**

1) 源操作数可以是8/16位的立即数、寄存器操作数、内存操作数。目标操作数不允许为立即数，其余同源操作数。源、目不能同时为内存操作数。



2) 源、目操作数类型必须匹配，属性明确

MOV BYTE PTR [BX], 12H

3) 不能向段寄存器写立即数

MOV DS, 2000 (错误) **MOV AX, 2000**

MOV DS, AX

4) 以CS和IP为目的操作数的一切传送指令都是非法的



► 练习：判断指令对错

MOV AL, BL ✓

MOV DS, AX ✓

MOV CX, [1000] ✓

MOV [SI], 4050 ✗ 都不

MOV CS, AX ✗ CS不能赋值

MOV [SI], [1000] ✗ 寄存器操作数
都是

例 编程将CL寄存器的内容传送到200:100H单元中。

MOV [200:100H], CL

X

编程1:

MOV AX, 200H

MOV DS, AX

; (DS) = 200H

MOV [100H], CL

; (02100H) = (CL)

编程2:

MOV AX, 200H

MOV DS, AX

; (DS) = 200H

MOV DI, 100H

; (DI) = 100H

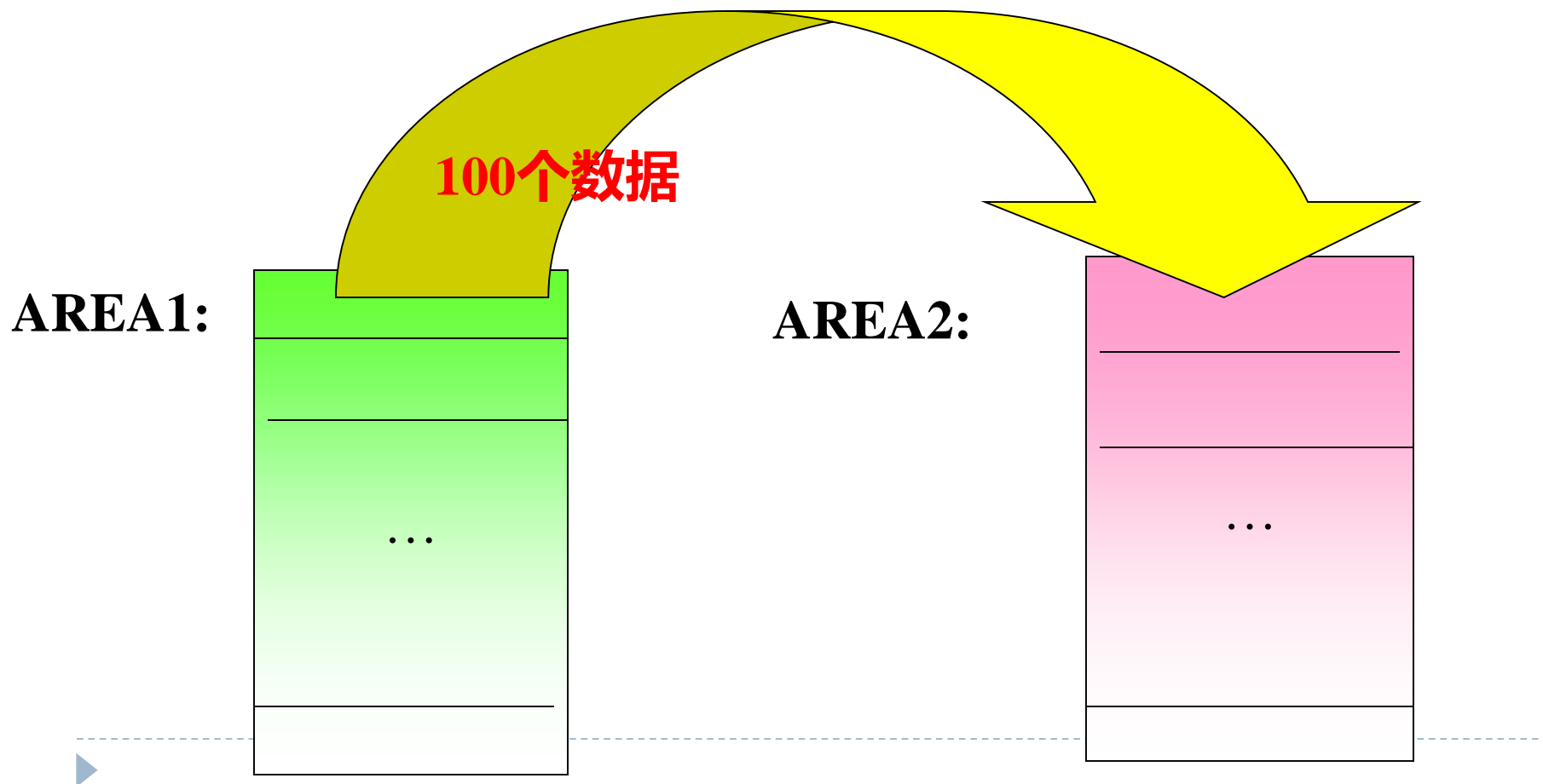
MOV [DI], CL

; (02100H) = (CL)

◆ MOV指令应用

例：

实现将AREA1开始的100个数据传送到AREA2开始的单元。



分析题意：

①可以用200条MOV指令来完成100个数据传送，
指令操作重复，每个数据传送后的地址是变化的。

② 可以利用循环，
但每循环一次要修改地址（源地址和目的地址），
必须把地址放在寄存器当中，用寄存器间接寻址来寻找操作数。



得到如下程序：

...

MOV SI, **OFFSET** AREA1

MOV DI, **OFFSET** AREA2

MOV CX, 100

AGAIN : MOV AL, [SI]

MOV [DI], AL

INC SI ; 修改地址指针

INC DI ; 修改地址指针

DEC CX ; 修改个数

JNZ AGAIN

...

前缀 = 属性操作符

DS }



堆栈操作指令

什么是堆栈，为什么需要堆栈

- 堆栈是按照先进后出原则组织的一段内存区，存在于堆栈段中，SP在任何时候都指向栈顶。

❖ 通常用于存放一些重要数据，
如：程序的地址、或是需要恢复的数据。

- 为方便数据的存放和恢复，
设置专门的指针，指向堆栈中要操作的单元。
段值由 SS 给出，偏移地址由 SP 给出 DS 数

SS → 堆栈段寄存器 (stack segment)

SP → 堆栈指针寄存器 (stack point)

堆栈使用的场合

- 用堆栈保存恢复信息
- 子程序的调用、返回以及中断调用、返回
- 用堆栈传送数据



•PUSH 源操作数 (操作数必须为16位，不能是立即数)

