

第3-3章 8086的指令系统

主讲：许万茹





本章教学内容

- 数据传输类指令

MOV、PUSH、POP、XCHG、LEA、IN和OUT

- 算术运算指令

ADD、ADC、INC、SUB、SBB、DEC、MUL、IMUL、CMP

- 逻辑运算和移位指令

AND、OR、NOT、XOR、TEST、SAL、SAR、SHL、SHR、ROL和ROR

- 控制转移类指令：无条件转移和条件转移指令

- 循环控制指令 LOOP

- 子程序的调用和返回指令 CALL、RET

- 处理器的控制指令 STC、CLC、STI和CLI





8086的指令系统

- 数据传送指令
- 算术指令
- 逻辑指令
- 串处理指令
- 控制转移指令
- 处理机控制指令
- 杂项操作指令

重点关注

- 指令的基本功能
- 指令支持的寻址方式
- 指令的执行对标志位的影响
- 指令的特殊要求



数据传送指令

- 通用数据传送指令 MOV、PUSH、POP、XCHG
- 累加器专用传送指令 IN、OUT、XLAT
- 地址传送指令 LEA、LDS、LES
- 标志寄存器传送指令 LAHF、SAHF、PUSHF、POPF
- 其它



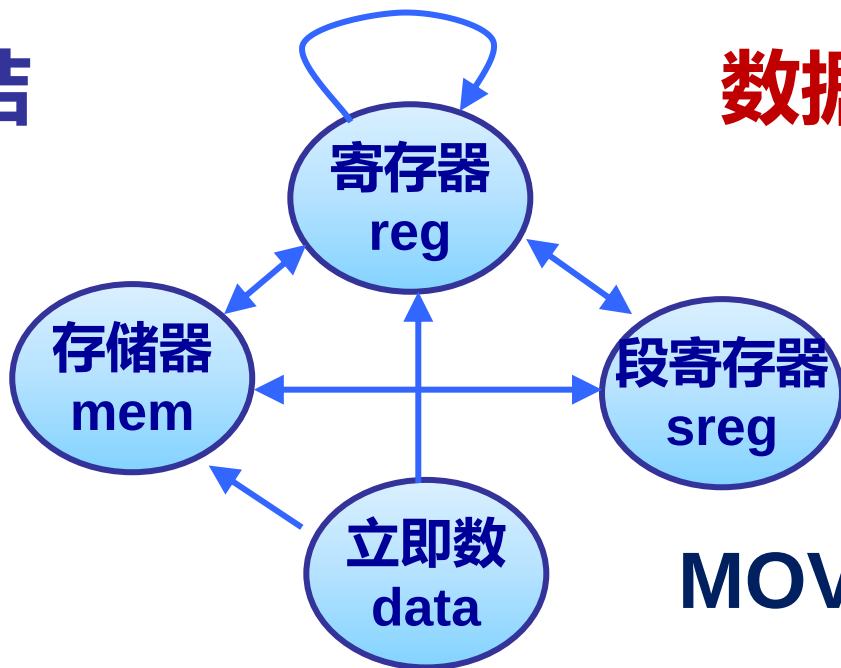
通用数据传送指令

1、传送指令：MOV DST, SRC 执行操作：(DST) $\leftarrow$ (SRC)

- ◆ DST、SRC 不能同时为段寄存器 MOV DS, ES $\times$
- ◆ 立即数不能直接送段寄存器 MOV DS, 2000H $\times$
- ◆ 不允许使用AX、CX、DX 存放 EA
- ◆ DST不能是立即数和CS。CS和IP寄存器的修改只能使用JMP XX:YY指令进行修改，不能用MOV指令修改。通常来说，CS都是由OS分配的，应用程序不能对其进行修改。
- ◆ DST、SRC不能同时为存储器寻址。因为两者的位宽是否匹配不能确定
- ◆ DST、SRC的类型要匹配。
- ◆ 不影响标志位 (对所有的数据传送指令而言)



小结



MOV AL, BL

MOV DS, AX

MOV DX, 5020H

数据传送指令 MOV

MOV 52, AH

目的操作数不能是立即数

MOV CS, AX

目的操作数不能是CS

MOV [DI], [SI]

不能同时为内存单元

MOV IP, BX

目的操作数不能是IP



2、堆栈操作指令 PUSH/POP

进栈指令: PUSH SRC

执行操作:

$(SP) \leftarrow (SP) - 2$

$((SP+1), (SP)) \leftarrow (SRC)$

出栈指令: POP DST

执行操作:

$(DST) \leftarrow ((SP+1), (SP))$

$(SP) \leftarrow (SP) + 2$

堆栈: “先进后出”的存储区。段地址存放在SS中，堆栈指针SP在任何时候都指向栈顶，进出栈后自动修改SP。

注意: * 堆栈操作**必须以字为单位**

* 不影响标志位

* 不能用立即寻址方式

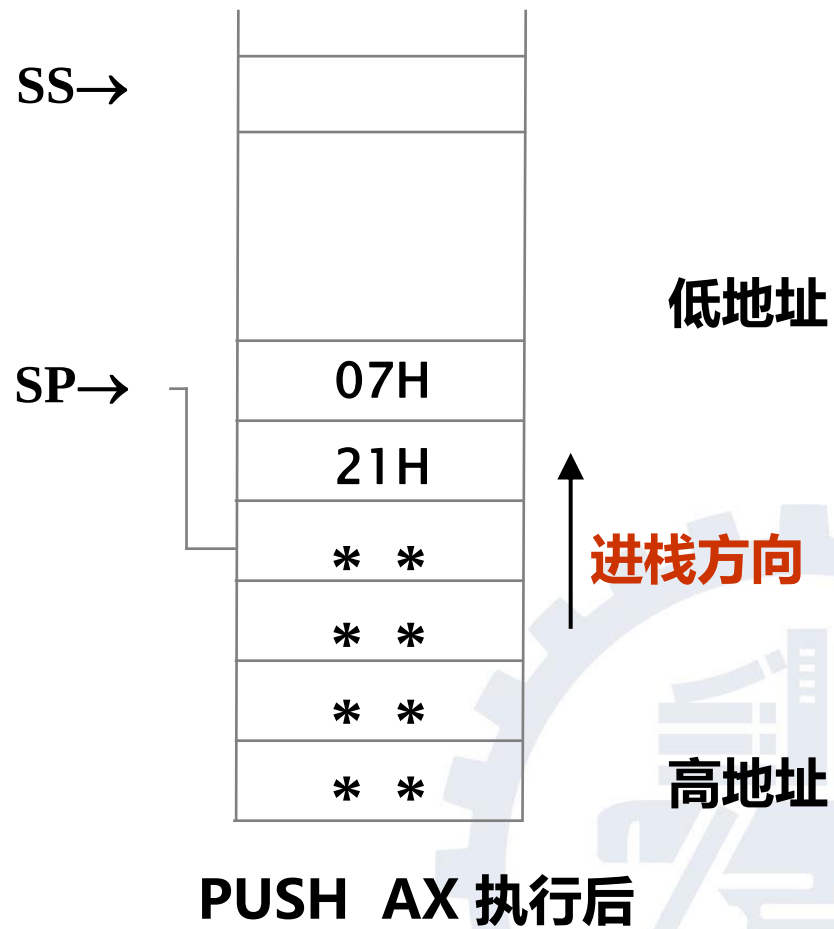
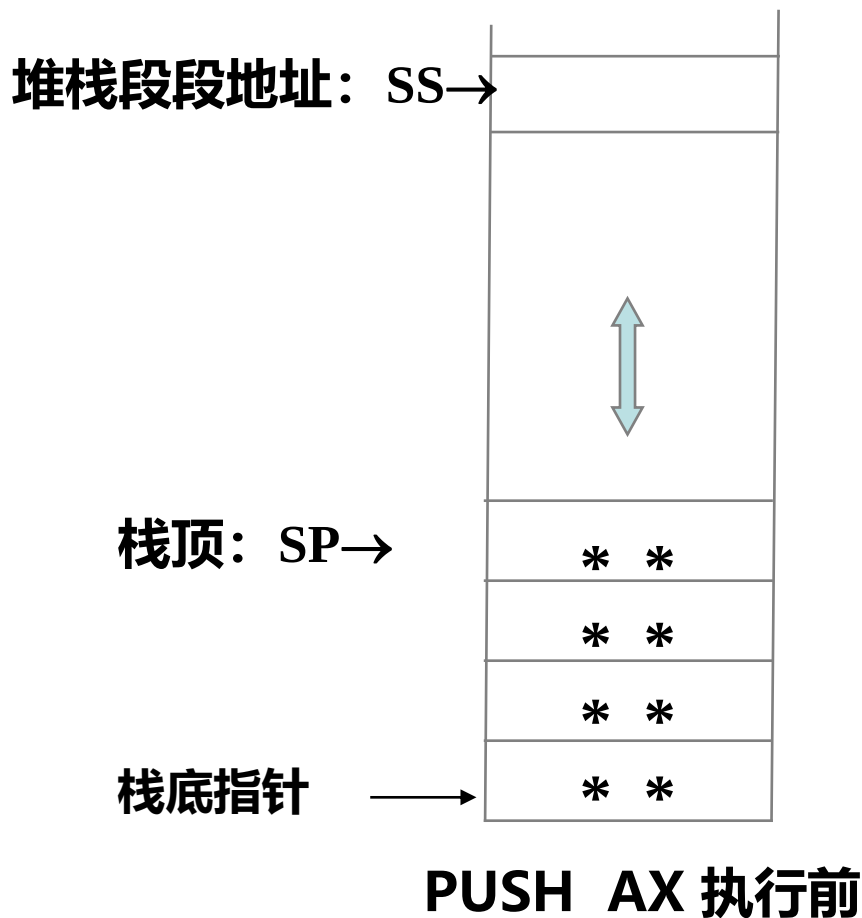
PUSH 1234H ×

