

# 《微型计算机原理与接口技术》

## 第5版

### 第3章

## 8086的寻址方式和指令系统



## § 3.3 8086的指令系统

3.3.1 数据传送指令

3.3.2 算术运算指令

3.3.3 逻辑运算和移位指令

3.3.4 字符串处理指令

3.3.5 控制转移指令

3.3.6 处理器控制指令



### 3.3.3 逻辑运算和移位指令

◆ 逻辑运算和移位指令，对字节或字操作数进行按位操作，见表3.7。

表 3.7 逻辑运算和移位指令

逻辑运算	
NOT	取反
AND	逻辑乘(与)
OR	逻辑加(或)
XOR	异或
TEST	测试
算术逻辑移位	
SHL/SAL	逻辑/算术左移
SHR	逻辑右移
SAR	算术右移
循环移位	
ROL	循环左移
ROR	循环右移
RCL	通过进位的循环左移
RCR	通过进位的循环右移



# 1. 逻辑运算指令 (Logical Operations)

## 1) NOT 取反指令 (Logical Not)

指令格式: NOT 目的

指令功能: 目的 $\leftarrow$ 目的取反

目的操作数可以是8位或16位寄存器或存储器, 对存储器操作数要说明类型。

例3.65 NOT指令只有一个操作数, 介绍几种用法。

NOT AX ; AX $\leftarrow$ AX取反

NOT BL ; BL $\leftarrow$ BL取反

NOT BYTE PTR [BX]  
; 对存储器字节单元内容取反后送回该单元



- ◆ 以下为双操作数指令。源操作数可以是8或16位立即数、寄存器、存储器，目的操作数只能是寄存器或存储器，两个操作数不能同时为存储器。
- ◆ 指令执行后，均将CF和OF清0，ZF、SF和PF反映操作结果，AF未定义，源操作数不变。

## 2) AND 逻辑与指令 (Logical AND)

指令格式： AND 目的， 源

指令功能： 目的←目的∧源

- 主要用于使操作数的某些位保留(和“1”相与)，而使某些位清除(和“0”相与)。

例3.66 设AX中是数字5和8的ASCII码，即AX=3538H，将它们转换成BCD码，结果仍放回AX。指令如下：

AND AX, 0F0FH            ; AX←0508H。屏蔽高4位，  
                             ; 截得低4位

# 1. 逻辑运算指令

## 3) OR 逻辑或指令 (Logical OR)

指令格式：OR 目的，源

指令功能：目的 $\leftarrow$ 目的 $\vee$ 源

- 它主要用于使操作数的某些位保留(和“0”相或)，而使某些位置1(和“1”相或)。

例3.67 设AX中存有两个BCD数0508H，要将它们分别转换成ASCII码，结果仍在 AX中。可用如下指令实现：

OR AX, 3030H ; AX $\leftarrow$ 3538H



# 1. 逻辑运算指令

## 4) XOR 异或操作指令 (Exclusive OR)

指令格式： XOR 目的， 源

指令功能： 对两个操作数进行按位逻辑异或运算，  
结果送回目的操作数，即

目的 $\leftarrow$ 目的 $\vee$ 源

- 用于使操作数的某些位保留(和“0”相异或)，而使某些位取反(和“1”相异或)。

例3.68 若AL中存有某外设端口的状态信息，其中D1位控制扬声器发声，要求该位在0和1之间来回变化，原来是1变成0，原来是0变成1，其余各位保留不变。

可用以下指令实现：

```
XOR  AL, 00000010B
```



# 1. 逻辑运算指令

## 5) TEST 测试指令 (Test)

指令格式: TEST 目的, 源

指令功能: 目的 ∧ 源, 并修改标志位, 但不回送结果

- 它常用在要检测某些条件是否满足, 但又不希望改变原有操作数的情况下。





# 1. 逻辑运算指令

例3.69 设AL寄存器中存有报警标志。若D7=1，表示温度报警，程序要转到温度报警处理程序T\_ALARM；D6=1，则转压力报警程序P\_ALARM。为此，可用TEST指令来实现这种功能：

TEST     AL, 80H                    ; 查AL的D7=1?

JNZ       T\_ALARM                   ; 是1(非零)，则转  
                                     ; 温度报警程序

TEST     AL, 40H                   ; D7=0, D6=1?

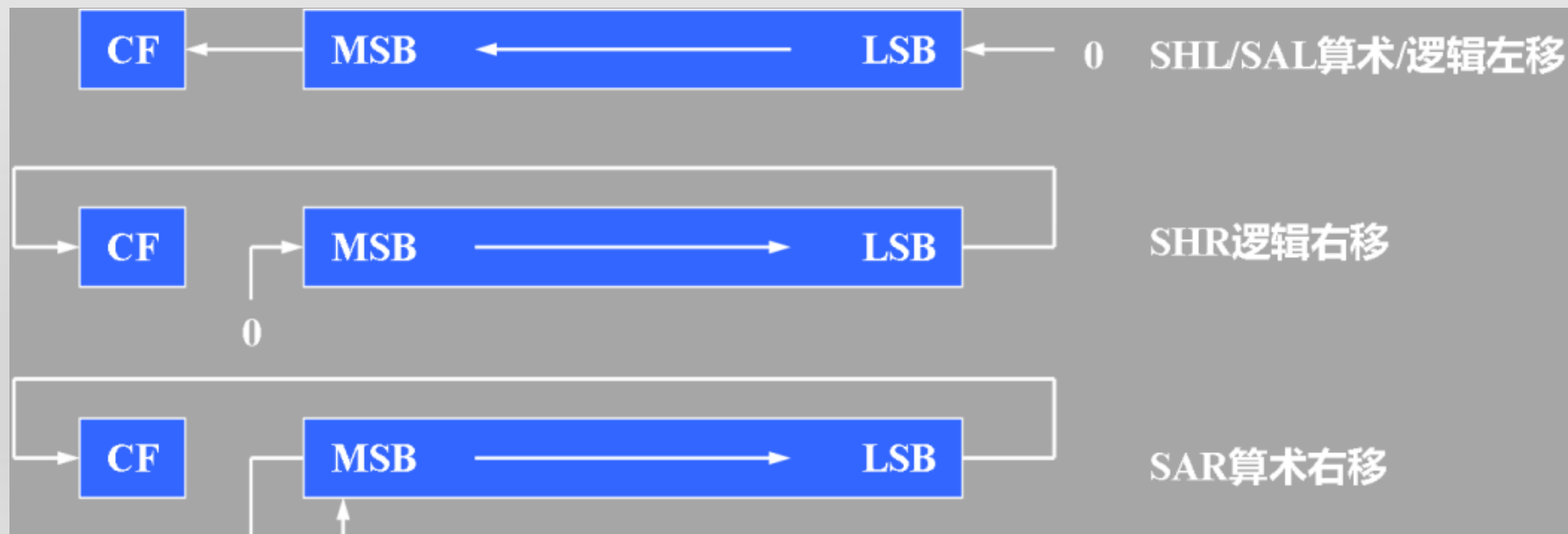
JNZ       P\_ALARM                   ; 是1，转压力报警

- 其中，JNZ为条件转移指令，表示结果非0(ZF=1)则转移。



## 2. 算术逻辑移位指令 (Shift Arithmetic and Shift Logical)

- 可对寄存器或存储器中的字或字节的各位进行算术移位或逻辑移位，移动的次数由指令中的计数值决定，如图3.17。



## 1) SAL 算术左移指令 (Shift Arithmetic Left)

指令格式: SAL 目的, 计数值

## 2) SHL 逻辑左移指令 (Shift Logic Left)

指令格式: SHL 目的, 计数值

指令功能: 以上两条指令的功能完全相同。

- 均将目的操作数的各位左移, 每移一次, 最低位LSB补0, 最高位MSB进标志位CF。移动一次, 相当于将目的操作数乘以2。
- 计数值表示移位次数, 可以是1。若大于1, 则用CL存放, 并要事先将次数存入CL。
- 移位次数最多为31(即00011111B)。