进栈指令，先调整堆栈指针，再把源操作数压栈

(I) $SP \leftarrow SP-1$

(II) $OPRDH \rightarrow (SP)$

(III) $SP \leftarrow SP-1$

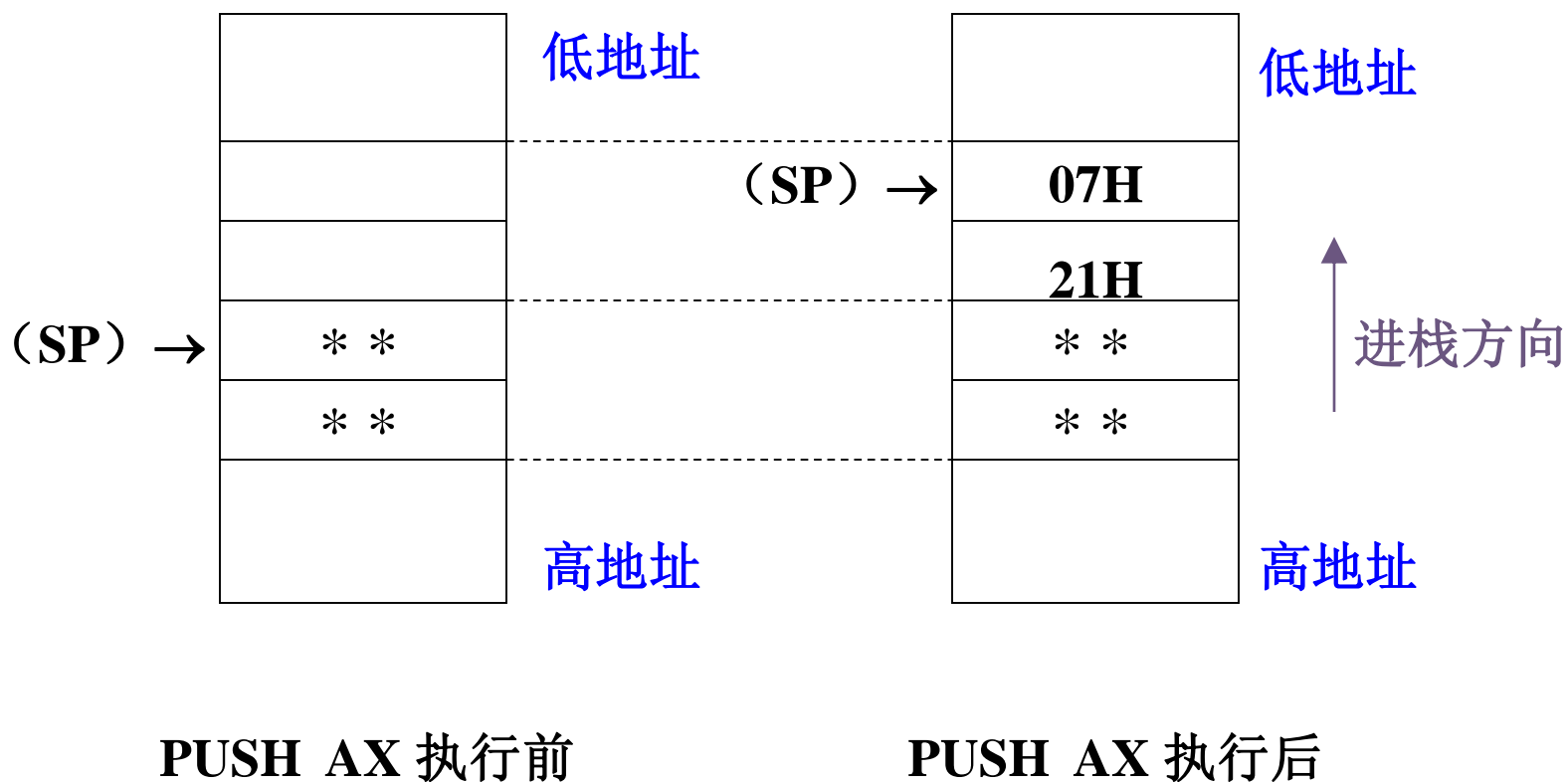
(IV) $OPRDL \rightarrow (SP)$

PUSH AX

PUSH WORD PTR [SI+5]



例： 假设 $(AX) = 2107H$ ，执行 **PUSH AX**



例 利用DEBUG学习PUSH指令

D:\MASM>DEBUG

-A ;汇编两条指令

1693:0100 MOV AX, 1234

1693:0103 PUSH AX

1693:0104

-R ;显示指令执行前寄存器值

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1693 ES=1693 SS=1693 CS=1693 IP=0100 NV UPEI PL NZ NA PO NC

1693:0100 B83412 MOV AX, 1234

-T=100 2 ;执行CS:100开始处的两条指令

....

AX=1234 BX=0000 CX=0000 DX=0000 SP=FFEC BP=0000 SI=0000 DI=0000

DS=1693 ES=1693 SS=1693 CS=1693 IP=0104 NV UPEI PL NZ NA PO NC

1693:0104 03C6 ADD AX,SI

-D SS:FFEC L10 ;查看栈顶内容

1693:FFE0 34 12 00 00 4...

1693:FFF0 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF
-

注意相关寄存器和内存单元内容的变化

•POP 目标操作数

出栈指令，先将栈顶2字节送目标操作数，再调整堆栈指针

(I) (SP) —OPRDL

(II) SP ← SP+1

(III) (SP) → OPRDH

(IV) SP ← SP+1

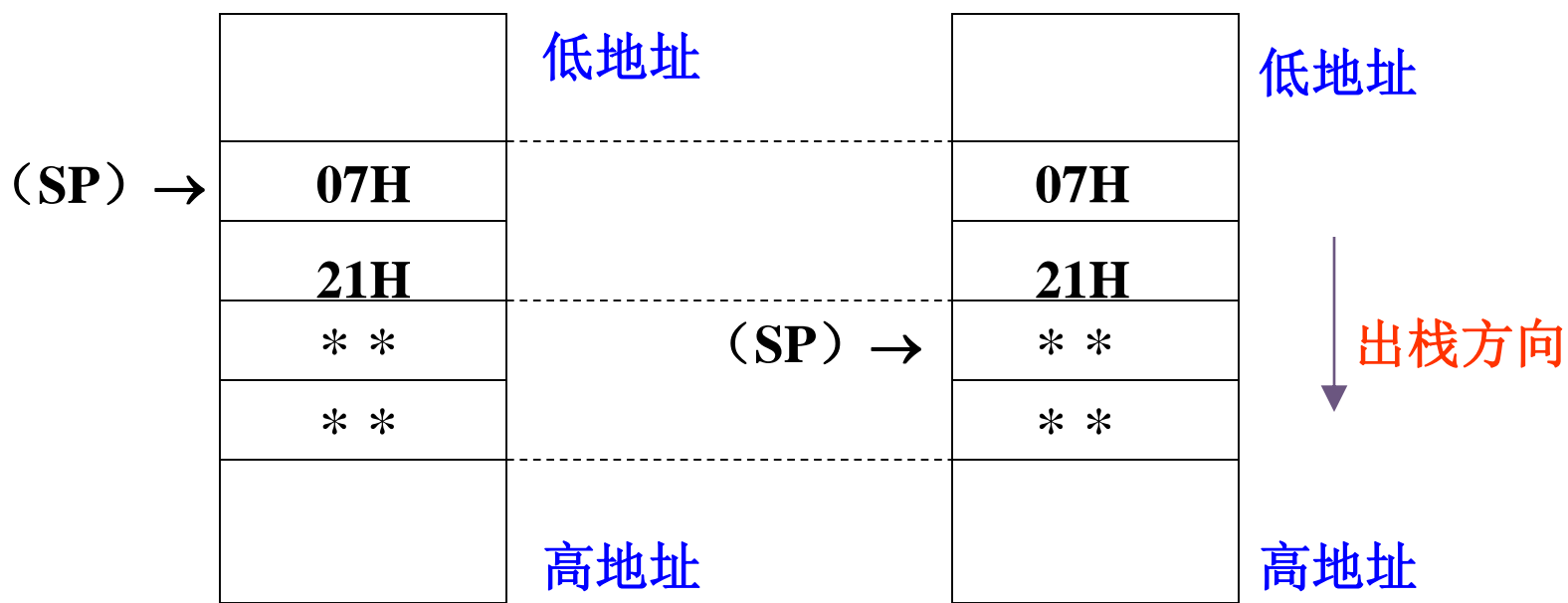
出栈 → 指针
指针 → 入栈

POP AX

POP WORD PTR [SI+5]