* DST不能是CS (CS只能用JMP指令间接修改)

POP CS ×

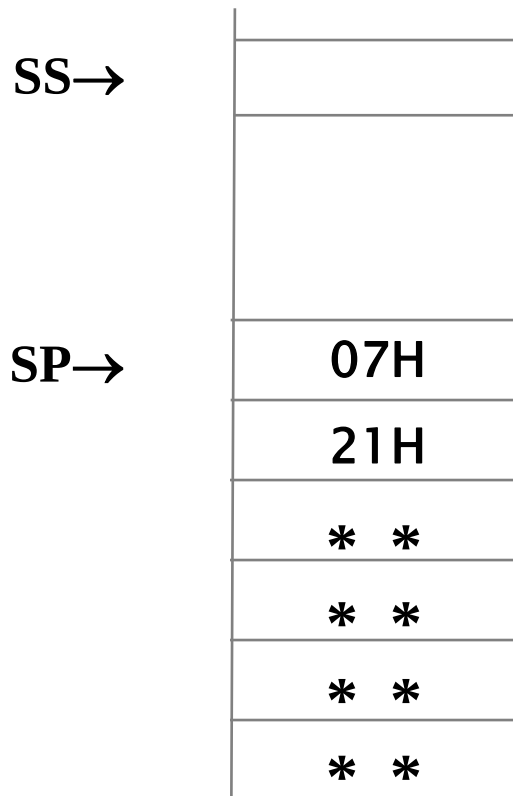


例：假设 $AX = 2107H$ ，执行 `PUSH AX`

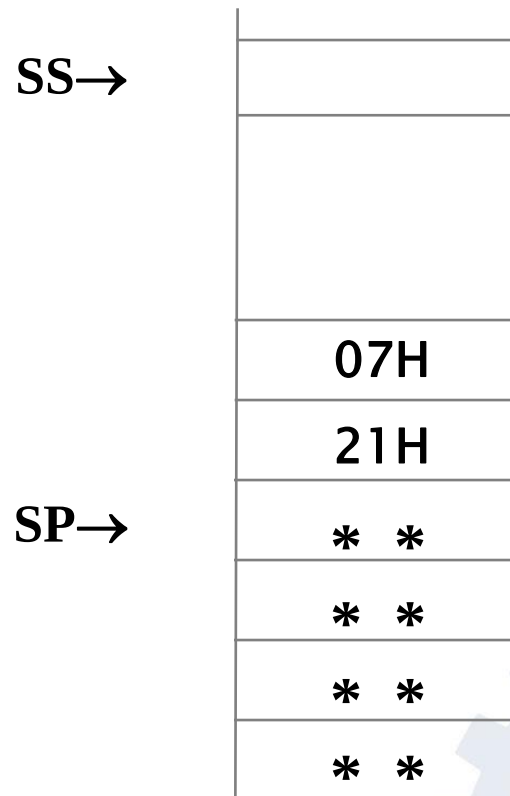




例： POP BX



POP BX 执行前



POP BX 执行后
BX = 2107H

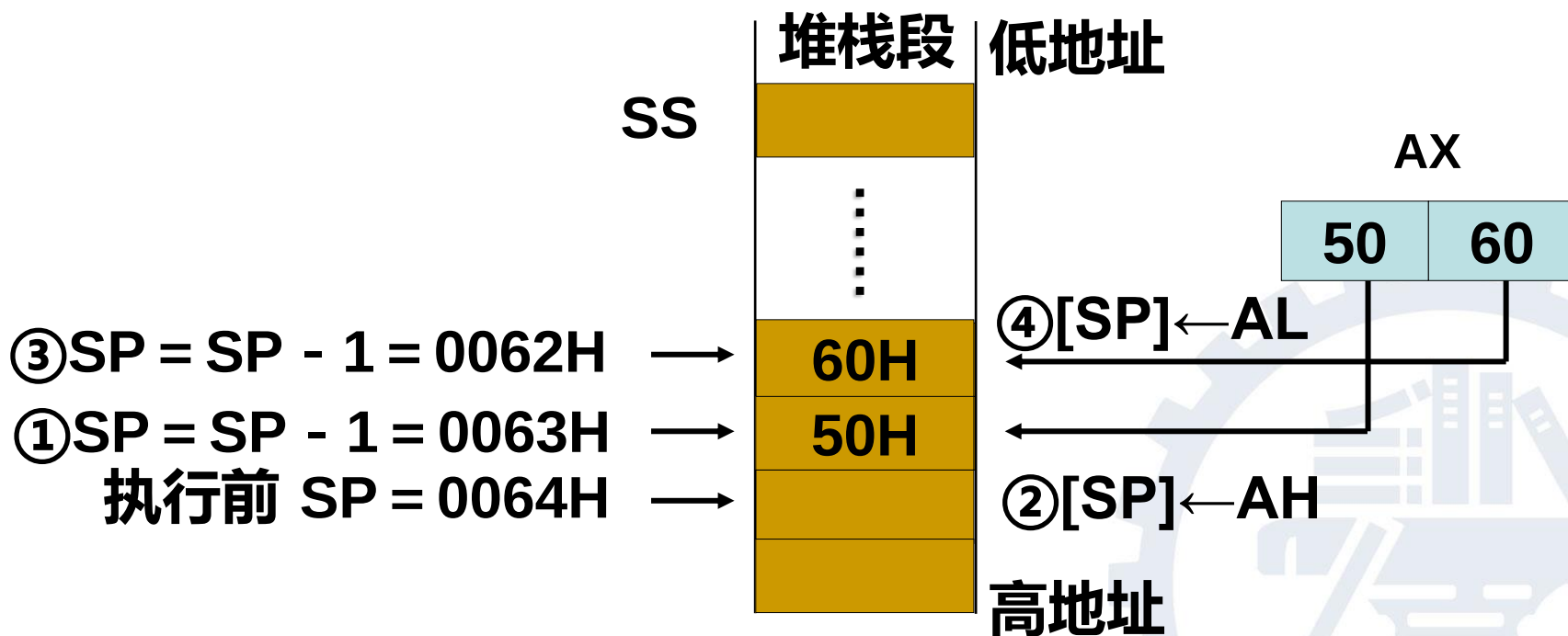
低地址

出栈方向

高地址

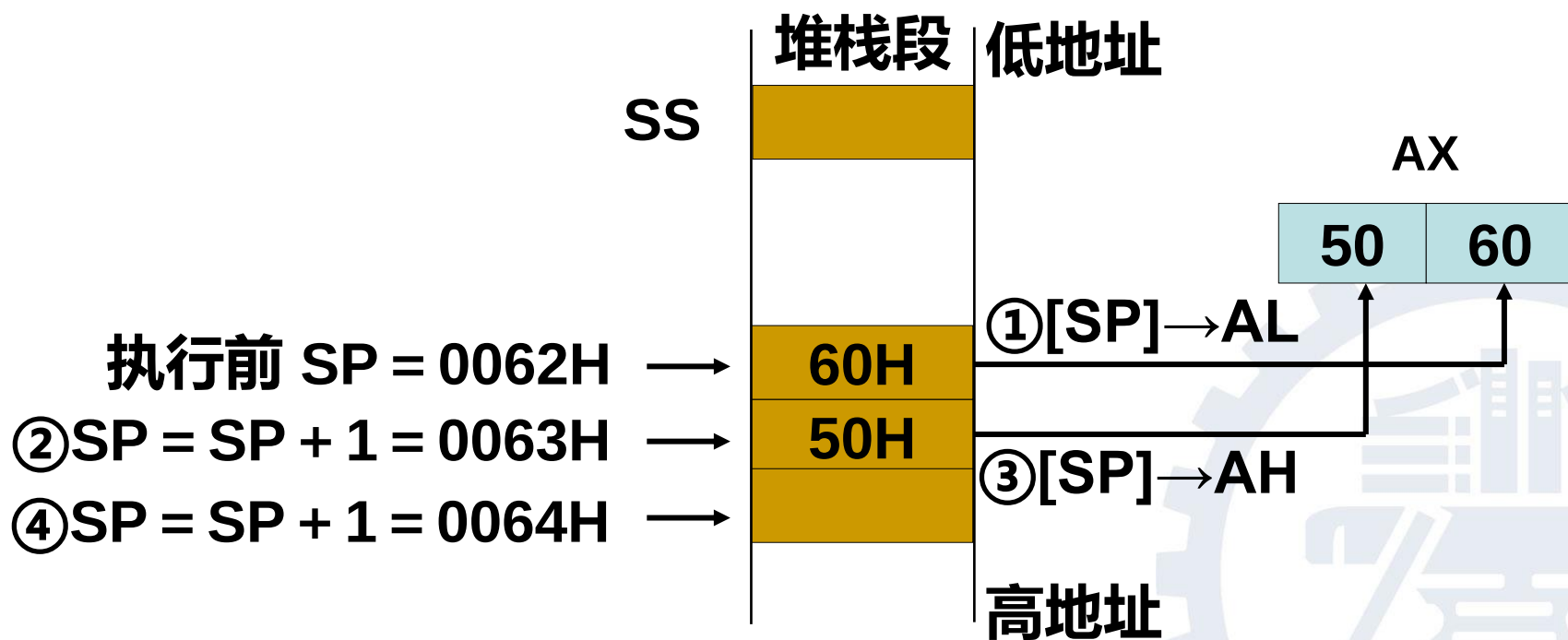


例： $AX=5060H$ ， $SP=0064H$ 。执行 **PUSH AX**
指令后，堆栈的情况如下：





例： $SP=0062H$ 。执行 **POP AX** 指令后，
堆栈的情况如下：





- **注意：不能将SS与栈底混为一谈！**
- **栈底指针说明：无明确寄存器来表示**
 - 如果程序中未定义SS段，则默认使用OS的堆栈段，此时栈底指针只有OS知道；
 - 如果程序中定义了SS段，则使用自定义的堆栈，此时栈底指针是用户程序可知的；
- **堆栈的两种访问方式：**
 - 随机访问方式：用BP寄存器来指示随机访问地址！
 - 先进后出访问方式：用SP寄存器来表示栈顶！



```
例:  PUSH DS
      SUB  AX, AX
      PUSH AX
      .....
      .....
      RET
```

```
例:  PUSH AX
      PUSH BX
      PUSH CX
      .....
      POP  CX
      POP  BX
      POP  AX
```