## 2. 算术逻辑移位指令

### 例3.70

MOV AH, 00000110B; AH=06H

SAL AH, 1 ; 将AH内容左移一位后,  
; AH=0CH

MOV CL, 03H ; CL←移位次数3

SHL DI, CL ; 将DI内容左移3次

SAL BYTE PTR [BX], 1  
; 将内存单元字节左移1位



## 2. 算术逻辑移位指令

### 3) SHR 逻辑右移指令 (Shift Logic Right)

指令格式: SHR 目的, 计数值

指令功能: 使目的操作数各位右移, 每移一次, 最低位进入CF, 最高位补0。

- 右移次数由计数值决定, 同SAR/SHL指令一样。
- 若目的操作数为无符号数, 每右移一次, 使目的操作数除以2。

例3.71 用右移的方法做除法 $133/8=16...5$ , 即:

MOV AL, 10000101B ; AL=133

MOV CL, 03H ; CL=移位次数

SHR AL, CL ; 右移3次, AL=10H, 余数5丢失



## 2. 算术逻辑移位指令

### 4) SAR 算术右移指令 (Shift Arithmetic Right)

指令格式：SAR目的，计数值

指令功能：每移位一次，最低位进入CF，但最高位(即符号位)保持不变，而不是补0。相当于对带符号数进行除2操作。

例3.72 用SAR指令计算 $-128/8=-16$ 的程序段如下：

MOV AL, 10000000B; AL= -128

MOV CL, 03H ; 右移次数为3

SAR AL, CL ; 算术右移3次后,  
; AL=0F0H= -16

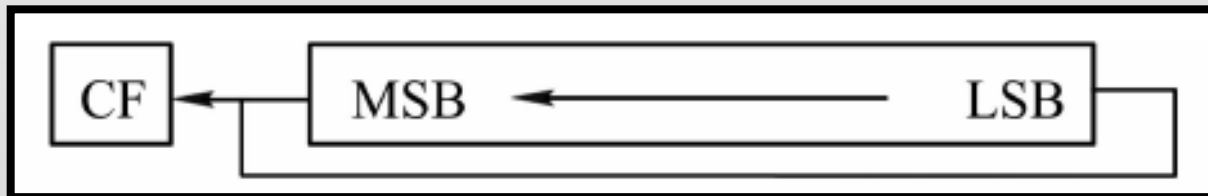


### 3. 循环移位指令 (Rotate)

- ❖ 算术逻辑移位指令，移出的操作数数位均丢失。  
循环移位指令则把数位从操作数的一端移到其另一端，从操作数中移走的位不会丢失。

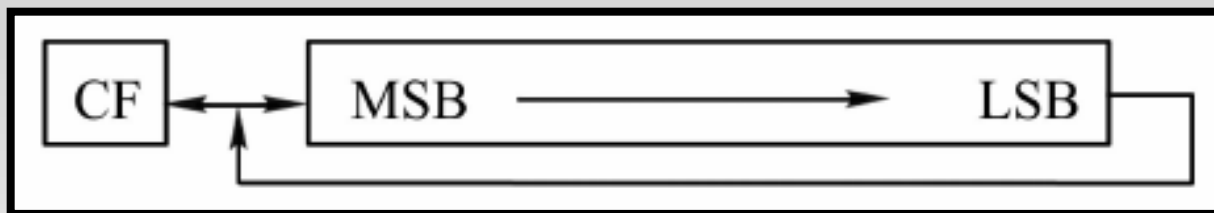
#### 1) ROL 循环左移指令 (Rotate Left)

指令格式： ROL 目的，计数值



#### 2) ROR 循环右移指令 (Rotate Right)

指令格式： ROR 目的，计数值

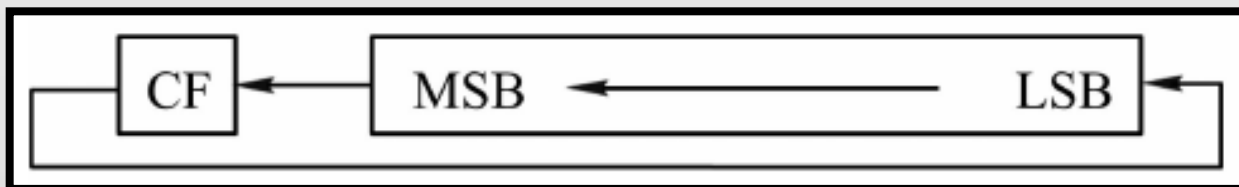


### 3. 循环移位指令 (Rotate)

#### 3) RCL 通过进位位循环左移

(Rotate through Carry Left)

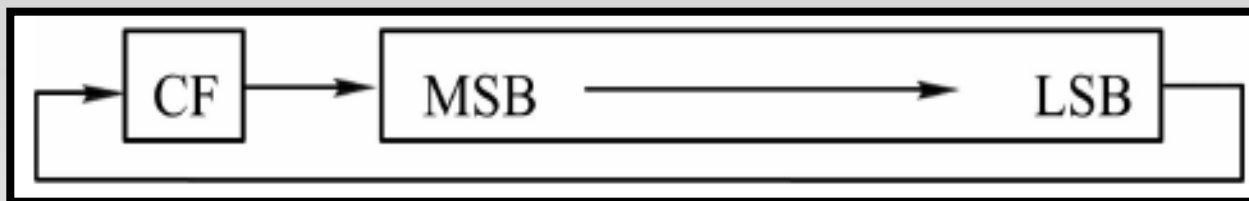
指令格式: RCL 目的, 计数值



#### 4) RCR 通过进位位循环右移

(Rotate through Carry Right)

指令格式: RCR 目的, 计数值





### 3. 循环移位指令

- 目的操作数可以是8/16位的寄存器操作数或内存操作数，计数值含义同上，即1或由CL指定。
- ROL和ROR为小循环移位指令，没有把CF包含在循环中；RCL和RCR为大循环指令，把 CF作为整个循环的一部分参加循环移位。
- CF的值由最后一次被移出的值决定。

#### 例3.73

ROL BX, CL

；将BX中的数，不带进位位左移规定次数

ROR WORD PTR [SI] , 1

；将内存单元的字，不带进位右移1次



### 3. 循环移位指令

#### 例3.74

设  $CF=1$ ,  $AL=1011\ 0100B$

若执行指令  $ROL\ AL,\ 1$

则  $AL=0110\ 1001B$ ,  $CF=1$ ,  $OF=1$ ;

若执行指令  $ROR\ AL,\ 1$

则  $AL=0101\ 1010B$ ,  $CF=0$ ,  $OF=1$ ;

若执行指令  $RCR\ AL,\ 1$

则  $AL=1101\ 1010B$ ,  $CF=0$ ,  $OF=0$ ;

若执行指令  $MOV\ CL,\ 3$  和  $RCL\ AL,\ CL$

则  $AL=1010\ 0110B$ ,  $CF=1$ ,  $OF$  不确定。



## § 3.3 8086的指令系统

3.3.1 数据传送指令

3.3.2 算术运算指令

3.3.3 逻辑运算和移位指令

3.3.4 字符串处理指令

3.3.5 控制转移指令

3.3.6 处理器控制指令



## 3.3.4 字符串处理指令

- 字符串是指一系列存放在存储器中的字或字节数据。
- 使用字符串操作指令时，可用指令中的源串和目的串名(即操作数)来表明是字节还是字，也可在指令助记符后加B说明是字节，加W说明是字操作，每种指令就都有3种格式。
- 5条1字节字符串操作指令见表3.8。

表 3.8 字符串操作指令的类型和格式

指令名称	字节/字操作	字节操作	字操作
字符串传送	MOVS 目的串,源串	MOVSB	MOVSW
字符串比较	CMPS 目的串,源串	CMPSB	CMPSW
字符串扫描	SCAS 目的串	SCASB	SCASW
字符串装入	LODS 源串	LODSB	LODSW
字符串存储	STOS 目的串	STOSB	STOSW

**字符串指令执行时，必须遵守以下的隐含约定：**

- (1) 源串位于数据段中，源串字符的始址(或末址)为 DS: SI。**
- (2) 目的串位于附加段中，目的串字符的始址(或末址)为 ES: DI。**
- (3) 每执行一次字符串指令，指针SI和DI会自动修改，指向下一待操作单元。**
- (4) DF标志控制字符串处理的方向：**
  - **DF=0递增。执行一次字节串操作，SI、DI各+1；字串操作，SI和DI各+2；**
  - **DF=1递减。执行一次字节串操作，SI、DI各-1；字串操作，SI和DI各-2。**
  - **STD指令使DF=1，CLD指令使DF=0。**
- (5) CX=要处理的字符串长度(字节或字数)。**