注意：堆栈操作只能对十六位的操作数进行操作，且为寄存器或存储器操作数。如：**PUSH AL** 是错误的。

例： POP BX



POP BX 执行前

POP BX 执行后

(BX) = 2107H

例 在DEBUG下学习POP指令

-A

```
1693:0100  MOV  BP, SP           ;取当前栈顶地址
1693:0102  MOV  WORD PTR [BP], 1234 ;用MOV指令使栈顶内容为1234H
1693:0107  POP  BX           ;出栈指令
1693:0108
```

-R

;查看指令执行前状态

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1693 ES=1693 SS=1693 CS=1693 IP=0100  NV UP EI PL NZ NA PO NC
```

```
1693:0100 89E5      MOV  BP, SP
```

-T=100

;执行CS:100处的第一条mov指令

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=FFEE SI=0000 DI=0000
DS=1693 ES=1693 SS=1693 CS=1693 IP=0102  NV UP EI PL NZ NA PO NC
```

```
1693:0102 C746003412  MOV  WORD PTR [BP+00], 1234      SS:FFEE=0000
```

-T

;执行下一条mov指令

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=FFEE SI=0000 DI=0000
DS=1693 ES=1693 SS=1693 CS=1693 IP=0107  NV UP EI PL NZ NA PO NC
```

```
1693:0107 5B      POP  BX
```

-T

;执行pop指令, 注意BX, SP的变化

```
AX=0000 BX=1234 CX=0000 DX=0000 SP=FFF0 BP=FFEE SI=0000 DI=0000
DS=1693 ES=1693 SS=1693 CS=1693 IP=0108  NV UP EI PL NZ NA PO NC
```

```
1693:0108 F5      CMC
```

-

交换指令

•XCHG 目标操作数, 源操作数

将源操作数与目的操作数的内容互换

XCHG AL, BL

XCHG [2500H], DX

- 1) 可以是字节交换也可以是字交换
- 2) 可以是寄存器与寄存器之间进行交换
- 3) 可以是寄存器与存储器之间进行交换
- 4) 不可以是存储器与存储器之间交换
- 5) CS和IP不能用来进行交换



例: **XCHG BX, [BP+SI]**
 XCHG AL, BH

注意:

- * 两个操作数中必须有一个在寄存器**
- * 可进行字或字节操作, 不影响标志位**
- * 不允许对立即数、段寄存器做操作数**

XCHG AX, 4

XCHG BX, DS



（二）累加器专用传送指令

1、查表转换指令

- **XLAT** 转换表名（即转换表首地址）

从转换表中查找出一个字节的内容，用其**取代AL**寄存器中的内容。

XLAT TABLE

转换表最长为256个字节，是由用户设计的。执行查表指令前，**BX**应指向转换表的起点。**AL**的内容被用作查表时索引，即被查找数在表中的位置。



格雷码转换表查找举例

MOV AL, 5

MOV BX, OFFSET TABLE

XLAT TABLE

执行结果为AL得到0AH，即5的格雷码。

格雷码转换表

TABLE

1

2

3

4

5

6

7

8

9

18H

34H

05H

06H

09H

0AH

0CH

11H

12H

14H

2. 输入/输出指令

完成累加器和I/O端口之间的数据传送

- **IN** 累加器, 端口号

- **OUT** 端口号, 累加器

端口号为8位时, 直接寻址, 最多可访问256个端口。

IN AL, PORT

OUT PORT, AL

端口地址为16位时, 间接寻址, 端口地址必须放在**DX**寄存器中, 最多可访问65536个端口。

IN AL, DX

OUT DX, AL



```
例:  IN  AX, 28H      ; MOV  DX, 28H
      ; IN   AX, DX
```

例: MOV DX, 3FCH
IN AX, DX

例: OUT 5, AL

例：测试某状态寄存器（端口号27H）的第2位是否为1

IN AL, 27H

TEST AL, 00000100B

JNZ ERROR ;若第2位为1，转ERROR处理

(三) 地址传送指令

- **LEA 目标寄存器, 源操作数**

有效地址传送指令，源操作数为内存操作数，将内存单元的**有效地址**（而不是内容）传送到目标寄存器，即将目的操作数的偏移地址送寄存器。

LEA R, SRC ; SRC代表源操作数, R代表寄存器

例：将TABLE的偏移地址送SI

LEA SI, TABLE

与**MOV SI, OFFSET TABLE**等效。

•LDS/LES 目标寄存器， 源操作数

指针传送指令， 将一个存放在4个存储单元中共计32位的目标指针（段地址和偏移量）传送到两个目的寄存器。

LDS DI, [2130H]

把2132H， 2133H中的内容送DS， 把2130H， 2131H的内容送DI。

LES与**LDS**基本相同， 区别只是把目的段地址送**ES**寄存器。



例:

TABLE
(DS):1000H

40 H
00 H
00 H
30 H

MOV BX, TABLE ; **(BX)=0040H**

MOV BX, OFFSET TABLE ; **(BX)=1000H**

LEA BX, TABLE ; **(BX)=1000H**

LDS BX, TABLE ; **(BX)=0040H**

; **(DS)=3000H**

LES BX, TABLE ; **(BX)=0040H**

; **(ES)=3000H**

（四）标志传送指令

- **LAHF** （PSW低8位——》 AH）

把标志寄存器中SF、ZF、AF、PF、CF五个标志位传到AH的第7、6、4、2、0，第5、3、1没定义。

- **SAHF**

作用与LAHF相反，将AH中的内容送至标志寄存器中。

- **PUSHF**

把标志寄存器压栈。

- **POPF**

将标志寄存器退栈。

•例：将标志寄存器的 TF 置 1 。

PUSHF

POP AX

OR AX , 0100H

PUSH AX

POPF



二、算术运算指令

加、减、乘、除，运算对象8/16位有符号/无符号整数，以及BCD码，影响标志位。

1. 加法指令

• **ADD** 目标操作数， 源操作数

源操作数+目标操作数——>目标操作数

不含进位

• **ADC** 目标操作数， 源操作数

源操作数+目标操作数+CF——>目标操作数

含进位

影响A,C,O,P,S,Z 6个标志位

• **INC** 目标操作数

目标操作数+1——>目标操作数

影响A,O,P,S,Z 5个标志位

加法指令对**条件标志位** (CF/OF/ZF/SF) 的影响:

$$SF = \begin{cases} 1 & \text{结果为负} \\ 0 & \text{否则} \end{cases}$$

$$ZF = \begin{cases} 1 & \text{结果为0} \\ 0 & \text{否则} \end{cases}$$

$$CF = \begin{cases} 1 & \text{和的最高有效位 有 向高位的进位} \\ 0 & \text{否则} \end{cases}$$

$$OF = \begin{cases} 1 & \text{两个操作数符号相同, 而结果符号与之相反 溢出} \\ 0 & \text{否则} \end{cases}$$

只有符号相同才可能溢出

CF 位表示 **无符号数** 相加的溢出。

OF 位表示 **带符号数** 相加的溢出。

n=8bit 带符号数(-128~127) 无符号数(0~255)

0 0 0 0 0 1 0 0

+ 0 0 0 0 1 0 1 1

0 0 0 0 1 1 1 1

带: $(+4) + (+11) = +15$ OF=0

无: $4 + 11 = 15$ CF=0

带符号数和无符号数都不溢出

1 0 0 0 0 1 1 1

+ 1 1 1 1 0 1 0 1

0 1 1 1 1 1 0 0

带: $(-121) + (-11) = +124$ OF=1

无: $135 + 245 = 124$ CF=1

带符号数和无符号数都溢出

0 0 0 0 0 1 1 1

+ 1 1 1 1 1 0 1 1

0 0 0 0 0 0 1 0

带: $(+7) + (-5) = +2$ OF=0

无: $7 + 251 = 2$ CF=1

无符号数溢出

0 0 0 0 1 0 0 1

+ 0 1 1 1 1 1 0 0

1 0 0 0 0 1 0 1

带: $(+9) + (+124) = -123$ OF=1

无: $9 + 124 = 133$ CF=0

带符号数溢出

例：双精度数的加法

(DX) = 0002H (AX) = 0F365H

(BX) = 0005H (CX) = 0E024H

指令序列 (1) **ADD AX, CX**

(2) **ADC DX, BX**

(1) 执行后, **(AX) = 0D389H**

CF=1 OF=0 SF=1 ZF=0

(2) 执行后, **(DX) = 0008H**

CF=0 OF=0 SF=0 ZF=0



2. 减法指令

- **SUB** 目标操作数， 源操作数

目标操作数-源操作数→目标操作数

- **SBB** 目标操作数， 源操作数

目标操作数-源操作数-CF→目标操作数

- **DEC** 目标操作数

目标操作数-1→目标操作数

- **NEG** 目标操作数

0-目标操作数→目标操作数， 即对给出的字节或字操作数**求补**

影响A,C,O,P,S,Z6个标志位

- **CMP** 目标操作数， 源操作数

比较指令，执行两数相减操作，但不送回相减结果只是影响标志位。

- (1) 两数为无符号数的比较

如果**CF**为**0**则表示无借位，被减数大于减数；如果**CF**为**1**，则表示有借位，被减数小于减数。

- (2) 两数为有符号数的比较

如果**OF**和**SF**相等，则表示被减数大于减数；如果**OF**和**SF**不相等（相异），则表示被减数小于减数。



例： x 、 y 、 z 均为双精度数，分别存放在地址为 X , $X+2$;
 Y , $Y+2$; Z , $Z+2$ 的存储单元中，用指令序列实现
 $w \leftarrow x + y + 24 - z$ ，并用 W , $W+2$ 单元存放 w