堆栈用于保存子程序返回地址和断点地址以及主程序通用寄存器内容的保护和恢复

因为堆栈遵从“后进先出”原则，在保存寄存器和恢复寄存器的内容时要按照相反的顺序执行一组压入和弹出指令。

**; 其间用到AX、BX、CX
; 后进先出**





3、数据交换指令 XCHG

交换指令: XCHG OPR1, OPR2 执行操作: $(OPR1) \leftrightarrow (OPR2)$

注意: * 不影响标志位

* 不允许使用段寄存器, 两个操作数必须有一个在寄存器中
(内存之间不能相互传送数据, 因为不知位宽是否匹配)

例: BX=6F30H, BP=0200H, SI=0046H, AX=2105H

SS=2F00H, (2F246H)=4154H

XCHG BX, [BP+SI] ; BX = 4154H

XCHG AL, BH ; AL= 41H



累加器专用传送指令

1、I/O数据传送指令 IN/OUT

- 用于外部设备I/O端口与CPU之间的信息交换。只能使用AX或AL来存放数据，只限8位立即数或DX来存放端口号；
- IN/OUT指令中AX（AL）操作数的位置是不同的，一个是目标操作数位置，另一个是源操作数位置；
- IN/OUT指令操作期间，端口地址处将产生一个负脉冲信号；



输入指令 IN (I/O → CPU)

I/O端口直接寻址方式

长格式: IN AL, PORT (字节)

IN AX, PORT (字)

执行操作: $AL \leftarrow (PORT)$ (字节)

$AX \leftarrow ((PORT+1), (PORT))$ (字)

I/O端口间接寻址方式

短格式: IN AL, DX (字节)

IN AX, DX (字)

执行操作: $AL \leftarrow (DX)$ (字节)

$AX \leftarrow ((DX+1), (DX))$ (字)





北京交通大学

BEIJING JIAOTONG UNIVERSITY



输出指令 OUT (CPU → I/O)

I/O端口直接寻址方式

长格式: OUT PORT, AL (字节)

OUT PORT, AX (字)

执行操作: (PORT) ← AL (字节)

((PORT+1), (PORT)) ← AX (字)

I/O端口间接寻址方式

短格式: OUT DX, AL (字节)

OUT DX, AX (字)

执行操作: (DX) ← AL (字节)

((DX+1), (DX)) ← AX (字)





注意

- * 不影响标志位
- * 前256个端口号00H~FFH可直接在指令中指定（长格式）
- * 如果端口号 ≥ 256 ，端口号 $\rightarrow$ DX（短格式）

例：IN AL, 28H ;将端口28H中的字节数据读出送AL
OUT 28H, AL ;将AL的数据写入28H中

例：MOV DX, 03FCH ; 将端口03FCH送DX
IN AL, DX ; 将端口03FCH中字节数读出送AL
OUT DX, AL ; 将AL中数据写入端口03FCH中

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



设备控制寄存器

例：Sound程序

端口61H



控制其它外部设备

2号定时器门控



放大器



```

mov    dx, 100
in     al, 61h
and    al, 11111100b
sound: xor    al, 2           ; 1→0→1
      out    61h, al         ; ON→OFF→ON
      mov    cx, 140h        ; 脉宽
Wait1: loop   wait1
      dec    dx
      jne    sound
    
```





2、换码/查表指令 XLAT 或 XLAT OPR

执行操作: $AL \leftarrow (BX + AL)$

功能: AL给出偏移量, 然后存储该偏移地址处的内容

例: MOV BX, OFFSET TABLE ; BX=0040H

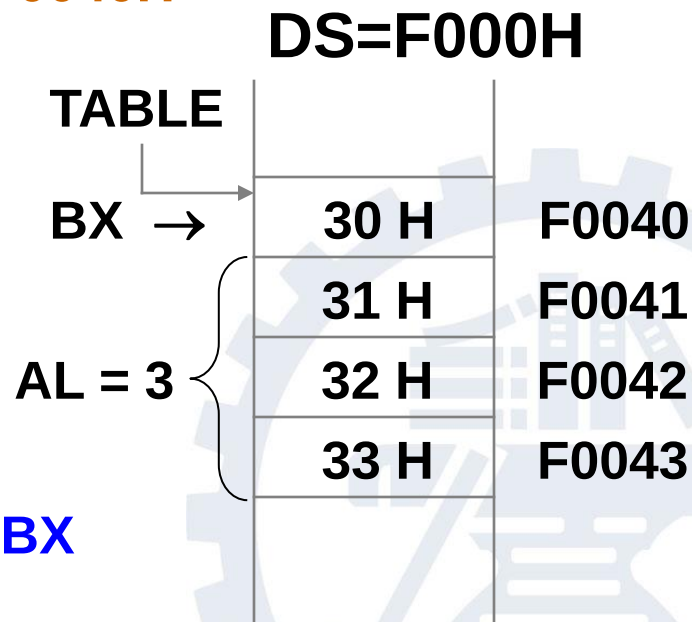
MOV AL, 3

XLAT TABLE

指令执行后 AL=33H

注意:

- * 不影响标志位
- * 字节表格(长度不超过256) 首地址 $\rightarrow$ BX
- * 需转换的位移量 $\rightarrow$ AL





地址传送指令

1、有效地址送寄存器指令：LEA REG, SRC

执行操作：REG ← SRC

功能：将源操作数的有效地址EA传送到目标寄存器

注意：SRC不能是立即寻址，寄存器寻址

2、指针送寄存器和DS指令：LDS REG, SRC

执行操作：REG ← (SRC) DS ← (SRC+2)

功能：相继二字 → 寄存器、DS

3、指针送寄存器和ES指令：LES REG, SRC

执行操作：REG ← (SRC) ES ← (SRC+2)

功能：相继二字 → 寄存器、ES



例:

TABLE		MOV BX, TABLE	; BX=0040H
(DS):1000H	40 H	MOV BX, OFFSET TABLE	; BX=1000H
	00 H	LEA BX, TABLE	; BX=1000H
	00 H	LDS BX, TABLE	; BX=0040H
	30 H		; DS=3000H
		LES BX, TABLE	; BX=0040H
			; ES=3000H

注意:

- * 不影响标志位
- * REG 不能是段寄存器
- * SRC 必须为存储器寻址方式



比 较

- | | | | |
|---|-----|-------------|----------------------|
| ① | LEA | BX, [2728H] | ;BX=2728H |
| ② | MOV | BX, [2728H] | ;BX=DS段内2728H处一个字的内容 |
| ③ | MOV | BX, 2728H | ;BX=2728H |
| ④ | LDS | BX, [2728H] | ;BX=DS段内2728H处一个字的内容 |
| ⑤ | LEA | BX, 2728H | × |
| ⑥ | LDS | BX, 2728H | × |
| ⑦ | LES | BX, 2728H | × |
| ⑧ | LEA | BX, SI | × |
| ⑨ | LDS | BX, SI | × |
| ⑩ | LES | BX, SI | × |





操作数的寻址方式总结

立即数寻址

MOV AX , 3069H

寄存器寻址

MOV AL , BH

直接寻址

MOV AX , [2000H]

寄存器间接寻址

MOV AX , [BX]

寄存器相对寻址

MOV AX , NUM [SI]

基址变址寻址

MOV AX , [BP][DI]

相对基址变址寻址

MOV AX , MASK [BX][SI]





标志寄存器传送指令

标志送AH指令: LAHF

执行操作:

$(AH) \leftarrow (\text{FLAGS的低位字节})$

AH送标志寄存器指令: SAHF

执行操作:

$(\text{FLAGS的低位字节}) \leftarrow (AH)$

标志进栈指令: PUSHF

(FLAGS的**高和低位两字节**)

执行操作:

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (\text{FLAGS})$

标志出栈指令: POPF

(FLAGS的**高和低位两字节**)

执行操作:

$(\text{FLAGS}) \leftarrow ((SP)+1, (SP))$

$(SP) \leftarrow (SP) + 2$

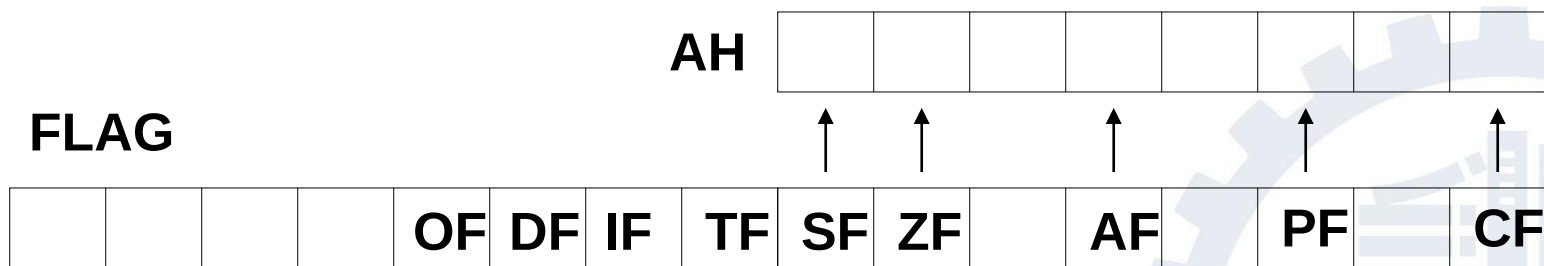
• 影响标志位

• **FLAGS为16位标志寄存器**



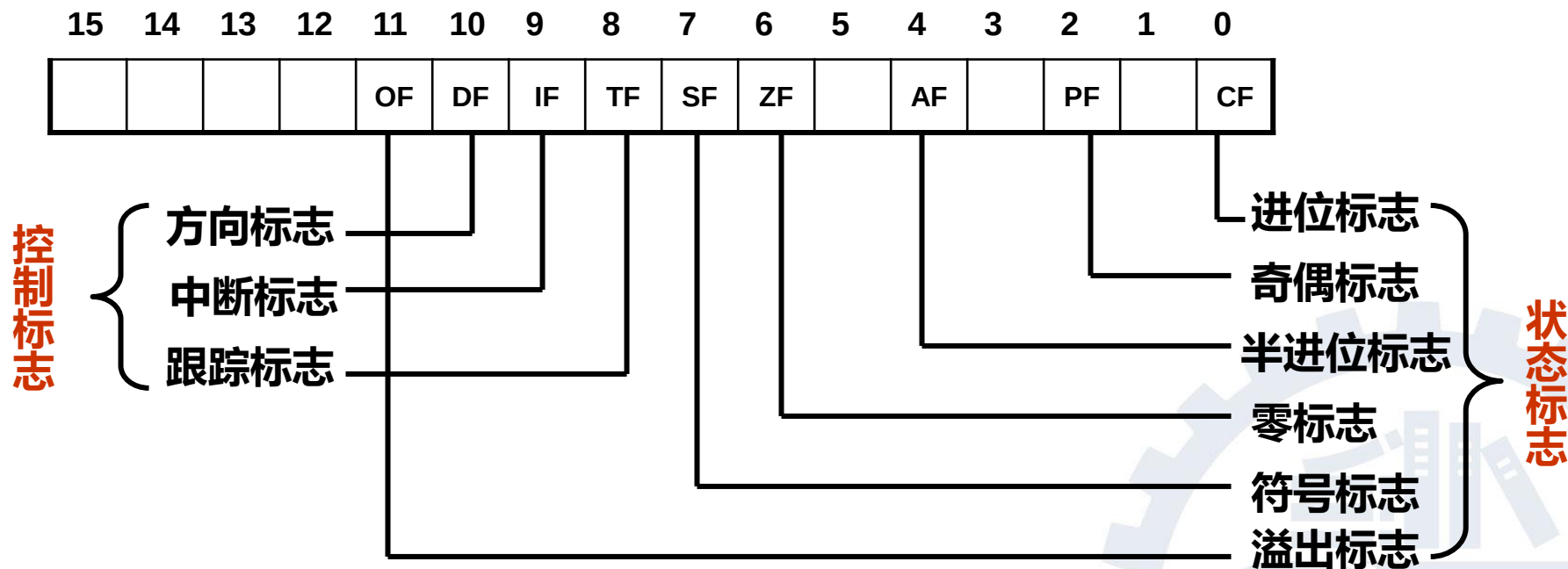
LAHF指令

该指令功能是把标志寄存器中的SF、ZF、AF、PF、CF五个标志传至AH的第7、6、4、2、0位，第5、3、1位未定义。





标志寄存器的格式及各位的含义：





算术指令

- 加法指令 ADD、ADC、INC
- 减法指令 SUB、SBB、DEC、NEG、CMP
- 乘法指令 MUL、IMUL
- 除法指令 DIV、IDIV
- 十进制调整指令
DAA、DAS、AAA、AAS、AAM、AAD



1、加法指令

加法指令: ADD DST, SRC

执行操作: $(DST) \leftarrow (SRC) + (DST)$

带进位加法指令: ADC DST, SRC

执行操作: $(DST) \leftarrow (SRC) + (DST) + CF$

加1指令: INC OPR

执行操作: $(OPR) \leftarrow (OPR) + 1$

注意:

- * 除INC指令不影响CF标志外(硬件结构决定的), 均对条件标志位有影响。



加法指令对条件标志位的影响

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

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

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

$$OF = \begin{cases} 1 & \text{两个操作数符号相同, 而结果符号与之相反} \\ & \text{(或者说 操作结果错误)} \\ 0 & \text{否则} \end{cases}$$

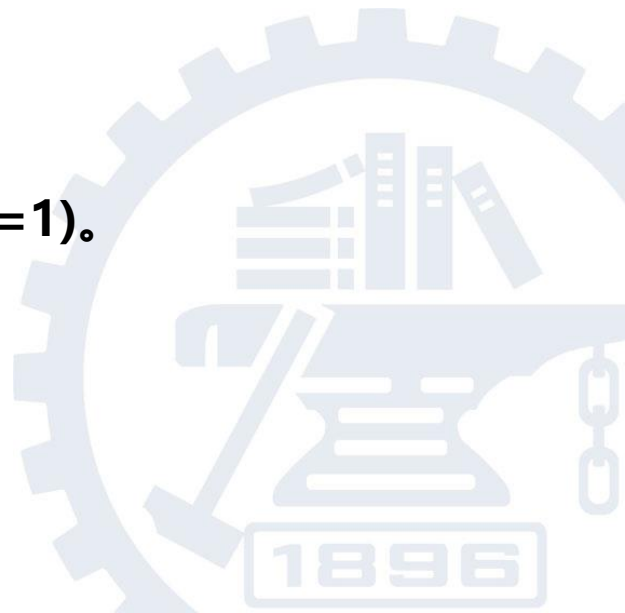
CF 位表示无符号数相加的进位与否

OF 位表示带符号数相加结果的正确与否



关于有符号数加减法指令中的OF

- 1、计算机中，当确定为有符号数运算时，**符号数一律用补码表示，运算时符号位和数字位一起参加运算。**
- 2、在加法/减法运算中的OF含义说明：
 - 不溢出 ($OF = 0$)，说明运算结果正确。
 - 溢出 ($OF = 1$)，说明运算结果错误。
- 3、在加法/减法运算中的溢出情况总结：
 - 异号数相加或同号数相减不会溢出($OF=0$)。
 - 同号数相加或异号数相减时有可能发生溢出($OF=1$)。



如果要将一个加法解释成无符号数加法，那么就仅使用CF，否则就需要使用OF；CF与OF不在同一解释中使用！

举例: n=8 bit 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

0 0 0 0 0 1 1 1

+ 1 1 1 1 1 0 1 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

1 0 0 0 0 1 1 1

+ 1 1 1 1 0 1 0 1

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 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



- 汇编器不会区分有符号还是无符号然后用两个标准来处理，它**统统当作有符号的！并且全部编译成补码形式！**
- **有符号运算中的溢出是一种出错**，在程序中应当尽量避免，当有溢出时，也应能尽快发现，否则程序再运行下去，其结果便毫无意义。
- 为此 8086 指令中专门提供了一条**溢出中断指令INTO**，用来**判断有符号数加减运算是否有溢出**。该命令常用于算术运算中，若算术运算（它的上一条指令）的结果产生溢出，即 $OF=1$ ，则立即调用一个处理算术溢出的中断服务程序；否则不进行任何操作，接着执行下一条指令。



例：双精度数的加法

DX = 0002H AX = 0F365H

BX = 0005H CX = 8100H

指令序列 ADD AX, CX ; (1)

 ADC DX, BX ; (2)

(1) 执行后, AX= 7465H

 CF=1 (OF=1)

(2) 执行后, DX = 0008H

 CF=0 (OF=0)





2、减法指令

减法指令: SUB DST, SRC
执行操作: $(DST) \leftarrow (DST) - (SRC)$

带借位减法指令: SBB DST, SRC
执行操作: $(DST) \leftarrow (DST) - (SRC) - CF$

减1指令: DEC OPR
执行操作: $(OPR) \leftarrow (OPR) - 1$

求补指令: NEG OPR
执行操作: $(OPR) \leftarrow -(OPR)$

比较指令: CMP OPR1, OPR2
执行操作: $(OPR1) - (OPR2)$

注意:

除DEC指令不影响CF标志外
(硬件结构决定的), 均对
条件标志位有影响。



减法指令对条件标志位 (CF/OF/ZF/SF) 的影响:

$CF = \begin{cases} 1 & \text{被减数的最高有效位有向高位的借位} \\ 0 & \text{否则} \end{cases}$

$OF = \begin{cases} 1 & \text{两个操作数符号相反, 而结果的符号与被减数相反} \\ 0 & \text{否则} \end{cases}$

有符号减法下CF是没有意义的
无符号数减法下OF是没有意义的





求补指令

指令格式: NEG OPR

执行操作 $(\text{OPR}) \leftarrow -(\text{OPR})$

即将操作数按位**求反后末位加1**，因而执行的操作也可以表示为： $(\text{OPR}) \leftarrow 0\text{FFFFH} - (\text{OPR}) + 1$