# 字符串处理指令

-

# 1. MOVS字符串传送指令 (Move String)

指令格式: MOVS 目的串, 源串

指令功能: 把源串中的一个字节或字, 传送目的串中, 且自动修改指针SI和DI。

- 利用MOVS指令, 能很方便地将数据从内存的某一地址(源地址)传送到另一个地址(目的地址), 还能自动修改源和目的地址。若使用重复前缀, 可用一条指令传送一批数据。

例3.75 要求把数据段中以SRC-MESS为偏移地址的一串字符“HELLO!”, 传送到附加段中以NEW-LOC开始的单元中。实现该操作的程序如下:



DATA	SEGMENT	; 数据段
SRC_MESS	DB 'HELLO!'	; 源串
DATA	ENDS	
;		
EXTRA	SEGMENT	; 附加段
NEW_LOC	DB 6 DUP(?)	; 存放目的串
EXTRA	ENDS	
;		
CODE	SEGMENT	; 代码段
	ASSUME CS:CODE, DS:DATA, ES:EXTRA	
START:	MOV AX, DATA	
	MOV DS, AX	; DS=数据段段址
	MOV AX, EXTRA	
	MOV ES, AX	; ES=附加段段址
	LEA SI, SRC_MESS	; SI 指向源串偏移地址
	LEA DI, NEW_LOC	; DI 指向目的串偏移地址
	MOV CX, 6	; CX 作串长度计数器
	CLD	; 清方向标志, 地址增量
	REP MOVSB	; 重复传送串中的各字节, 直到 CX=0 为止
CODE	ENDS	
	END START	

➤ 比较发现，使用有重复前缀REP的MOVSB指令，程序更简洁。



## 2. CMPS 字符串比较指令 (Compare String)

指令格式： CMPS            目的串， 源串

指令功能：将源串中数据减去目的串数据，但不改变两数据串的原始值，结果反映在标志位上。操作后源串和目的串指针会自动修改。

- 常用此指令来比较两个串是否相同，并由其后的条件转移指令，根据CMPS执行后的标志位值，决定程序的转向。

CMPS指令前可加重复前缀，下面每两条指令功能相同：

REPE CMPS            ； 若CX≠0(未比完)和

REPZ CMPS            ； ZF=1(两串相等)，则重复

REPNE CMPS            ； 若CX≠0(串没有结束)和串不相等


REPNZ CMPS            ； (ZF=0)，则重复比较。



## 2. CMPS 字符串比较指令

例3.76 比较两个字符串，一个是在程序中设定的口令串PASSWORD，另一个是从键盘输入的字符串IN-WORD，若输入串与口令串相同，程序开始执行。否则程序驱动扬声器发声，警告用户口令不符，拒绝往下执行。这可以用CMPS指令来实现，有关程序段如下：

```
DATA          SEGMENT                                ; 数据段
PASSWORD DB '8086 CPU'                               ; 口令串
IN_WORD  DB '8088 CPU'                               ; 从键盘输入的
串
COUNT   EQU      8                                  ; 串长度
DATA      ENDS
```



## 2. CMPS 字符串比较指令

```
CODE    SEGMENT                ; 代码段
        ASSUME DS: DATA, ES: DATA
        ...
        LEA SI, PASSWORD      ; 源串指针
        LEA DI, IN_WORD       ; 目的串指针
        MOV CX, COUNT          ; 串长度
        CLD                    ; 地址增量
        REPZ CMPSB ; CX≠0且串相等时重复比较
        JNE SOUND ; 若不相等, 转发声程序
OK:      ...                    ; 比完且相等, 往下执行
        ...
SOUND:   ...                    ; 使PC机扬声器发声
        ...                    ; 并退出
CODE    ENDS
```



### 3. SCAS 字符串扫描指令 (Scan String)

指令格式: SCAS 目的串

指令功能: 从AL(字节操作)或AX(字操作)寄存器的内容, 减去ES:DI为指针的目的串元素, 结果反映在标志位上, 但不改变源操作数。串操作后目的串指针DI会自动修改。

- 利用SCAS指令, 可在内存中搜索所需要的数据 (关键字)。指令执行前, 必须事先将它存在AL(字节)或AX(字)中。
- SCAS指令前也可加重复前缀。



### 3. SCAS 字符串扫描指令

例3.77 在某字符串中搜寻字符A。若有，搜索次数送到BX；若无，将BX清0。设字符串始址STRING的偏址为0，字符串长度为CX。程序段如下：

```
MOV  DI, OFFSET STRING ; DI=字符串偏移地址
MOV  CX, COUNT        ; CX=字符串长度
MOV  AL, 'A'           ; AL=关键字A的ASCII码
CLD                    ; 清方向标志
REPNE SCASB            ; CX≠0(没查完)和
                        ; ZF=0(不相等)时重复
JZ   FIND              ; 若ZF=1, 已搜到, 转出
MOV  DI, 0              ; 若ZF=0, 没搜到, DI←0
FIND:
MOV  BX, DI             ; BX←搜索次数
```



## 4. LODS 数据串装入指令 (Load String)

指令格式: LODS 源串

指令功能: 把数据段中以SI作为指针的串元素, 传送到AL(字节操作)或AX(字操作)中, 同时修改SI。

- 为该指令加重重复前缀没有意义。因为每重复传送一次数据, 累加器中的内容就被改写, 执行重复传送操作后, 只能保留最后写入的那个数据。



## 5. STOS 数据串存储指令 (Store String)

指令格式: STOS 目的串

指令功能: 将累加器AL或AX中的一个字节或字, 传送到以ES:DI为目标指针的目的串中, 同时修改DI, 以指向串中的下一个单元。

- STOS指令与REP重复前缀连用, 即执行指令REP STOS, 能方便地用累加器中的一个常数, 对一个数据串进行初始化。例如, 初始化为全0的串。





## 5. STOS 数据串存储指令

例3.78 数据段中有个数据块，存有8位带符号数，始址BLOCK，要求将正、负数分开，正数送到附加段中始址为PLUS\_DATA的缓冲区，负数送到附加段中始址为MINUS\_DATA的缓冲区。

数据块可看成一个数据串，用SI作源串指针，DI和BX作正、负数目的缓冲区指针，CX控制循环次数。程序段如下：

START:

MOV SI, OFFSET BLOCK ; SI为源串指针

MOV DI, OFFSET PLUS\_DATA

; DI为正数目的区指针

MOV BX, OFFSET MINUS\_DATA

; BX为负数目的区指针





MOV CX, COUNT	; CX放循环次数
CLD	
GOON:	
LODS BLOCK	; AL←取源串的一个字节
TEST AL, 80H	; 是负数?
JNZ MINUS	; 是, 转MINUS
STOSB	; 非负数, 将字节送正数区
JMP AGAIN	; 处理下一个字节
MINUS:	
XCHG BX, DI	; 交换正负数指针
STOSB	; 负数送入负数区
XCHG BX, DI	; 恢复正负数指针
AGAIN:	
DEC CX	; 次数减1
JNZ GOON	; 未处理完, 继续传送
HLT	; 已完, 停机



## 5. STOS 数据串存储指令

- 程序中，正负数的存储均使用STOSB指令，该指令必须以SI为源指针，DI为目的指针。
- 但存储负数时，负数区的目的指针在BX中，因此要用XCHG指令将BX内容送进DI，让DI指向负数区，同时也把DI中的正数区目的指针保护起来。
- 执行STOSB指令后，再用XCHG指令将BX和DI交换回来，以便下次转回GOON标号后，LODS指令仍能正确执行。



## § 3.3 8086的指令系统

3.3.1 数据传送指令

3.3.2 算术运算指令

3.3.3 逻辑运算和移位指令

3.3.4 字符串处理指令

3.3.5 控制转移指令

3.3.6 处理器控制指令



### 3.3.5 控制转移指令

- ◆ 通常，程序中的指令都是顺序地逐条执行的，执行顺序由CS和IP决定，每取出一条指令，指令指针IP自动进行调整，指向下一个存储单元。
- ◆ 利用控制转移指令可以改变CS和IP的值，从而改变指令的执行顺序。8086提供了5类转移指令，如表3.9。



# 控制转移指令

表 3.9 控制转移指令

无条件转移和过程调用指令	
JMP CALL RET	无条件转移 过程调用 过程返回
条 件 转 移	
JZ/JE 等 10 条指令 JA/JNBE 等 8 条指令	直接标志转移 间接标志转移
条 件 循 环 控 制	
LOOP LOOPE/LOOPZ LOOPNE/LOOPNZ JCXZ	CX $\neq$ 0 则循环 CX $\neq$ 0 和 ZF=1 则循环 CX $\neq$ 0 和 ZF=0 则循环 CX=0 则转移
中 断	
INT INTO IRET	中断 溢出中断 中断返回