```
MOV    AX,    X
MOV    DX,    X+2
ADD    AX,    Y
ADC    DX,    Y+2           ;  x+y
ADD    AX,    24
ADC    DX,    0             ;  x+y+24
SUB    AX,    Z
SBB    DX,    Z+2          ;  x+y+24-z
MOV    W,     AX
MOV    W+2,   DX           ;  结果存入W, W+2单元
```

3. 乘法指令

•**MUL** 乘数
无符号乘法

•**IMUL** 乘数
有符号乘法

(1) 字节乘，**被乘数放在AL中**，乘积结果的低8位放在AL中，高8位放在AH中。

(2) 字乘，**被乘数放在AX中**，乘积结果的低16位放在AX中，高16位放在DX中。



4. 除法指令

• **DIV** 除数
无符号除法

• **IDIV** 除数
有符号除法

注意：除数必须为被除数的一半字长

(1) 字节除，被除数放在**AX**，则商放在**AL**中，余数放在**AH**中。

(2) 字除，被除数放在**DX AX**中，则商放在**AX**中，余数放在**DX**中。

● 类型转换指令

CBW

AL → AX (字节扩展成字)

执行操作：若 **AL < 80H**，则 **(AH) = 00H**

若 **AL ≥ 80H**，则 **(AH) = 0FFH**

CWD

AX → (DX, AX) (字扩展成双字)

执行操作：若 **AX < 8000H**，则 **(DX) = 0000H**

若 **AX ≥ 8000H**，则 **(DX) = 0FFFFH**

例：(AX) = 0BA45H

CBW ; (AX)=0045H

CWD ; (DX)=0FFFFH (AX)=0BA45H

注意：

- * 无操作数指令
- * 隐含对AL 或AX 进行符号扩展
- * 不影响条件标志位

例: x, y, z, v 均为16位带符号数, 计算
 $(v - (x * y + z - 540)) / x$

```
MOV    AX, X
IMUL   Y           ; x*y
MOV    CX, AX
MOV    BX, DX
MOV    AX, Z
CWD
ADD    CX, AX
ADC    BX, DX      ; x*y+z
SUB    CX, 540
SBB    BX, 0       ; x*y+z-540
MOV    AX, V
CWD
SUB    AX, CX
SBB    DX, BX      ; v-(x*y+z-540)
IDIV   X           ; (v-(x*y+z-540))/x
```

5. BCD码运算指令

- 十进制调整指令

BCD码： 用二进制编码的十进制数，又称**二--十进制数**

组合的BCD码： ^{压缩型}用 4 位二进制数表示 1 位十进制数

例： $(59)_{10} = (0101\ 1001)_{\text{BCD}}$

非组合的BCD码： 用 8 位二进制数表示 1 位十进制数

例： $(59)_{10} = (0000\ 0101\ 0000\ 1001)_{\text{BCD}}$

→ ASCII

例：写出 $(3590)_{10}$ 的组合BCD码和非组合BCD码，并分别把它们存入数据区**PACKED**和**UNPACK**。

组合BCD： $(3590)_{10} = (0011\ 0101\ 1001\ 0000)_{\text{BCD}}$

18

非组合BCD：

$(3590)_{10} = (00000011\ 00000101\ 00001001\ 00000000)_{\text{BCD}}$

PACKED

90H
35H

UNPACK

00H
09H
05H
03H

组合的BCD码调整指令

问题的提出:

$$\begin{array}{r} 19 \\ + 08 \\ \hline 27 \end{array} \quad \text{组合BCD:} \quad \begin{array}{r} 0001 \ 1001 \\ + 0000 \ 1000 \\ \hline \end{array}$$

$(0010 \ 0111)_{\text{BCD}}$ ← $0010 \ 0001 + 110$ (AF=1)

十六修正

加法的十进制调整指令: **DAA**

执行操作: $(AL) \rightarrow (AL)_{\text{组合BCD}}$

注意:

- * 隐含的操作寄存器为AL
 - * 紧接在加减指令之后使用
 - * 影响条件标志位
- (对OF无定义)

减法的十进制调整指令: **DAS**

执行操作: $(AL) \rightarrow (AL)_{\text{组合BCD}}$

AAA AAS
AAM AAD 非组合

组合的BCD码运算调整指令 **DAA**

两个组合的BCD码相加，**结果在AL中**，执行该指令后将结果调整为十进制，放在AL中。

MOV AL, 56H

ADD AL, 47H ; AL: 9DH

DAA ; AL: 03H CF=1



未组合的BCD码运算调整指令 AAA

两个未组合的BCD码相加，结果在AL中，执行该指令后将结果调整为十进制，放在AX中。

MOV AL, 7H

ADD AL, 5H ; AL: 0CH

AAA ; AX: 0102H CF=AF=1



- DAS**

减法的组合BCD码调整指令

- AAS**

减法的非组合BCD码调整指令

- AAM**

BCD码的乘法十进制调整指令

- AAD**

BCD码除法十进制调整指令



三 逻辑运算指令

👉 位操作

AND 目标操作数, 源操作数

OR 目标操作数, 源操作数

XOR 目标操作数, 源操作数

NOT 目标操作数

👉 位操作

AND 按位**相与**，主要用于将二进制数的某些位清0。

AND BL, 0FH

	xxxxxxxx	(BL)
AND	00001111	0F
	<hr/>	
	0000xxxx	结果

👉 位操作

OR 按位 **相或**，主要用于将二进制数的某些位置1。

OR BL, 0FH

	xxxxxxxx	(BL)
OR	00001111	0F
	<hr/>	
	xxxx1111	结果

👉 位操作

XOR 按位**相异或**，主要用于将二进制数的某些位求反。

XOR BL, 0FH

	XXXXXXXX	(BL)
XOR	00001111	0F
	<hr/>	
	XXXXXXXX	结果

☞ 位测试 TEST指令

TEST 与**AND**一样将两个操作数**按位相与**，但**结果不回送**，只影响标志位。

当被测试为为0时，ZF置位（ZF=1）；
当被测试为为1时，ZF复位（ZF=0）