例：

如果CL=100，执行NEG CL后，

CL=1001 1100B，是-100的补码

如果CL=-8，执行NEG CL后，

CL=0000 1000B，是8的补码





比较指令

指令格式: `CMP OPR1, OPR2`

执行操作 $(OPR1) - (OPR2)$

- 该指令与SUB指令一样执行减法操作，但它并不保存结果，只是根据结果设置条件标志位。
- CMP指令后往往跟一条条件转移指令，根据比较结果产生不同的程序分支。

说明: OPRT1和OPRT2可以是寄存器或存储器，但不能同时为存储器，OPRT2还可以为立即数。



CMP指令执行后对标志位的影响

操作数类型	CF	ZF	SF	OF	两操作数的关系
带符号的 二进制	0	1	0	0	等于
	-	0	1	0	小于
	-	0	0	1	负数减正数得正数时的：小于,如1001-0111=0010
	-	0	0	0	大于
	-	0	1	1	正数减负数得负数时的：大于, 如0001-1001=1000
不带符号的 二进制	0	1	0	0	等于
	1	0	-	-	目的操作数 低于 源操作数
	0	0	-	-	高于



3、乘法指令

无符号数乘法指令：MUL SRC

带符号数乘法指令：IMUL SRC

执行操作：

字节操作数 $AX \leftarrow AL * (SRC)$

字操作数 $(DX, AX) \leftarrow AX * (SRC)$

注意：

- * AL (AX) 为隐含的乘数寄存器。
- * AX (DX, AX) 为隐含的乘积寄存器。
- * SRC不能为立即数，**因为位宽不详**。
- * 除CF和OF外，对条件标志位无定义。



乘法指令对 CF/OF 的影响:

MUL指令: $CF, OF = \begin{cases} 00 & \text{乘积的高一半为零} \\ 11 & \text{否则} \end{cases}$

IMUL指令: $CF, OF = \begin{cases} 00 & \text{乘积的高一半是低一半的符号扩展} \\ 11 & \text{否则} \end{cases}$

※ 乘法指令不会产生溢出和进位，这时用OF和CF位来表示乘积有效数字的长度：

※ 若乘积的高半部分（字节乘法为AH，字乘法为DX）有效，即：

- ✓ MUL指令中是指AH或DX中的内容不为0
- ✓ IMUL指令指的则是AH或DX中的内容不是符号位的扩展

则CF和OF都为1，表示DX或AH中含有乘积的有效数字，否则CF和OF为0。



乘法指令对 CF/OF 的影响:

MUL指令: $CF, OF = \begin{cases} 00 & \text{乘积的高一半为零} \\ 11 & \text{否则} \end{cases}$

IMUL指令: $CF, OF = \begin{cases} 00 & \text{乘积的高一半是低一半的符号扩展} \\ 11 & \text{否则} \end{cases}$

例: $(AX) = 16A5H, (BX) = 0611H$

(1) IMUL BL ; $(AX) \leftarrow (AL) * (BL)$
; $(AX) = 0F9F5H$ CF=OF=1

(2) MUL BX ; $(DX, AX) \leftarrow (AX) * (BX)$
; $(DX)=0089H (AX)=5EF5H$ CF=OF=1



逻辑指令

- 逻辑运算指令

AND、OR、NOT、XOR、TEST

- 移位指令

SHL、SHR、SAL、SAR、

ROL、ROR、RCL、RCR



1、逻辑运算指令

逻辑非指令: NOT OPR

执行操作: $(OPR) \leftarrow \neg (OPR)$

* OPR不能为立即数

* 不影响标志位

逻辑与指令: AND DST, SRC

执行操作: $(DST) \leftarrow (DST) \wedge (SRC)$

逻辑或指令: OR DST, SRC

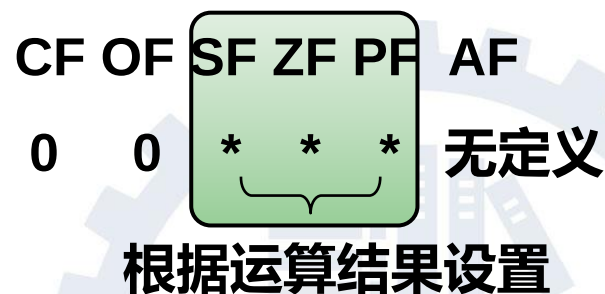
执行操作: $(DST) \leftarrow (DST) \vee (SRC)$

异或指令: XOR DST, SRC

执行操作: $(DST) \leftarrow (DST) \vee (SRC)$

测试指令: TEST OPR1, OPR2

执行操作: $(OPR1) \wedge (OPR2)$





AL中的数

例：屏蔽AL的第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的第0、1位变反

XOR AL, 3

```

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

```

```

      * * * * * 1 0
    
```

例：测试某些位是0是1

TEST AL, 1
JZ EVEN

```

      * * * * *
AND  0 0 0 0 0 0 0 1
-----
      0 0 0 0 0 0 0 *
    
```



2、移位指令

逻辑左移 **SHL** OPR, CNT



将 (OPR) 向左移动CNT指定的次数, 最低位补入 0, CF的内容为最后移入位的值; CNT=1, 移1位; CNT > 1, 放在CL

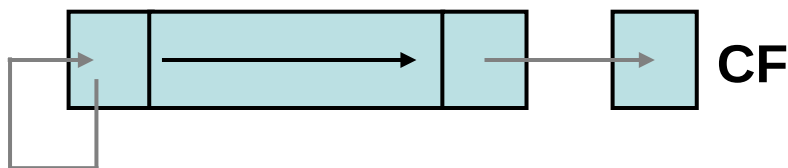
逻辑右移 **SHR** OPR, CNT



将 (OPR) 向右移动CNT规定的次数, 最高位补入相应个数的 0, CF的内容为最后移入位的值

算术左移 **SAL** OPR, CNT (同SHL)

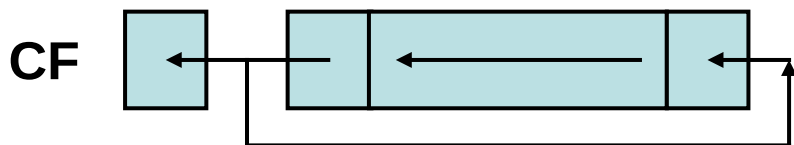
算术右移 **SAR** OPR, CNT



将 (OPR) 向右移动CNT指定的次数且**最高位保持不变**, 但参与移位; CF的内容为最后移入位的值

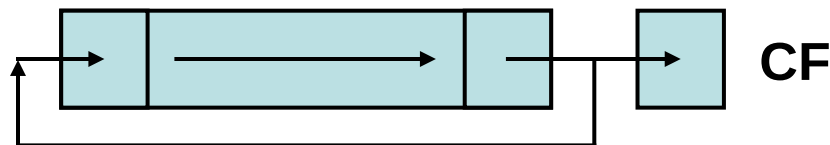


循环左移 ROL OPR, CNT



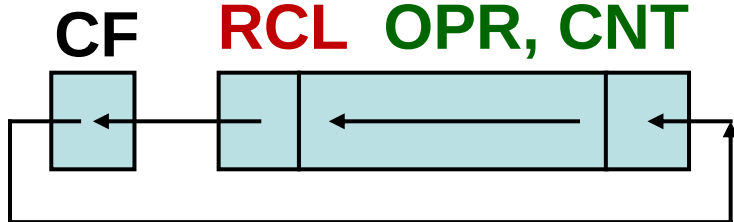
将目的操作数的最高位与最低位连成一个环，将环中的所有位一起向左移动CNT规定的次数。CF的内容为最后移入位的值

循环右移 ROR OPR, CNT

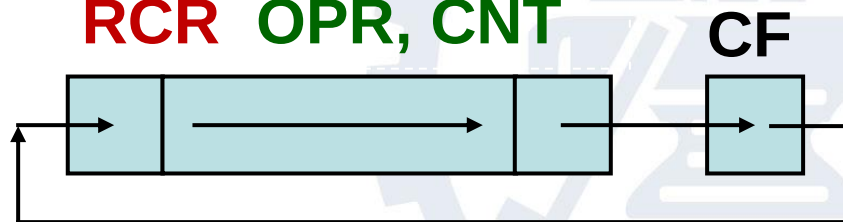


将目的操作数的最高位与最低位连成一个环，将环中的所有位一起向右移动CNT规定的次数，CF的内容为最后移入位的值

带进位循环左移 RCL OPR, CNT



带进位循环右移 RCR OPR, CNT





注意:

- * OPR可用除立即数以外的任何寻址方式

- * 条件标志位:

CF = 移入的数值

$$OF = \begin{cases} 1 & \text{CNT}=1\text{时, 最高有效位的值发生变化} \\ 0 & \text{CNT}=1\text{时, 最高有效位的值不变} \end{cases}$$

CNT>1时, OF无意义

移位指令: SF、ZF、PF 根据移位结果设置, AF无定义

循环移位指令可以改变操作数中所有位的位置, 不影响 SF、ZF、PF、AF

- * 算术/逻辑移位指令常用来乘以2或除以2, 以及2的整数幂, 根据移位的位数, 即移动1、2、3..., 分别为乘以或除以 2^1 、 2^2 、 2^3 ...
- * 算术移位适用于**带符号数**的运算; 逻辑移位指令用于**无符号数**的运算