# 1. 无条件转移和过程调用指令

(Unconditional Transfer and Call)

## 1) JMP 无条件转移指令 (Jump)

指令格式: JMP 目的

指令功能: 无条件地转移到目的地址去执行。

这类指令又分成两种类型:

- 段内转移或近(NEAR)转移。转移指令的目的地址和JMP指令在同一代码段中, 转移时**仅改变IP的内容**, 段地址**CS的值不变**。
- 段间转移, 又称远(FAR)转移。转移指令的目的地址和JMP指令不在同一段中, 转移时, **CS和IP的值都要改变**, 程序要转移到另一个代码段去执行。



# 无条件转移指令

- 就转移地址提供的方式而言，又可分为两种方式：
  - **直接转移**。在指令码中直接给出转移的目的地址，目的操作数用一个标号来表示。它又可分为**段内直接转移**和**段间直接转移**。
  - **间接转移**。目的地址包含在某个16位寄存器或存储单元中，CPU必须根据寄存器或存储器寻址方式，间接地求出转移地址。同样，这种转移类型又可分为**段内间接转移**和**段间间接转移**。
- ◇ 所以无条件转移指令可分成段内直接转移、段内间接转移、段间直接转移和段间间接转移四种不同类型和方式，如表3.10所示。

# 无条件转移指令

表 3.10 无条件转移指令的类型和方式

类 型	方式	寻址目标	指令举例
段内 转移	直接	立即短转移(8 位)	JMP SHORT PROG-S
	直接	立即近转移(16 位)	JMP NEAR PTR PROG-N
	间接	寄存器(16 位)	JMP BX
	间接	存储器(16 位)	JMP WORD PTR 5[BX]
段间 转移	直接	立即转移(32 位)	JMP FAR PTR PROG-F
	间接	存储器(32 位)	JMP DWORD PTR [DI]





# 无条件转移指令

## (1) 段内直接转移指令

指令格式:

JMP SHORT 标号

JMP NEAR PTR 标号 (或 JMP 标号)

- 段内相对转移指令，目的操作数均用标号表示。
- 转向的有效地址=IP+8位/16位位移量(DISP)。
- 若转移范围在-128~+127字节内，称为短转移，指令中只需要用8位位移量，在标号前加说明符 SHORT。
- 若位移量是16位，称为近转移，目的地址与当前IP的距离在-32768~+32767字节之间。可加说明符 NEAR PTR，也可省略。这类指令用得最多。

# 无条件转移指令

例3.79 给出一个含有一条无条件转移指令的简单程序的列表文件，它是由汇编语言源程序经汇编程序翻译后产生的。即

； 行号	偏移量	机器码	程 序
1	0000	CODE	SEGMENT
2			ASSUME CS:CODE
3	0000	0405	PROG-S: ADD AL, 05H
4	0002	90	NOP
5	0003	EBFB	JMP SHORT PROG-S
6	0005	90	NOP
7	0006	CODE	ENDS
8			END

对程序的解释请参看教材。

# 无条件转移指令

## (2) 段内间接转移指令

转向的16位地址存放在一个16位寄存器或字存储器单元中。

- 用寄存器间接寻址的段内转移指令，转向的地址存放在寄存器中，执行操作： $IP \leftarrow \text{寄存器内容}$ 。

例3.80

**JMP BX**

若指令执行前， $BX=4500H$ ；

指令执行时， $IP \leftarrow 4500H$ ，程序转到代码段内偏移地址为4500H处执行。

# 无条件转移指令

- 用存储器间接寻址的段内转移指令，先计算出存储单元的物理地址，再从中取一个字送到IP。即 $IP \leftarrow \text{字存储单元内容}$ 。

例3.81

`JMP WORD PTR 5 [BX]`

； WORD PTR说明是字操作

设指令执行前

$DS=2000H$ ,  $BX=100H$ ,  $(20105H)=4F0H$ ;

则指令执行后

$IP=(20000H+100H+5H) = (20105H)=4F0H$ ,  
转到段内 $IP=4F0H$ 处执行。



# 无条件转移指令

## (3) 段间直接(远)转移指令

指令中用远标号直接给出转向的CS:IP, 程序从一个代码段转到另一个代码段。

例3.82

JMP FAR PTR PROG\_F

; FAR PTR说明PROG\_F为远标号

指令执行的操作:

$IP \leftarrow \text{PROG\_F的段内偏移量}$

$CS \leftarrow \text{PROG\_F所在段的段地址}$

设标号PROG\_F的逻辑地址=3500H: 080AH, 则:

指令执行后,  $IP=080AH$ ,  $CS=3500H$ , 程序转到3500:080AH处执行。

# 无条件转移指令

## (4) 段间间接转移指令

操作数为存储器，要转移的目的地址CS:IP存放在存储器中。需加说明符DWORD PTR，表示转向地址需取双字。

例3.83 JMP DWORD PTR [SI+0125H]