```
TEST AL, 1
JNZ  RIGHT
TEST AL, 128
JNZ  LEFT
```

TEST — AND
CMP — SUB

例：屏蔽AL的0、1两位

AND AL, 0FCH

	*	*	*	*	*	*	*	*	*
AND	1	1	1	1	1	1	0	0	
	*	*	*	*	*	*	0	0	

例：置AL的第5位为1

OR AL, 20H

	*	*	*	*	*	*	*	*	*
OR	0	0	1	0	0	0	0	0	
	*	*	1	*	*	*	*	*	

例：使AL的0、1位变反

XOR AL, 3

	*	*	*	*	*	*	*	*	*
XOR	0	0	0	0	0	0	1	1	
	*	*	*	*	*	*	*	*	

例：测试某些位是0是1

TEST AL, 1
JZ EVEN

移位操作

逻辑移位：无符号数 (**SHL, SHR**)

算术移位：有符号数 (**SAL, SAR**)

指令格式：

SHL 目标操作数，计数

移1位时，计数值可以为立即数1

移多位时，计数值必须先存入CL寄存器

MOV CL, 3

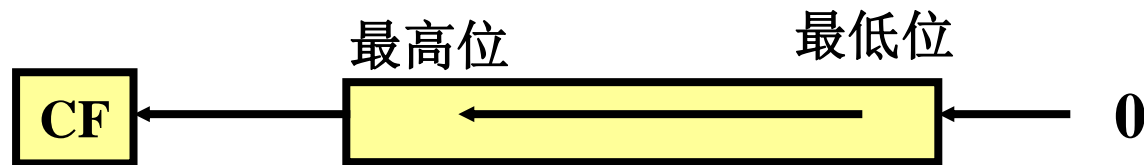
SHL AX, CL

SHL AX, 1

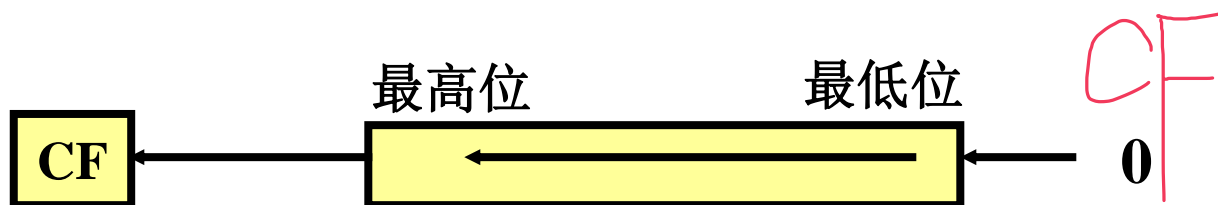
← 3位

☞ 移位指令操作过程 —— 非循环移位

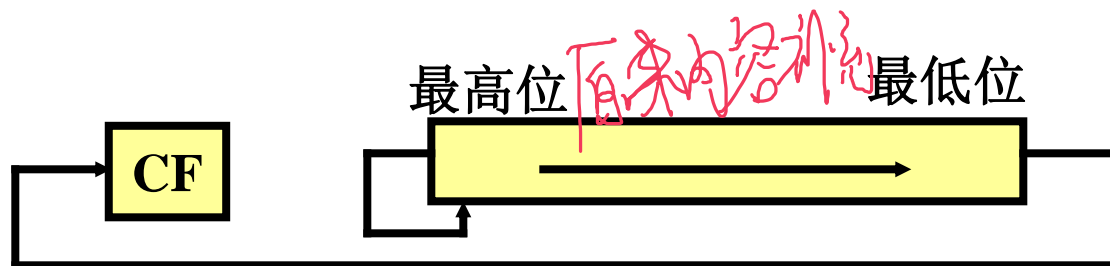
算术左移SAL



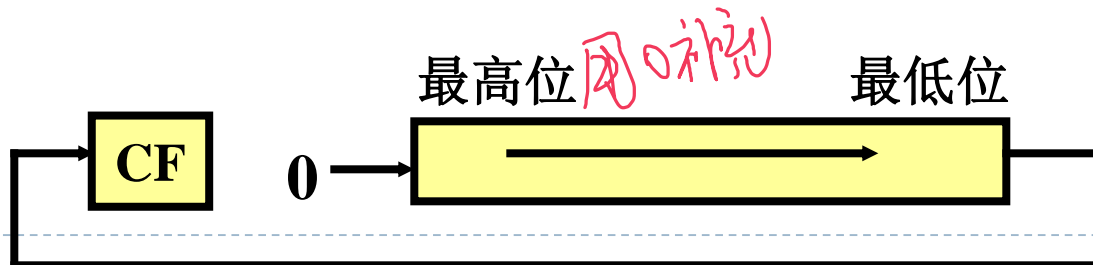
逻辑左移SHL



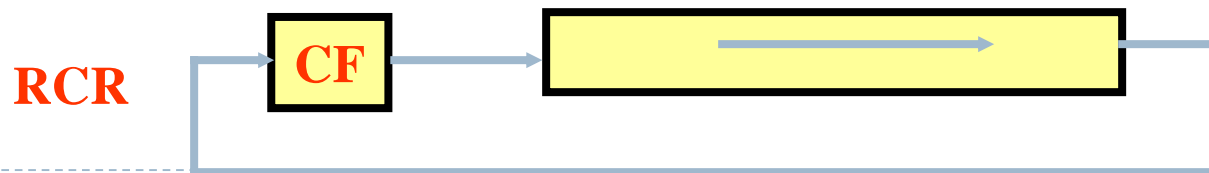
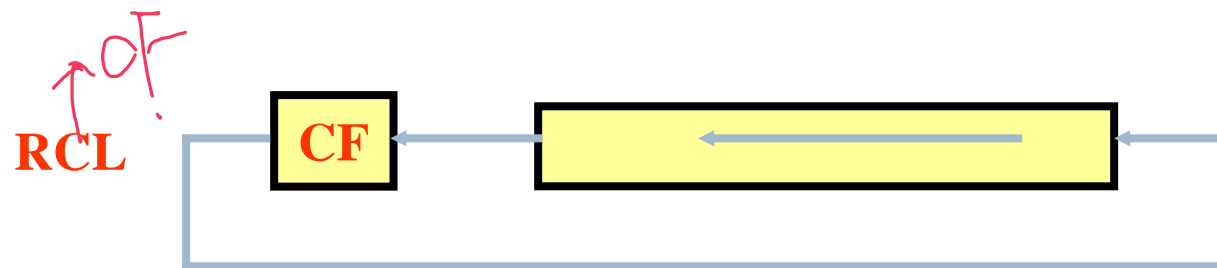
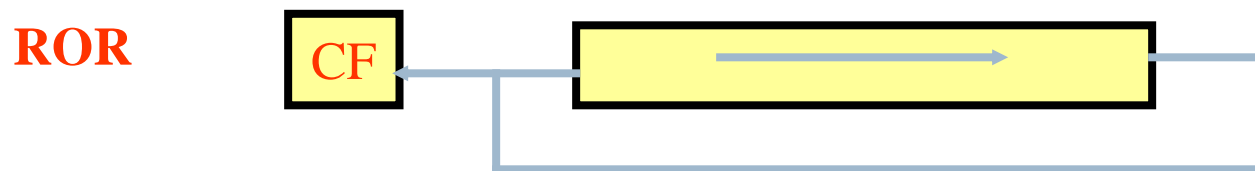
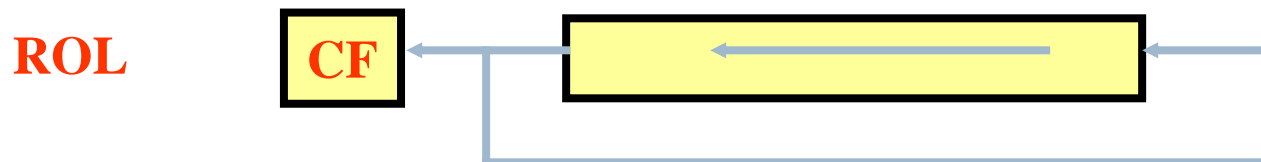