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
```

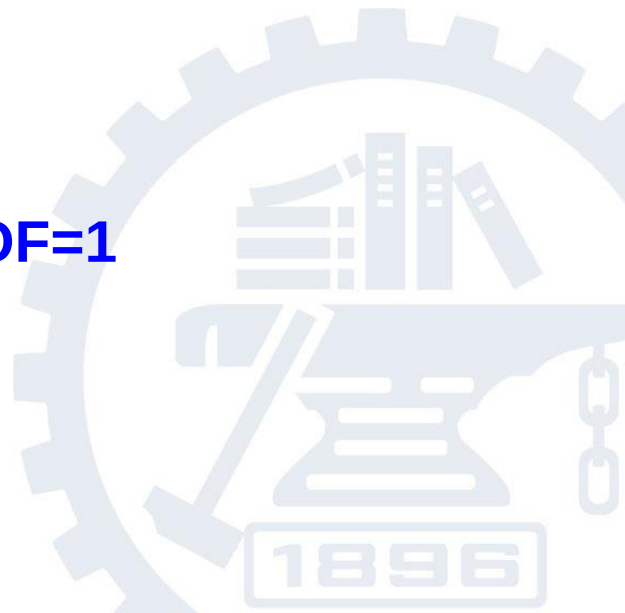
(2) BX 为带符号数，求 BX × 2

```
SAL BX, 1 ; BX = 09E0H, OF=1
```

(3) BX 为带符号数，求 BX / 4

```
MOV CL, 2
```

```
SAR BX, CL ; BX = 0E13CH
```





控制转移指令

- 无条件转移指令 JMP

- 条件转移指令

JZ / JNZ、JE / JNE、JS / JNS、JO / JNO、
JP / JNP、JB / JNB、JL / JNL、JBE / JNBE、
JLE / JNLE、JCXZ

- 循环指令

LOOP、LOOPZ / LOOPE、LOOPNZ / LOOPNE

- 子程序调用和返回指令 CALL、RET

- 中断与中断返回指令 INT、INTO、IRET



1、无条件转移指令

段内直接短转移: JMP SHORT OPR

执行操作: $IP \leftarrow IP + 8\text{位位移量}$

段内直接近转移: JMP NEAR PTR OPR

执行操作: $IP \leftarrow IP + 16\text{位位移量}$

段内间接转移: JMP WORD PTR OPR

OPR为存放在16位通用寄存器或字内存单元内的EA

执行操作: $IP \leftarrow (EA)$





1、无条件转移指令

例

```
mov ax, 0
jmp next01 ; 无条件转移到标号为next01
...        ; 的语句，即修改指令指针，
...        ; 指向标号next01
next01:    mov cx, 100
...
```





2、条件转移指令

将标志位的状态作为测试的条件，所以首先需要执行影响标志位的指令（如CMP或TEST等），然后采用条件转移指令测试相应的标志，决定程序的转移

注意：

只能使用段内直接寻址的8位位移量；

条件转移指令均为段内短转移，即转移范围为：-128 ~ +127



(1) 根据单个条件标志的设置情况转移

格式	测试条件
JZ(JE) OPR	ZF = 1
JNZ(JNE) OPR	ZF = 0
JS OPR	SF = 1
JNS OPR	SF = 0
JO OPR	OF = 1
JNO OPR	OF = 0
JP OPR	PF = 1
JNP OPR	PF = 0
JC OPR	CF = 1
JNC OPR	CF = 0



回顾：CMP指令执行后对标志位的影响

操作数类型	CF	ZF	SF	OF	两操作数的关系
带符号的 二进制	0	1	0	0	等于
	-	0	1	0	小于
	-	0	0	1	负数减正数得正数时的：小于,如 1001-0111=0010
	-	0	0	0	大于
	-	0	1	1	正数减负数得负数时的：大于, 如 0001-1001=1000
	-	0	1	1	
不带符号的 二进制	0	1	0	0	等于
	1	0	-	-	目的操作数 低于 源操作数
	0	0	-	-	高于



(2) 比较两个无符号数，并根据比较结果转移

说明	格式	测试条件
<	JB (JNAE,JC) OPR	CF = 1
≥	JNB (JAE,JNC) OPR	CF = 0
≤	JBE (JNA) OPR	CF ∨ ZF = 1
>	JNBE (JA) OPR	CF ∨ ZF = 0

* 适用于地址（为无符号数）或双精度数低位字（为无符号数）的比较



(3) 比较两个带符号数，并根据比较结果转移

说明	格式	测试条件
<	JL (JNGE) OPR	$SF \vee OF = 1$
$\geq$	JNL (JGE) OPR	$SF \vee OF = 0$
$\leq$	JLE (JNG) OPR	$(SF \vee OF) \vee ZF = 1$
>	JNLE (JG) OPR	$(SF \vee OF) \vee ZF = 0$

- * 参考带符号数的CMP指令或减法指令对SF/OF的影响
- * 适用于带符号数的比较



北京交通大学
BEIJING JIAOTONG UNIVERSITY



(4) 测试 CX 的值为 0 则转移

格式

JCXZ OPR

测试条件

(CX)=0





指令名	转移条件	说明
JZ/JE	ZF=1	相等/为零转移(=)
JNZ/JNE	ZF=0	不相等/不为零转移($\neq$)
JS	SF=1	为负转移
JNS	SF=0	为正转移
JO	OF=1	溢出转移
JNO	OF=0	不溢出转移
JP/JPE	PF=1	偶数个 1 转移
JNP/JPO	PF=0	奇数个 1 转移
JB/JBAE/JC	CF=1	低于转移(<)
JNB/JAE/JNC	CF=0	不低于转移($\geq$)
JBE/JNA	CF=1或ZF=1	不高于转移($\leq$)
JNBE/JA	CF=0且ZF=0	高于转移(>)
JL/JNGE	SF$\neq$OF	小于转移(<)
JNL/JGE	SF=OF	不小于转移(>)
JLE/JNG	(SF$\neq$OF)且ZF=1	不大于转移($\leq$)
JNLE/JG	(SF=OF)且ZF=0	大于转移(>)

无符号数

A: 高于

B: 低于

E: 等于

有符号数

G: 大于

L: 小于

E: 等于



3、循环指令

注意

- * CX 中存放循环次数
- * 只能使用段内直接寻址的 8 位位移量

LOOP



无条件循环指令

LOOPZ / LOOPE

LOOPNZ / LOOPNE



条件循环指令

执行步骤:

(1) $(CX) \leftarrow (CX) - 1$

(2) 检查是否满足测试条件, 如满足则 $(IP) \leftarrow (IP) + 8\text{位位移量}$, 实行循环; 不满足则 IP 不变, 退出循环。



循环指令: LOOP OPR

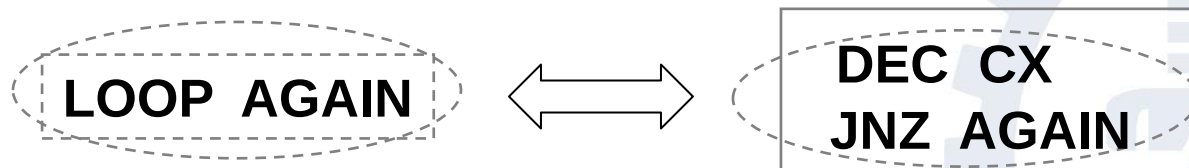
测试条件: (CX) $\neq$ 0

为零或相等时循环指令: LOOPZ(LOOPE) OPR

测试条件: ZF=1 且 (CX) $\neq$ 0

不为零或不相等时循环指令: LOOPNZ(LOOPNE) OPR

测试条件: ZF=0 且 (CX) $\neq$ 0



AGAIN是一个地址标号



例：在多重循环的程序结构中，CX 计数器的保存和恢复；
堆栈段的作用此时就很明显！

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

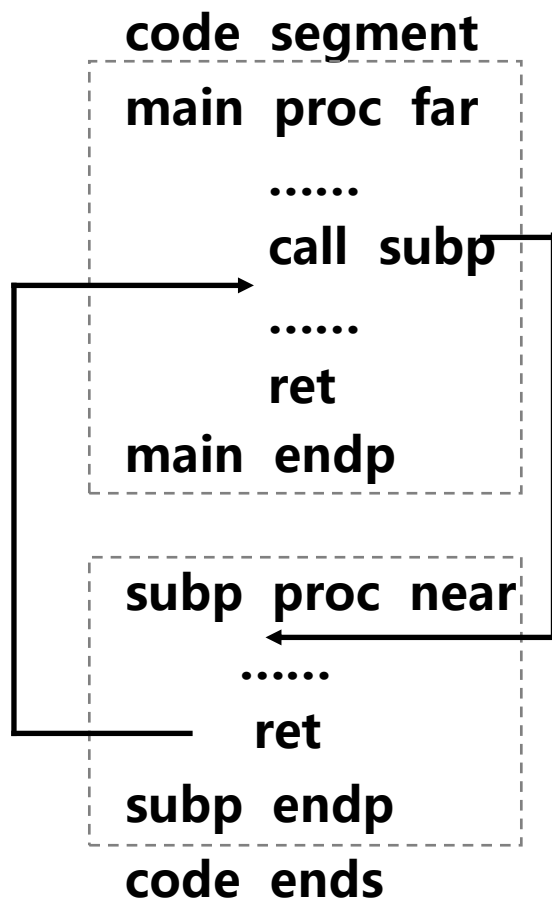
```
        MOV  DI, M
AGAIN: .....
        MOV  CX, N
NEXT: .....
        LOOP NEXT
        .....
        DEC  DI
        JNZ  AGAIN
```



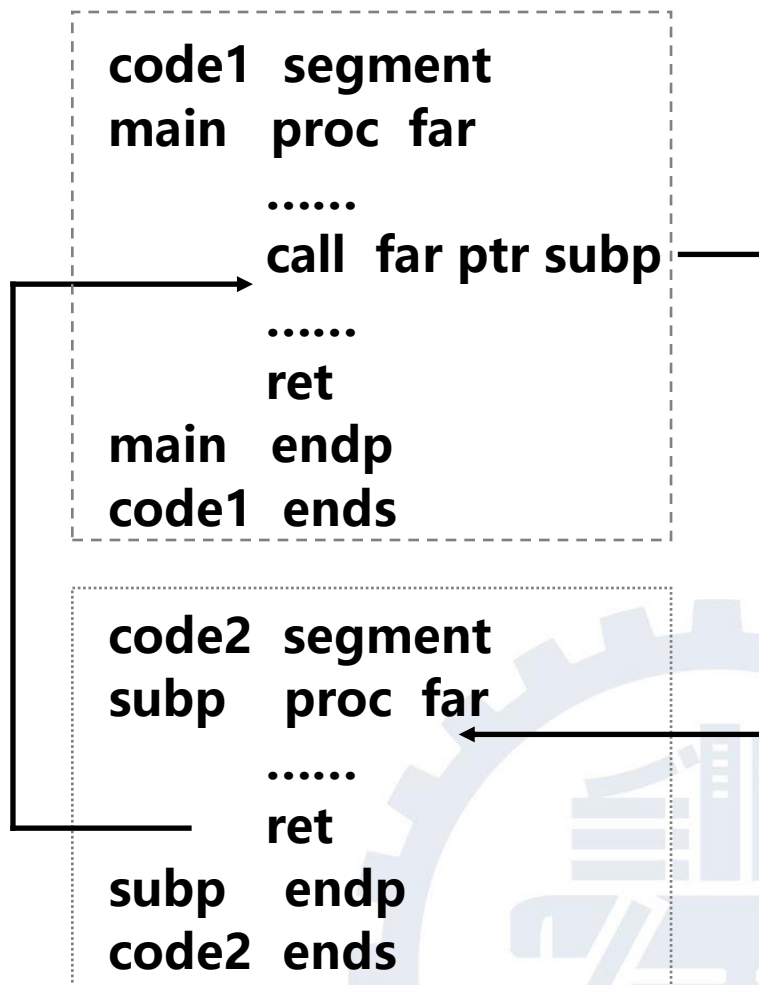

4、子程序调用和返回指令

执行过程

- 保护断点；
 - 将调用指令的下一条指令的地址（断点）压入堆栈
- 获取子过程的入口地址；
 - 子过程第1条指令的偏移地址
- 执行子过程，含相应参数的保存及恢复；
- 恢复断点，返回原程序。
 - 将断点偏移地址由堆栈弹出



段内调用和返回



段间调用和返回



CALL 调用指令

段内直接近调用: CALL DST (DST为子程序名)

执行操作: $(SP) \leftarrow (SP) - 2$;断点压入堆栈

$((SP)+1, (SP)) \leftarrow (IP)$; 主程序地址压栈

$(IP) \leftarrow (IP) + 16$ 位位移量

注意: IP为Call指令的下一条指令的地址, 其与DST子程序的地址间的相对地址或位移量是固定的。

段内间接近调用: CALL DST

(DST为寄存器如BX或存储器地址 WORD PTR [BX])

执行操作: $(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (IP)$

$(IP) \leftarrow (EA)$ (**DST为内存地址**)



北京交通大学

BEIJING JIAOTONG UNIVERSITY



RET 返回指令

段内近返回: RET

功能: 将堆栈中保存的2字节断点的偏移地址恢复IP中, CS不变

执行操作: $(IP) \leftarrow ((SP)+1, (SP))$
 $(SP) \leftarrow (SP) + 2$

段内带立即数近返回: RET EXP

EXP表示弹出断点之后, 使SP内容再回退EXP个字节单元, 作用是使断点之后的EXP个字节单元数据失效



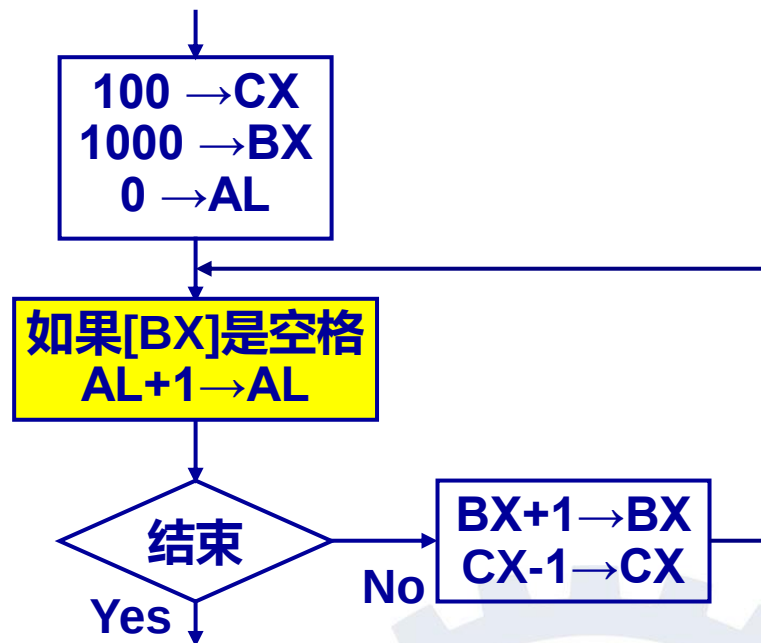


例

在数据段偏移量为1000H处连续存放着一个长度为100的字符串（ASCII码），查找其中空格的个数，将结果存放在AL寄存器。

主程序