设指令执行前，CS=1200H，IP=05H，DS=2500H，SI=1300H；

内存单元(26425H)=4500H，(26427H)=32F0H，指令中的位移量DISP=0125H。

指令的执行过程如图3.20。



# 无条件转移指令

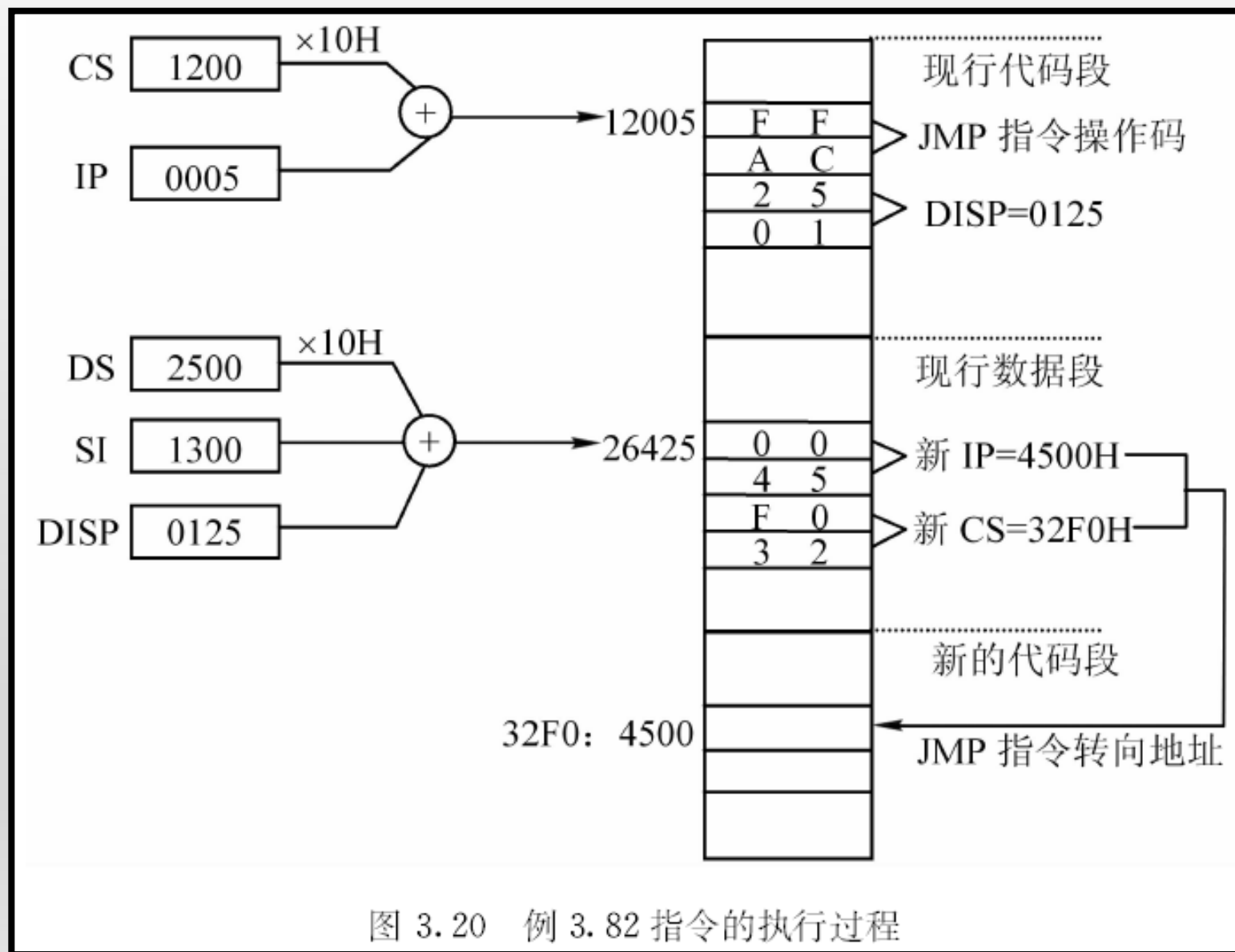


图 3.20 例 3.82 指令的执行过程

# 过程调用指令

## 2) 过程调用和返回指令 (Call and Return)

- 把某些能完成特定功能又常用的程序段，编写成独立模块，称为过程(Procedure)或子程序(Subroutine)。
- 在主程序中用CALL语句调用这些过程，格式为：  
CALL 过程名
- 过程以PROC开头，ENDP结束。过程中要安排一条返回指令RET，过程执行完后能正确返回主程序。
- 若在过程运行中又调用另一过程，称为过程嵌套。
- 主程序和过程在同一代码段，称为近调用，不在同一段则称为远调用。
- 过程调用的寻址方式与转移指令类似，但没有段内短调用。由于调用结束后需返回原程序继续运行，要执行保护和恢复返址操作，比转移复杂。





# 过程调用指令

- CALL指令分两步执行：

**第一步：返址入栈，将CALL下面指令的地址推入堆栈**

近调用执行的操作： $SP \leftarrow SP - 2$ ，IP入栈

远调用执行的操作： $SP \leftarrow SP - 2$ ，CS入栈

$SP \leftarrow SP - 2$ ，IP入栈

**第二步：转到子程序入口执行相应的子程序。入口地址由CALL指令的目的操作数提供，寻址方法与JMP指令类似。**

- 执行过程中的RET指令时，从栈中弹出返址，使程序返回主程序继续执行。也有两种情况：
  - 从近过程返回，从栈中弹出1个字 $\rightarrow$ IP，并且使 $SP \leftarrow SP + 2$ 。
  - 从远过程返回，先从栈中弹出1个字 $\rightarrow$ IP，并且使 $SP \leftarrow SP + 2$ ；再弹出1个字 $\rightarrow$ CS，并使 $SP \leftarrow SP + 2$ 。



# 过程调用指令

下面举例说明CALL和RET指令的4种寻址方式。

## (1) 段内直接调用和返回

例3.84

CALL PROG-N ; PROG-N是一个近标号  
该指令含3个字节，编码格式为：



设调用前：CS: IP=2000H: 1050H, SS: SP=5000H:  
0100H, PROG-N与CALL指令之间的字节距离等于1234H  
(即DISP=1234H)。



## 例3.84

则执行CALL指令的过程：

➤  $SP \leftarrow SP - 2$

即新的 $SP = 0100H - 2 = 00FEH$

➤ 返回地址的IP入栈

由于存放CALL指令的内存首地址为CS:  
 $IP = 2000: 1050H$ ，该指令占3字节，所以返回地址为 $2000: 1053H$ ，即 $IP = 1053H$ 。于是 $1053H$ 被推入堆栈。

➤ 根据当前IP值和位移量DISP计算出新的IP值，作为子程序的入口地址，即

$$IP = IP + DISP = 1053H + 1234H = 2287H$$

➤ 程序转到本代码段中偏移地址为 $2287H$ 处执行

指令CALLPROG\_N的执行过程如图3.21(a) ↓

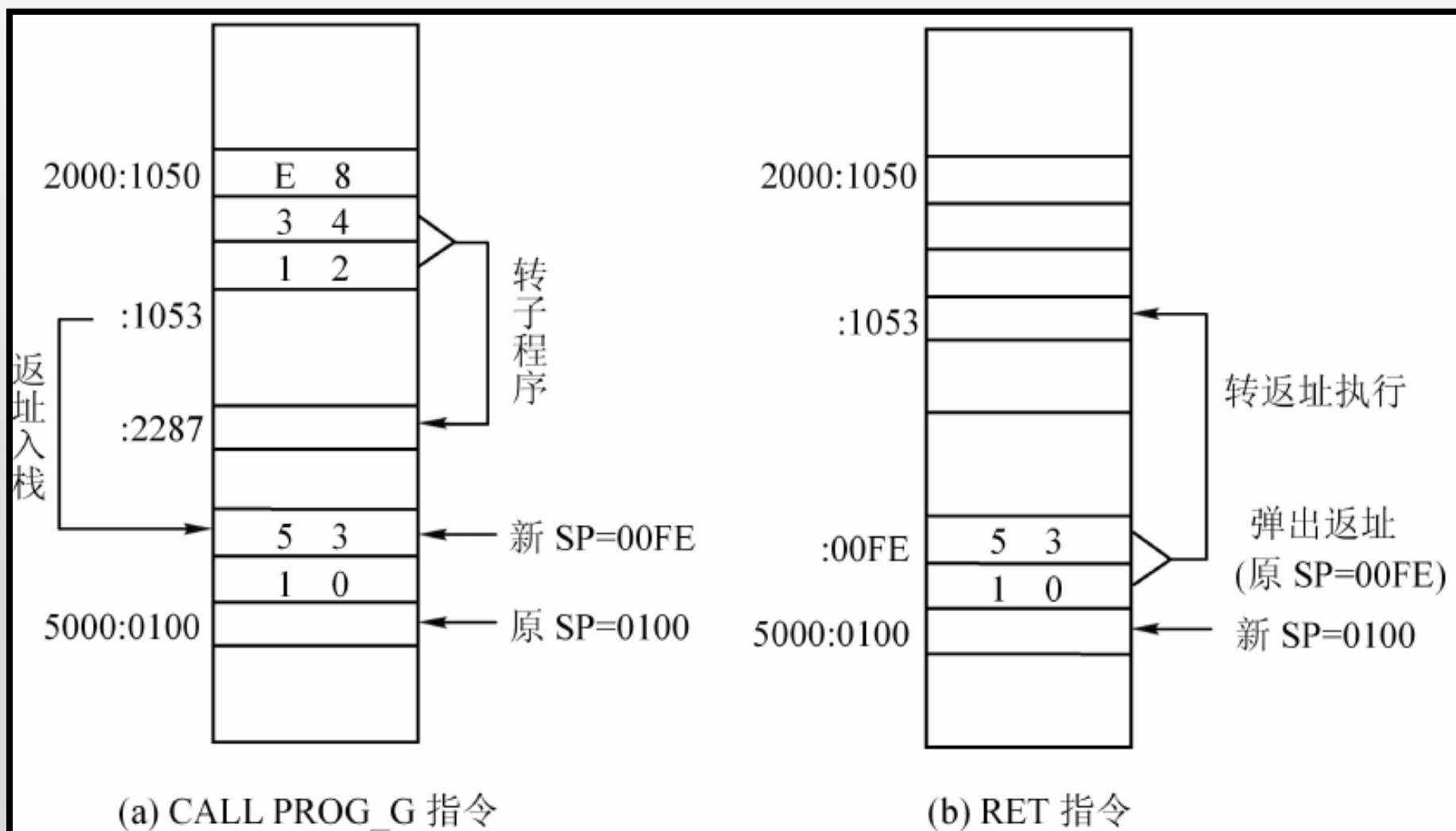


图 3.21 段内直接调用和返回指令执行情况

# 过程调用指令

RET指令的寻址方式与CALL一样，在本例中也是段内直接调用。执行过程如下：

■  $IP \leftarrow (SP \text{ 和 } SP+1) \text{ 单元内容}$

即返址 $IP=1053H$ 从栈中弹出

■  $SP \leftarrow SP+2$

$SP=00FEH+2=0100H$ ，即恢复原SP

结果，返回CALL下面的那条指令，即从2000:1053处继续执行程序，如图3.21(b)。



# 过程调用指令

## (2) 段内间接调用和返回

例3.85 下面是两条段内间接调用指令的例子，返址在寄存器或内存中。

CALL BX

CALL WORD PTR [BX+SI]