算术右移SAR



逻辑右移SHR



👉 移位指令操作过程 — 循环移位指令



例： (AX)= 0012H, (BX)= 0034H, 把它们装配成(AX)= 1234H

MOV CL, 8

ROL AX, CL

ADD AX, BX

例： (BX)=84F0H

(1) (BX)为无符号数, 求(BX)/2

SHR BX, 1 ; (BX) = 4278H

(2) (BX)为带符号数, 求(BX)/2

SAR BX, 1 ; (BX) = 0C278H

(3) 把(BX)中的16位数每4位压入堆栈

MOV CH, 4 ; 循环次数

MOV CL, 4 ; 移位次数

NEXT: ROL BX, CL

MOV AX, BX

AND AX, 000FH

PUSH AX

DEC CH

JNZ NEXT

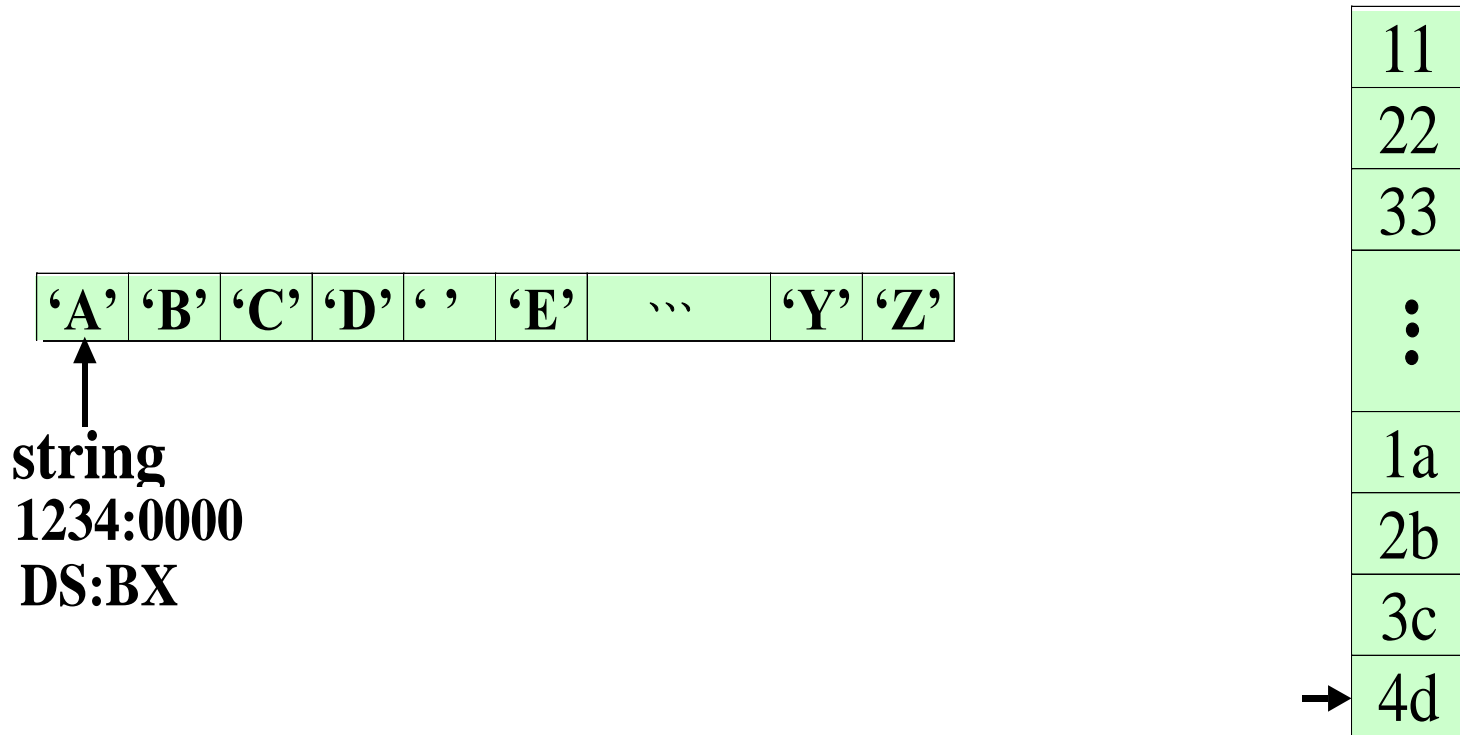
	← (SP)
0000	
000F	
0004	
0008	

四、串操作指令

- 串的基本概念

顺序存放在内存中的一组数据，称为串。

用串的首（末）地址、元素类型、串的长度表示。



👉 串传送

MOVSB 将一个字节/字从
MOVSW **DS:SI** → **ES:DI**

指令前要先将源串首地址 → **DS:SI**
目标串首地址 → **ES:DI**



👉 串传送

完成操作后自动修改SI、DI，使其指向串的下一个元素

串操作方向由**CLD**和**STD**指令设置

CLD 地址递增方向 (**DF=0**)

STD 地址递减方向 (**DF=1**)

FLAG, DF

👉 串传送

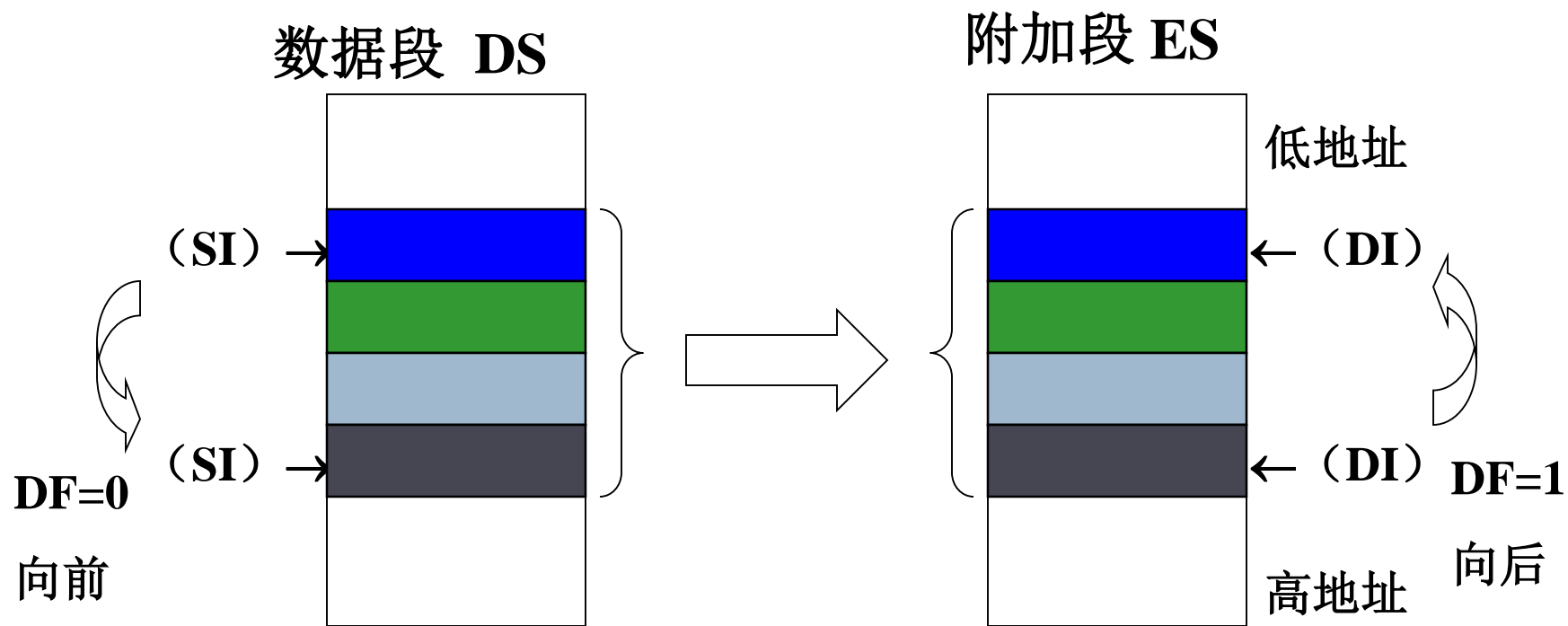
——重复前缀, 等效 LOOP 循环

REP MOVSB

REP MOVSW

需要先将串的长度存入 CX寄存器

每处理完一个元素自动使 CX-1, 直到 **CX=0**
才结束串传送——完成 整个串 的传送



例