```
mov cx, 100  
mov bx, 1000h  
xor al, al  
p01: call s01  
inc bx  
loop p01
```



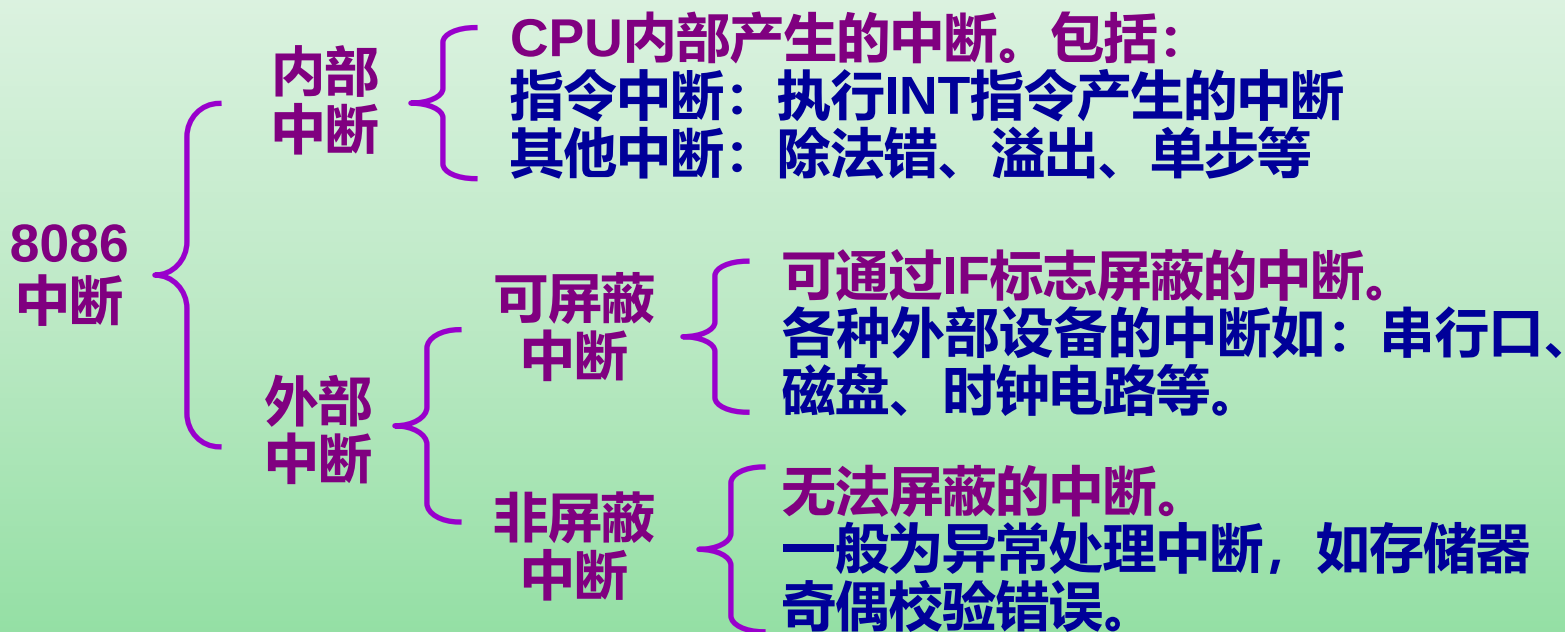
子程序