它们执行的操作分三步，前两步与直接调用相同，第三步不同，具体为：

➡  $SP \leftarrow SP - 2$

➡ IP入栈

➡  $IP \leftarrow EA$ ，计算出目的地址的有效地址EA，送入IP，以此转移。

# 过程调用指令

设：DS=1000H, BX=200H, SI=300H, (10500H)=3210H

CALL BX

转移地址在BX中，此调用指令执行后， $IP \leftarrow 0200H$ ，转到段内偏移地址为0200H处执行。

CALL WORD PTR [BX+SI]

子程序入口地址在内存字单元中，其值为  $(16 \times DS + BX + SI)$   
 $= (10000H + 0200H + 0300H)$

$= (10500H) = 3210H$ ，即EA=3210H

此指令执行后， $IP \leftarrow 3210H$ ，转到段内偏移地址为3210H处执行。

对应的RET指令执行的操作与段内直接过程的返回指令类似。



# 过程调用指令

## (3) 段间直接调用

例3.86

**CALL FAR PTR PROG\_F** ; PROG\_F是一个远标号  
该指令含5个字节，编码格式为：

9A	DISP_L	DISP_H	SEG_L	SEG_H
----	--------	--------	-------	-------

设调用前：CS: IP=1000: 205AH, SS: SP=2500: 0050H,  
标号PROG-F所在单元的地址指针CS: IP=3000: 0500H。  
存放CALL指令的内存首址为1000: 205AH，由于该指令长度  
为5个字节，所以返回地址应为1000: 205FH。



# 过程调用指令

执行远调用CALL指令的过程如图3.22所示，具体为：

- ➡  $SP \leftarrow SP - 2$                       即  $SP = 0050H - 2 = 004EH$
- ➡ CS入栈                                即  $CS = 1000H$  入栈
- ➡  $SP \leftarrow SP - 2$                       即  $SP \leftarrow 004CH$
- ➡ IP入栈                                即  $IP = 205FH$  入栈
- ➡ 子程序入口                          将  $PROG-F$  的段地址和  
偏移地址送  $CS: IP$   
即  $CS \leftarrow 3000H, IP \leftarrow 0500H$
- ➡ 行子程序  $PROG-F$

# 过程调用指令

过程PROG-F中的RET指令的寻址方式也是段间直接调用，返回时执行的操作为：

➡  $SP \leftarrow SP + 2$                       即  $SP \leftarrow 004C + 2 = 004EH$

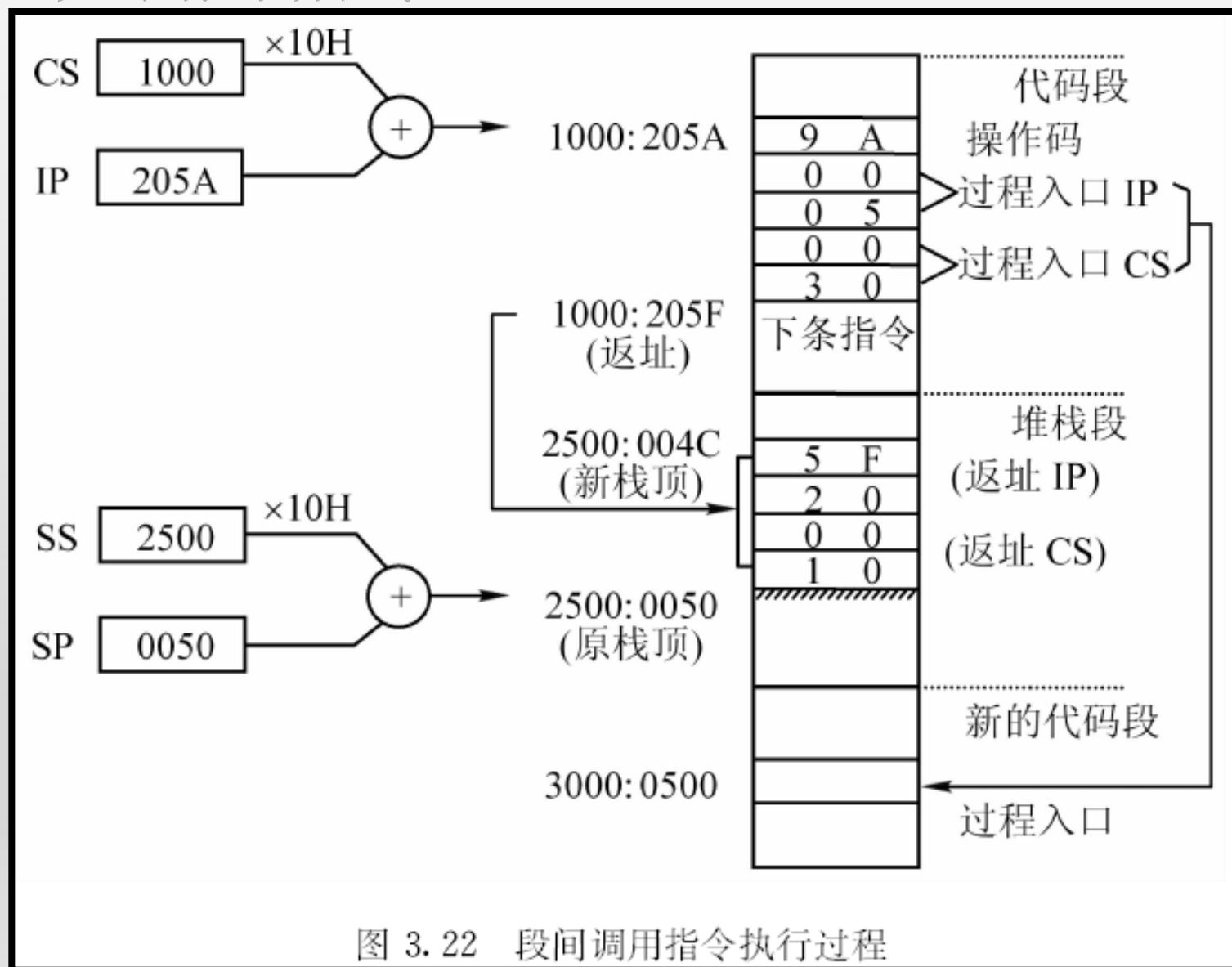
➡  $IP \leftarrow \text{栈中内容}$                        $IP \leftarrow 205FH$

➡  $SP \leftarrow SP + 2$                        $SP \leftarrow 004EH + 2 = 0050H$

➡  $CS \leftarrow \text{栈中内容}$                        $CS \leftarrow 1000H$

所以程序转返回地址CS: IP=1000: 205FH处执行。

# 过程调用指令



# 过程调用指令

## (4) 段间间接调用

- 操作数必须是存储单元，从该单元开始存放的双字表示过程的入口地址，指令中用DWORD PTR说明是对存储单元进行双字操作。

例3.87 CALL DWORD PTR [BX]

设调用前，DS=1000H，BX=200H，(10200H)=31F4H，  
(10202)=5200H。

执行时先将返址的CS: IP推入堆栈，再转向过程入口。

指令中操作数地址=DS×16+BX=10000H+200H=10200H从中取得的双字就是过程入口地址，即：

IP←(10200H)                      即IP=31F4H

CS←(10202H)                      即CS=5200H

# 过程调用指令

- 8086还有另一种带参数的返回指令，形式为：

RET n

n称为弹出值，它让CPU在弹出返回地址后，再从堆栈中弹出n个字节的数据，也就是让SP再加上n。n可以是0000~FFFFH范围内的任何一个偶数。

例如，指令RET 8，表示从堆栈中弹出地址后，再使SP的值加上8。

## 2. 条件转移指令 (Conditional Transfer)

- 将上条指令执行后的状态标志，作为测试条件，来决定是否转移。当条件成立，程序转向指令中给出的目的地址去执行；否则，仍顺序执行。
- 条件转移均为**段内短转移**，转移指令与目的地址必须在同一代码段中。转移距离范围为-128~+127字节。8位偏移量需用符号扩展法扩展到16位后才能与IP相加。
- 在指令中，目的地址均用标号表示，指令格式：  
条件操作符 标号
- 条件转移指令共18条，归类成**直接标志转移**和**间接标志转移**两大类。

## 1) 直接标志转移指令

- 助记符中直接给出标志状态测试条件，以CF、ZF、SF、OF和PF等5个标志的10种状态为判断条件，形成10条指令，如表3.11。有的指令有两种助记符，代表同样的指令，如JZ/JE。