```
dataarea segment  
mess1 db 'personal_computer'  
dataarea ends
```

```
extra segment  
mess2 db 17 dup (?)  
extra ends
```

```
code segment
```

...

```
lea si, mess1  
lea di, mess2  
mov cx, 17  
cld  
rep movsb
```

...

```
code ends
```

```
lea si, mess1+16  
lea di, mess2+16  
mov cx, 17  
std  
rep movsb
```

👉 串比较

CMPSB 比较地址为**DS:SI**、**ES:DI**的两个
CMPSW 字节/字，同时修改**SI**和**DI**指向下一个元素。

指令前通常加重复前缀**REPZ/REPE**,
REPNZ/REPNE，从而当发现两个串不同
(或相同) 时结束比较,这样就可以找到两个
串中第一个不相等元素或第一个相等的元
素。



👉 串比较

例:

```
MOV SI, OFFSET S1  
MOV DI, OFFSET S2  
MOV CX, xx  
REPZ CMPSB  
JNZ Not_Equal
```

Equal:

.....

Not_Equal:

.....



👉 目标串搜索指令

SCASB	在首地址为 ES:DI 的串中搜索
SCASW	某个元素（字节/字），同时 修改 DI 指向下一个元素。

事先要将待搜索的元素存入**AL/AX**

指令前通常加重复前缀**REPZ/REPNE**，
从而当发现待搜索的元素时结束搜索。



👉 串搜索

例：

```
MOV DI, OFFSET String  
MOV CX, xx  
MOV AL, 'h'  
REPNZ SCASB  
JNZ Not_Found
```

Found:

.....

Not_Found:

.....



串装载

LODSB

LODSW

将地址为**DS:SI**的一个字节/字装入**AL/AX**，同时修改**SI**指向下一个元素。

串装入指令没有重复前缀



👉 串装载

例：

```
MOV SI, OFFSET String
MOV CX, 8
NextChar: LODSB
           MOV DL, AL
           MOV AH, 2
           INT 21
           LOOP NextChar
```



👉 串存储

STOSB	将 AL/AX 的值存入地址为
STOSW	ES:DI 的内存单元，并修改 DI
	指向下一个元素。

利用重复前缀**REP**，可以建立一个取值相同的数据串。



五、程序控制指令

控制程序的流向:

无条件转移

条件转移

循环控制

过程调用与返回

中断指令



👉 无条件转移

相当于goto语句

JMP Label1

.....

Label1:

- | | |
|-----------|---|
| (1)段内直接转移 | JMP 2000H ; IP=2000H |
| (2)段内间接转移 | JMP AX ; IP=(AX) |
| (3)段间直接转移 | JMP 2500H:0100H
;CS=2500H,IP=0100H |
| (4)段间间接转移 | JMP DWORD PTR[SI] |

;IP和CS的内容用内存中2个连续的字来替代。

👉 条件转移

根据执行上一指令后标志寄存器的状态而决定是否转移

1、判断无符号数大小的条件转移

JA/JNBE ; $>$,CF v ZF=0

JAE/JNB ; \geq ,CF=0

JB/JNAE ; $<$,CF=1

JBE/JNA ; \leq ,CF v ZF=1

👉 条件转移

2、判断有符号数大小的条件转移

JG/JNLE	;> ,ZF=0且SF xor OF=0
JGE/JNL	;>= ,SF xor OF=0
JL/JNGE	<b;< ,sf="" of="1</b" xor=""></b;<>
JLE/JNG	;<= ,ZF=1或Sf xor OF=1



👉 条件转移

AF BCD码调整用

3、单标志位条件转移

JZ/JE ;ZF=1

JNZ/JNE ;ZF=0

JC ;CF=1

JNC ;CF=0

JO ;OF=1

JNO ;OF=0

JP/JPE ;PF=1

JNP/JPO ;PF=0

JS ;SF=1

JNS ;SF=0

例

**X>50, 转到TOO_HIGH;
计算X-Y, 溢出转到OVERFLOW, 否则
|X-Y|→RESULT**

```
MOV AX, X
CMP AX, 50
JNLE TOO_HIGH
SUB AX, Y
JO OVERFLOW
JNS NONNEG
NEG AX
NONNEG:  MOV RESULT, AX
        INT 20H
TOO_HIGH: ...
        INT 20H
OVERFLOW: ...
        INT 20H
```

👉 循环控制 *for/while*

循环控制指令所控制的目的地址都用**标号**表示，该标号都在距当前IP: -128~+127范围，且与CX配合使用，CX存放循环次数。

LOOP

LOOPE/LOOPZ

LOOPNE/LOOPNZ

结束条件:

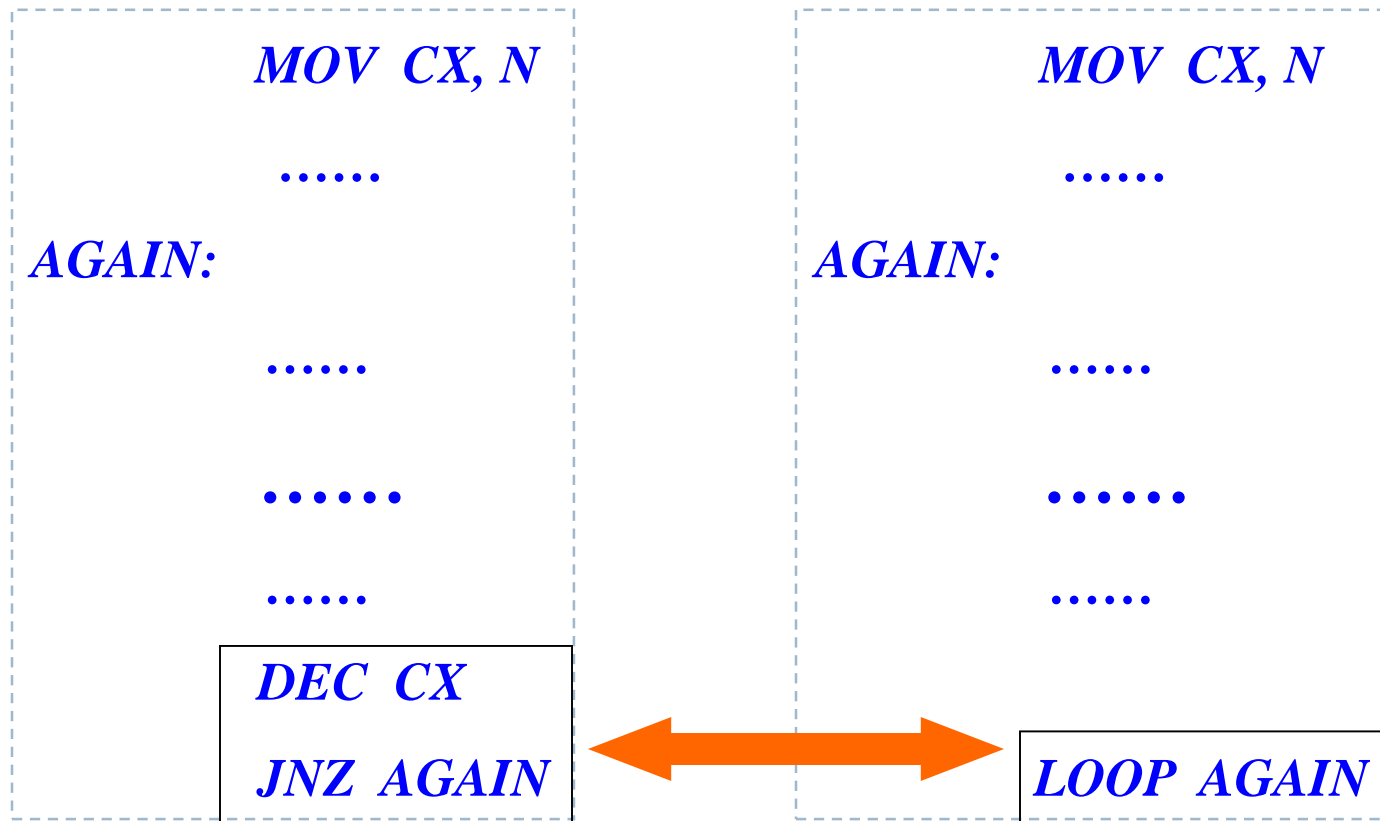
CX=0

CX=0或者**ZF=0**

CX=0或者**ZF=1**

IF有效
ZF有效
ZF=1

默认减一



循环的条件转移指令
实现方法

循环的循环指令实现方法

例：在多重循环的程序结构中，CX计数器的保存和恢复