```
s01:  CMP byte ptr [bx], 20h  
      jnz p02  
      inc al  
p02:  ret
```



5、中断与中断返回指令

- 8086中断系统最大支持256个中断，每个中断具有唯一的**中断号**，且具有独立的中断服务程序。发生中断时CPU当前程序被暂停转而执行中断处理程序。





中断向量:

中断例行程序的入口地址，存放于中断向量区，指向中断服务程序的段值和偏移量 (CS:IP)

中断发生时，从中断向量表中取出相应的值赋值给CS和IP的过程是由系统硬件自动完成的，程序员不可见！程序员能够做的就是修改中断向量表中存储的值。

在8086中，reset不算是中断，不在中断向量表内。reset时CS=FFFFH，IP=0000H；在内存FFFF0处存放了一个JMP指令！

00000	类型0的(IP)	}	0号(除法错误)
	类型0的(CS)		
00004	类型1的(IP)	}	1号(调试)
	类型1的(CS)		
4*N	类型N的(IP)	}	4号(溢出)
	类型N的(CS)		
003FC	类型255的(IP)	}	
	类型255的(CS)		

中断向量表



中断过程

保存标志: (FLAGS)→堆栈

保存当前地址: CS、IP→堆栈

进入中断程序: $[n \times 4] \rightarrow IP, [n \times 4 + 2] \rightarrow CS$

:

中断返回: 堆栈→IP, 堆栈→CS

恢复标志: 堆栈→(FLAGS)

中断指令格式: INT TYPE 或 INT 隐含的类型号为3

中断返回指令: IRET

注意

- * TYPE (0~255) 是中断类型号
- * INT 指令还把 IF 和 TF 置0, 但不影响其它标志位
- * IRET 指令执行完, 标志位由堆栈中取出的值确定



INT指令是一种隐指令

- CPU在执行INT指令时，经过某些操作，转去执行**中断服务程序**。这些操作是由硬件直接实现的，把它称为**中断隐指令**，其操作过程程序员不可见。
- 中断隐指令并不是指令系统中的一条真正的指令，它没有操作码，所以中断隐指令是一种特殊指令。





中断指令格式: **INT TYPE 或 INT**

隐含的类型号为3

隐含的操作:

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (FLAGS)$

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (CS)$

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (IP)$

$(IP) \leftarrow (TYPE * 4)$

$(CS) \leftarrow (TYPE * 4 + 2)$

溢出中断指令: **INTO**

执行操作:

若 $OF=1$,



$(IP) \leftarrow (10H)$

$(CS) \leftarrow (12H)$

类型4软中断
在中断向量表
中的地址



从中断返回指令：IRET

执行操作：

$$(IP) \leftarrow ((SP)+1, (SP))$$
$$(SP) \leftarrow (SP) + 2$$
$$(CS) \leftarrow ((SP)+1, (SP))$$
$$(SP) \leftarrow (SP) + 2$$
$$(FLAGS) \leftarrow ((SP)+1, (SP))$$
$$(SP) \leftarrow (SP) + 2$$

注意：

- TYPE (0~255) 是中断类型号
- INT 指令还把 IF 和 TF 置0，但不影响其它标志位
- * IRET 指令执行完，标志位由堆栈中取出的值确定



处理机控制指令

控制CPU工作方式的指令，多用于多任务处理

1、标识处理指令

CLC、STC、CMC：对CF位清零/置位/取反

CLD、STD：字符串移动方向增/减标志

CLI、STI：开/关中断





2、其他处理机控制与杂项操作指令

- **空操作指令NOP**

该指令没有的显式操作数，主要起**延迟下一条指令的执行**，不影响任何标志位。

- **等待指令WAIT**

该指令使**CPU处于等待状态，直到协处理器(Coprocessor)完成运算，并用一个重启信号唤醒CPU为止**。该指令的执行不影响任何标志位。*协处理器不算外设，而是另一个FPU！



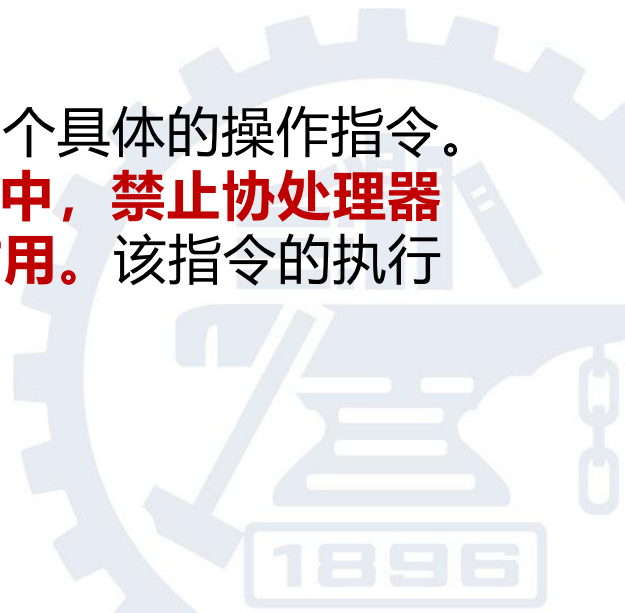
- **暂停指令HLT**

在等待中断信号时，该指令使CPU处于暂停工作状态【低功耗模式】，CS:IP指向下一条待执行的指令，但不进行任何操作，保持先前操作中所有寄存器的状态不变。当产生了中断信号，CPU把CS和IP压栈，并转入中断处理程序。在中断处理程序执行完后，中断返回指令IRET弹出IP和CS，并唤醒CPU执行下条指令。指令的执行不影响任何标志位。

- **封锁数据指令LOCK**

该指令是一个前缀指令形式，在其后面跟一个具体的操作指令。**LOCK指令可以保证是在其后指令执行过程中，禁止协处理器修改数据总线上的数据，起到独占总线的作用。**该指令的执行不影响任何标志位。

指令的格式：LOCK INSTRUCTION





1、指出下列指令的错误

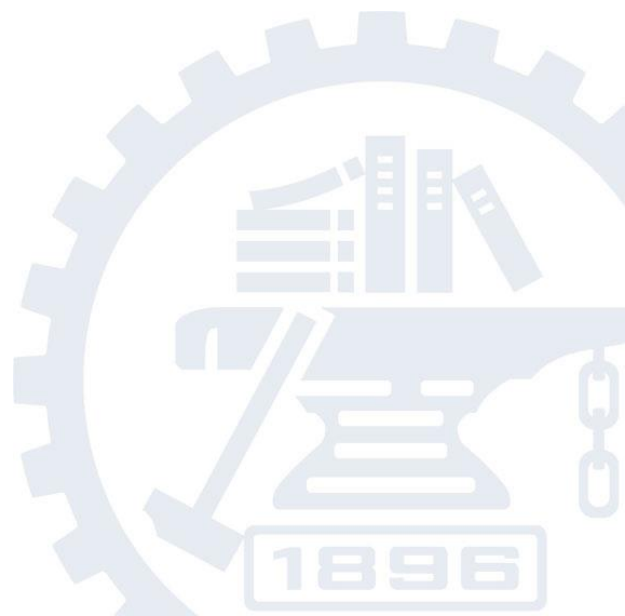
- (1) MOV AH,BX
- (2) MOV [BX],[SI]
- (3) MOV AX,[DI][SI]
- (4) MOV MYDAT[BX][SI],ES:AX
- (5) MOV CS,AX



作业

2、下列哪些指令是非法的

- (1) CMP 15,BX
- (2) CMP BYTE PTR OP1,25
- (3) CMP OP1,OP2
- (4) CMP AX,OP1





3、假设下列指令中的所有标示符均为类型属性为字的变量，请指出下列指令中哪些是非法的？它们的错误是什么？

- (1) MOV BP,AL
- (2) MOV WORD_OP[BX+4*3][DI],SP
- (3) MOV WORD_OP1,WORD_OP2
- (4) MOV AX,WORD_OP1[DX]
- (5) MOV SAVE_WORD,DS
- (6) MOV SP,SS:DATA_WORD[BX][SI]
- (7) MOV [BX][SI],2





4、假设程序中的数据定义如下：

LNAME DB 30 DUP(?)

ADDRESS DB 30 DUP(?)

CITY DB 15 DUP(?)

CODE_LIST DB 1,7,8,3,2

- (1) 用一条MOV指令将LNAME的偏移地址放入AX。**
- (2) 用一条指令将CODE_LIST的头两个字节的内容放到SI。**
- (3) 写一条伪操作使CODE_LENGTH的值等于CODE_LIST域的实际长度**



5、试说明下述指令中哪些需要加上PTR伪操作:

BVAL DB 10H,20H

WVAL DW 1000H

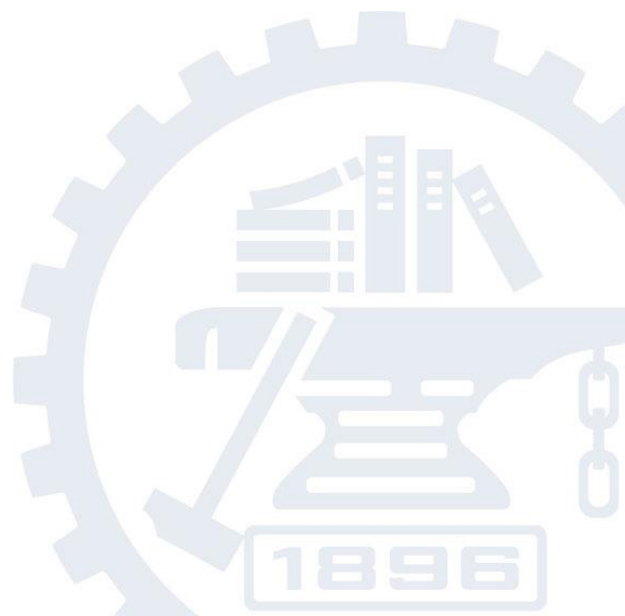
(1) MOV AL,BVAL

(2) MOV DL,[BX]

(3) SUB [BX],2

(4) MOV CL,WVAL

(5) ADD AL,BVAL+1





6、若TABLE为数据段0032H单元的符号名，其中存放的内容为1234H，试问下列两条指令有什么区别？执行完指令后，AX寄存器中的内容是什么？

MOV AX, TABLE

LEA AX, TABLE





7、已知程序段如下：

MOV AX, 1234H

MOV CL, 4

ROL AX, CL

DEC AX

MOV CX, 4

MUL CX

试问：

- (1) 每条指令执行后，AX的内容是什么？**
- (2) 每条指令执行后，CF、SF和ZF的值是什么？**
- (3) 程序执行完后，AX和CX的内容是什么？**