表 3.11 直接标志条件转移指令

指令助记符	测试条件	指令功能	
JC	CF=1	有进位	转移
JNC	CF=0	无进位	转移
JZ/JE	ZF=1	结果为零/相等	转移
JNZ/JNE	ZF=0	不为零/相等	转移
JS	SF=1	符号为负	转移
JNS	SF=0	符号为正	转移
JO	OF=1	溢出	转移
JNO	OF=0	无溢出	转移
JP/JPE	PF=1	奇偶位为 1/为偶	转移
JNP/JPO	PF=0	奇偶位为 0/为奇	转移



## 2. 条件转移指令

例3.88 求AL和BL中的两数之和，若有进位，则AH置1，否则AH清0。

程序如下：

```
ADD    AL, BL           ; 两数相加
JC     NEXT             ; 若有进位，转NEXT
MOV     AH, 0            ; 无进位，AH清0
JMP     EXIT             ; 往下执行
```

NEXT:

```
MOV     AH, 1            ; 有进位，AH置1
EXIT:    ...              ; 程序继续进行
```





## 2. 条件转移指令

### 2) 间接标志转移

- 不在指令助记符中直接给出标志状态位的测试条件，但仍以某一个或几个标志的状态组合为测试条件，若条件成立则转移，否则顺序往下执行。
- 间接标志转移指令共有8条，列于表3.12中。每条指令都有两种不同的助记符。
- 在无符号数比较测试指令中，指令助记符中的“A”是英文Above的缩写，表示“高于”之意，“B”是英文Below的缩写，表示“低于”之意。

## 2. 条件转移指令

### ● 间接标志条件转移指令

表 3.12 间接标志条件转移指令

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	$CF \vee ZF=0$	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	$CF \vee ZF=1$	低于等于/不高于 转移
带符号数 比较测试	JG/JNLE	$(SF \nabla OF) \vee ZF=0$	大于/不小于等于 转移
	JGE/JNL	$SF \nabla OF=0$	大于等于/不小于 转移
	JL/JNGE	$SF \nabla OF=1$	小于/不大于等于 转移
	JLE/JNG	$(SF \nabla OF) \vee ZF=1$	小于等于/不大于 转移

## 2. 条件转移指令

例3.89 设 AL=F0H, BL=35H, 执行指令

CMP AL, BL ; AL-BL

JAE NEXT ; AL大于等于BL, 则转到NEXT

- JAE/JNB根据CF标志是否为0决定转移。若CF=0, 即无进位, 则转移, 这与直接标志转移指令中的JNC功能完全一样。
- 同样, JB/JNAE与JC指令的功能相同。
- 带符号数进行比较时, 不能仅根据SF或OF标志来判定, 而要将它们组合起来考虑。指令助记符中, “G”(Great than)表示“大于”, “L”(Less than)表示“小于”。

## 2. 条件转移指令

例3.90 设某学生的英语成绩已存放在AL中，如低于60分打印F(FAIL)；高于或等于85分，打印G(GOOD)；在60~84分之间，打印P (PASS)。程序为

CMP AL, 60	; 与60分比较
JB FAIL	; <60, 转FAIL
CMP AL, 85	; ≥60, 与85分比较
JAE GOOD	; ≥85, 转GOOD
MOV AL, 'P'	; 其它, 将AL←'P'
JMP PRINT	; 转打印程序
FAIL: MOV AL, 'F'	; AL←'F'
JMP PRINT	; 转打印程序
GOOD: MOV AL, 'G'	; AL←'G'
PRINT: ...	; 打印存在AL中的字符

## 2. 条件转移指令

### 例3.91

设某温度控制系统中，从温度传感器输入一个8位二进制摄氏温度值。当温度低于 $100^{\circ}\text{C}$ 时，打开加热器；温度升到 $100^{\circ}\text{C}$ 或以上时，关闭加热器。

温度传感器端口号为320H，控制加热器的输出信号连到端口321H的最低有效位，当它置1加热器打开，清0则关闭。

实现上述温度控制的程序：



## GET-TEMP:

```
MOV     DX, 320H    ; DX指向温度输入端口
IN      AL, DX      ; 读取温度值
CMP     AL, 100     ; 与100 °C比较
JB      HEAT_ON     ; <100 °C, 加热
JMP     HEAT_OFF    ; ≥100 °C, 停止加热
```

## HEAT-ON:

```
MOV     AL, 01H     ; D0位置1, 加热
MOV     DX, 321H    ; 加热器口地址
OUT     DX, AL      ; 打开加热器
JMP     GET_TEMP    ; 继续检测温度
```

## HEAT-OFF:

```
MOV     AL, 00      ; D0位置0, 停止加热
MOV     DX, 321H
OUT     DX, AL      ; 关闭加热器
...           ; 进行其它处理
```

## 2. 条件转移指令

例3.92 在首地址为TABLE的10个内存字节单元中, 存放了10个带符号数, 统计其中正数、负数和零的个数, 结果存入PLUS、NEGT和ZERO单元。

```
TABLE DB 01H, 80H, 0F5H, 32H, 86H
       DB 74H, 49H, 0AFH, 25H, 40H
PLUS   DB 0           ; 存正数个数
NEGT   DB 0           ; 存负数个数
ZERO   DB 0           ; 存0的个数
...
MOV    CX, 10         ; 数据总数
MOV    BX, 0          ; BX清0
```



## 2. 条件转移指令

AGAIN:

```
CMP TABLE [BX] , 0 ; 取一个数与0比
JGE GRET_EQ          ; ≥0, 转GRET_EQ
INC NEG_T             ; <0, 负数个数加1
JMP NEXT              ; 往下执行
```

GRET-EQ:

```
JG P-INC              ; >0, 转P-INC
INC ZERO              ; =0, 零个数加1
JMP NEXT              ; 往下执行
```

P-INC:

```
INC PLUS              ; 正数个数加1
```

NEXT:

```
INC BX                ; 数据地址指针加1
DEC CX                ; 数据计数器减1
JNZ AGAIN              ; 未完, 继续统计
```



### 3. 循环控制指令 (Iteration)

- 是一组增强型条件转移指令，用来控制程序段的重复执行，重复次数由CX中的内容决定，转移目标都是短标号，偏移量都是负值，即只能向前转移。均不影响任何标志。

#### 1) LOOP循环指令 (Loop)

指令格式：LOOP 短标号

指令功能：重复执行一系列指令。

- 重复次数放在CX中，执行一次指令， $CX-1$ 。如减1后 $CX \neq 0$ ，转到指令给定的标号处继续循环； $CX=0$ ，结束循环，转去执行LOOP指令后的那条指令。
- 一条LOOP指令相当于执行以下两条指令：  
    DEC CX  
    JNZ 标号

### 3. 循环控制指令

**例3.93** 商店里有8种商品，价格为83元，76元，65元，84元，71元，49元，62元和58元，要将每种商品提价7元，编程计算每种商品提价后的价格。

- 先将商品原价按BCD码形式，依次存放在以OLD开始的8个存储单元中，新价格存放进以NEW开始的8个单元，然后用LOOP指令来实现8次循环。即

```
OLD  DB  83H, 76H, 65H, 84H
```

```
      DB  71H, 49H, 62H, 58H
```

```
NEW      DB  8 DUP(?)
```

...