```
        MOV CX, M
AGAIN:   .....
        PUSH CX
        MOV CX, N
NEXT:    .....
        LOOP NEXT
        .....
        POP CX
        LOOP AGAIN
```

```
        MOV DI, M
AGAIN:   .....
        MOV CX, N
NEXT:    .....
        LOOP NEXT
        .....
        DEC DI
        JNZ AGAIN
```

👉 子程序调用和返回指令

段内、段间调用

段内直接调用: **CALL 1000H** ;IP=1000H

CALL LABEL; IP=LABEL地址

段内间接调用: **CALL AX** ;IP=(AX)

段间直接调用: **CALL 2500:1000H**

;IP=1000H , CS=2500H

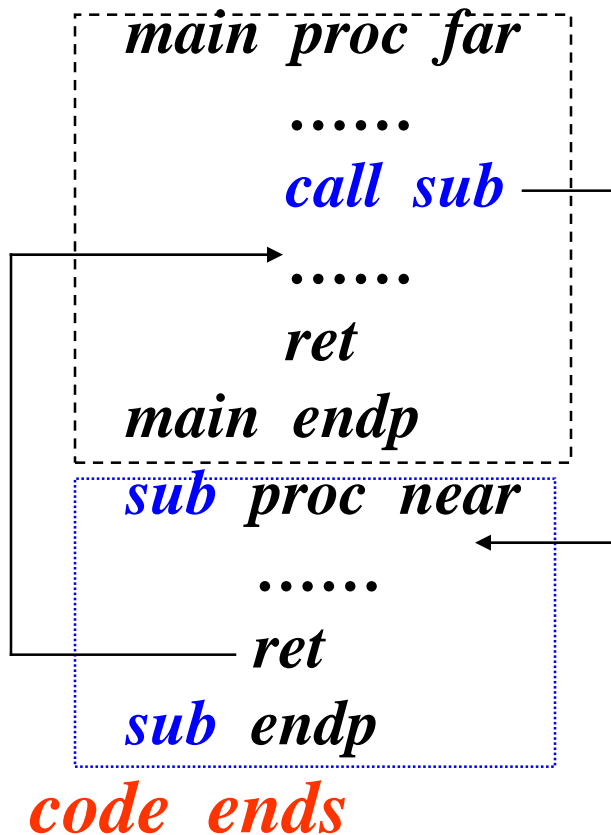
段间间接调用: **CALL DWORD PTR [SI]**

;IP在SI指向的单元中, CS
在IP+2指向的单元中

▶ 返回指令: **RET** ;弹出IP和CS, 返回主程序

子程序调用和返回指令

code segment



code1 segment

```
main proc far
    .....
    call sub
    .....
    ret
main endp
code1 ends
```

code2 segment

```
sub proc far
    .....
    ret
sub endp
code2 ends
```

段内调用和返回

段间调用和返回

中断指令

1、什么是中断？

当系统运行或程序运行期间在遇到某些特殊情况时，需要计算机自动执行的一组专门的例行程序进行处理。

2、什么是中断例行程序？

中断时所执行的这组程序

3、CPU响应一次中断的过程是怎样的？

类似与子程序的调用，只不过多了保护**FLAGS**

4、什么是中断向量？

► 中断例行程序的入口地址

中断向量区

00000	类型0的(IP)	}	类型0
	类型0的(CS)		
00004	类型1的(IP)	}	类型1
	类型1的(CS)		
4*N		}	类型N
	类型N的(IP)		
	类型N的(CS)		
003FC	类型255的(IP)	}	类型255
	类型255的(CS)		

中断向量:

中断例行程序的入口地址, 存放于中断向量区。

☞ 中断指令 **INT n**, **INTO** 和 **IRET**

执行 **INT n** 时，会引起 CPU 转入一个中断服务程序。其过程为：

- (I) 首先标志位入堆栈
- (II) 清除中断允许标志 **IF** 和单步标志 **TF**
- (III) 保护断点，即断点地址入栈
- (IV) 将 $n \times 4$ 得到的中断向量送 **IP** 和 **CS**



☞ 中断指令INT n, INTO和IRET

INTO 为溢出中断指令，当OF=1是，INTO的中断处理程序会给出出错标志。

INTO=INT 4

IRET 中断返回指令，和中断指令配套使用，用以退出中断过程，返回到主程序。

👉 处理器控制指令

1、标志操作指令

<u>CLC</u>	;清CF=0
CMC	;使CF取反
STC	;置CF=1
CLD	;清DF=0
STD	;置DF=1
CLI	;清IF=0
STI	;置IF=1



☞ 处理器控制指令

2. 处理器暂停指令 HLT

挂机

以下三种情况可使8088脱离暂停状态:

- (1) 在RESET上有复位信号
- (2) 在NMI线上有非屏蔽中断请求
- (3) 中断允许 (IF=1) 时, INTR上有请求



👉 处理器控制指令

3. 处理器交权指令ESC

用于向外部处理机提供了从8088获得操作码和存储器操作数的手段，用于多处理机中，即8088的最大工作模式。

4. 等待指令WAIT

使CPU进入等待状态，直至TEST线上的信号有效为止，用于与外部接口电路的同步。

☞ 处理器控制指令

5. 总线封锁指令LOCK

可以放在任何一条指令前，是一个指令前缀。用于最大工作模式。在执行紧跟在LOCK前缀之后的那条指令期间，发出总线封锁信号，并使该信号保持到该指令执行完。LOCK信号有效期间其它处理机不得占用总线。

6. 空操作指令NOP

NOP使CPU不做任何工作，它不影响标志，主要用于时序配合。