### 例3. 93

```
MOV      CX, 08H          ; 共8种商品
MOV      BX, 00H          ; BX作指针, 初值为0
NEXT:
MOV      AL, OLD [BX]      ; 读入一个商品的原价
ADD      AL, 7             ; 加上提价因子
DAA                      ; 调整为十进制数
MOV      NEW [BX], AL      ; 存放结果
INCBX                    ; 地址指针加1
LOOP     NEXT             ; 未加满8次, 继续循环
...                      ; 已加完8次
```



### 3. 循环控制指令

- 循环操作也可以只含一条指令，即LOOP指令自身，这样的程序段常用来实现延时。

例如：

```
MOV    CX, 10    ; 循环次数为10
```

DELAY:

```
LOOP   DELAY    ; 本指令重复执行10次
```



### 3. 循环控制指令

#### 例3.94

- 用循环和跳转指令，编写控制PC机扬声器发声的程序。
- 在PC机中，61H口的D1和D0位接到扬声器接口电路上。在D0=0的情况下，当D1=1时，扬声器被接通；当D1=0时，则断开。通过控制D1位的值，就能产生一个由1和0构成的二进制序列，使扬声器发声。
- 61H 口中的其它位则用来控制PC机的内部开关状态、奇偶校验及键盘状态等。要将这些状态保存起来。

## 控制扬声器发声程序：

```
IN      AL, 61H ; AL←从61H口读取数据  
AND     AL, 0FCH ; 保护D7~D2位, D0位清0
```

MORE:

```
XOR     AL, 02      ; 触发D1位使之在0和1间变化  
OUT     61H, AL    ; 控制扬声器开关通断  
MOV     CX, 260     ; CX=循环次数
```

```
DELAY:  LOOP  DELAY ; 循环延时
```

```
JMP     MORE       ; 再次触发
```

- 本例中，LOOP指令重复执行260遍，起延时作用，使开关通断维持一定时间。否则开关动作太快，发出的声音频率太高，人耳听不出来。

### 3. 循环控制指令

#### 2) LOOPE/LOOPZ 相等或结果为0时循环

(Loop If Equal/Zero)

指令格式: LOOPE 标号

或 LOOPZ 标号

指令功能: LOOPE是相等时循环, LOOPZ是结果为0时循环。

- 它们能完成相同功能, 具有不同助记符, 用来控制重复执行一组指令。
- 指令执行前, 先将重复次数送入CX, 每执行一次CX-1, 若-1后CX≠0和ZF=1, 则转到指令指定的标号处重复执行; 若CX=0或ZF=0, 便退出循环, 执行本指令后的那条指令。

**例3.95** 设1个50字节组成的数组存在ARRAY开始的内存中，测试数组中的元素，若为0，且不是最后1个，便继续进行下个元素的测试，直到找到第一个非零元素或查完了为止。

```
ARRAY      DB    xx, xx, ...  
              ; 含50个元素的数组  
MOV        BX, OFFSET ARRAY  
              ; BX指向数组开始单元  
DEC        BX      ; 指针-1  
MOV        CX, 50   ; CX=元素个数  
NEXT:  
INCBX      ; 指向数组的下个元素  
CMP        [BX], 00H ; 数组元素与0比较  
LOOPE     NEXT ; 若元素为0和CX≠0, 循环  
              ; 否则, 结束查找  
          ...
```



### 3. 循环控制指令

#### 3) LOOPNE/LOOPNZ 不相等或结果不为0循环

(Loop If Not Equal/Not Zero)

指令格式: LOOPNE 标号

或 LOOPNZ 标号

指令功能: LOOPNE是不相等循环, 而LOOPNZ是结果ZF≠1循环, 它们也是一对功能相同但形式不一样的指令。

- 指令执行前, 应将重复次数送入CX, 每执行一次, CX自动-1, 若-1后CX≠0和ZF=0, 则转移到标号所指定的地址重复执行; 若CX=0或ZF=1, 则退出循环, 顺序执行下一条指令。

### 例3.96

设一个由17个字符组成的字符串存放在以STRING开始的内存中，查找字符串中是否包含空格符。若没有找到空格符和尚未查完，则继续查找，直到找到第一个空格符或查完了，才退出循环。

```
STRING DB 'Personal Computer' ; 字符串
MOV BX, OFFSET STRING ; BX指向字符串始址
DEC BX ; BX-1
MOV CX, 17 ; CX=字符串长度
NEXT:
INC BX ; 指向下一个字符
CMP [BX], 20H ; 字符串元素与空格比较
LOOPNE NEXT ; 若不是空格和CX≠0, 循环
... ; 找到空格或CX已为0
```

### 3. 循环控制指令

4) JCXZ 若CX为0跳转 (Jump If CX Zero)

指令格式: JCXZ 标号

指令功能: 若CX寄存器为0, 则转移到指令中标号所指定的地址处, 否则将往下顺序执行。

- 它不对CX寄存器进行自动减1操作。
- 这条指令用在循环程序开始处。为了使程序跳过循环, 只要事先把CX 寄存器清0。

## 4. 中断指令 (Interrupt)

### 1) 中断概念

- 计算机在执行正常程序过程中，由于某些事件发生，需要暂时中止当前程序的运行，转到中断服务程序去为临时发生的事件服务。中断服务程序执行完，又返回正常程序继续运行。此过程称为中断。

8086的中断有两种：

第一种，外部中断或硬件中断，它们从8086的不可屏蔽中断引脚NMI 或可屏蔽中断引脚INTR引入。

第二种，内部中断或软中断，是为了解决CPU在运行中发生意外的外情况或是为便于对程序调试而设置的。

- 此外，也可在程序中安排一条中断指令INT n，利用它直接产生8086的内部中断。



## 4. 中断指令

### 2) 中断指令

#### (1) INT n 软件中断指令 (Interrupt)

- 软件中断指令，n为中断类型号，范围0~255。可安排在程序的任何位置上。

#### (2) INTO 溢出中断指令 (Interrupt on Overflow)

- 当带符号数进行算术运算后，若OF=1，则可由INTO指令产生类型为4的中断；若OF=0，则INTO指令不产生中断。
- 为此，在带符号数进行加减法运算之后，必须安排一条INTO指令，一旦溢出就能及时向CPU提出中断请求，CPU可做出相应的处理。



## 4. 中断指令

### (3) IRET (Interrupt Return)

**中断返回指令IRET。被安排在中断服务程序的出口处，指令执行后，从堆栈中依次弹出程序断点和FLAGS的内容，使CPU继续执行原来被打断的程序。**



## § 3.3 8086的指令系统

3.3.1 数据传送指令

3.3.2 算术运算指令

3.3.3 逻辑运算和移位指令

3.3.4 字符串处理指令

3.3.5 控制转移指令

3.3.6 处理器控制指令



## 3.3.6 处理器控制指令

### 1. 标志操作指令

- 8086提供一组标志操作指令，可直接对CF，DF和IF标志位进行设置或清除等操作，但不包含TF标志，如表3.13。

表 3.13 标志操作指令

指令助记符	操 作	指 令 名 称
CLC	$CF \leftarrow 0$	进位标志清 0 (Clear Carry)
CMC	$CF \leftarrow \overline{CF}$	进位标志求反 (Complement Carry)
STC	$CF \leftarrow 1$	进位标志置 1 (Set Carry)
CLD	$DF \leftarrow 0$	方向标志位清 0 (Clear Direction)
STD	$DF \leftarrow 1$	方向标志位置 1 (Set Direction)
CLI	$IF \leftarrow 0$	中断标志位清 0 (Clear Interrupt)
STI	$IF \leftarrow 1$	中断标志位置 1 (Set Interrupt)



# 1. 标志操作指令

## 1) CLC, CMC和STC

- 利用CLC指令，使进位标志CF清0，CMC指令使CF取反，STC指令则使CF置1。

## 2) CLD和STD

- 方向标志DF在执行字符串操作指令时用来决定地址的修改方向，CLD指令使DF清0，而STD指令则使DF置1。

## 3) CLI和STI

- 中断允许标志IF决定CPU能否响应可屏蔽中断请求，指令CLI使IF清0，禁止CPU响应这类中断。STI使IF置1，允许CPU响应。

## 2. 外部同步指令

### 1) ESC 换码指令 (Escape)

指令格式：ESC 外部操作码，源操作数

指令功能：换码指令实现8086对8087协处理器的控制。

### 2) WAIT 等待指令 (Wait)

- 通常跟在ESC指令之后。ESC指令执行后，8086 CPU处于等待状态，不断检测TEST引脚，若为高电平，则重复执行WAIT指令，处理器处于等待状态；如变为低电平，便退出等待状态，执行下条指令。

### 3) LOCK 封锁总线指令 (Lock Bus)

- 可加在任何指令的前面。凡带有LOCK前缀的指令在执行过程中，将禁止其它处理器使用总线。



### 3. 停机指令和空操作指令

#### 1) HLT 停机指令 (Halt)

- 使CPU进入暂停状态，当下列情况之一发生时，则脱离暂停状态：
  - 在RESET线上加复位信号；
  - 在NMI引脚上出现中断请求信号；
  - 在允许中断的情况下，在INTR引脚上出现中断请求信号。
- 程序中常用HLT指令来等待中断的出现。

#### 2) NOP 空操作或无操作指令 (No Operation)

- 单字节指令，执行时耗费3个时钟周期的时间，但不完成任何操作。