

第3章 8086的寻址方式和指令系统



本章主要内容：

§ 3.1 8086的寻址方式

§ 3.2 8086的指令系统



§ 3.1 8086的寻址方式



□ 指令Instruction

- 计算机的**指令**，通常包含**操作码**（Opcode）和**操作数**（Operand）两部分，操作码指出操作的性质，操作数给出操作的对象。
- **寻址方式**就是指令中说明操作数所在地址的方法。
- 指令有**单**操作数、**双**操作数和**无**操作数之分。如果是双操作数，要用逗号分开，左边的为**目的操作**，右边的为**源操作数**。

□ 寻址方式

8086的寻址方式有以下几种：

- **立即数寻址** 可直接从指令队列中取数，指令执行速度较快；
- **寄存器寻址** 操作数在寄存器中，执行速度最快；
- **存储器寻址** 操作数在存储器中，又分几种形式，执行速度较慢；
- **其它寻址** 如隐含寻址、I/O端口寻址、转移类指令寻址



§ 3.1 8086的寻址方式

3.1.1 立即寻址方式

3.1.2 寄存器寻址方式

3.1.3 直接寻址方式

3.1.4 寄存器间接寻址方式

3.1.5 寄存器相对寻址方式

3.1.6 基址变址寻址方式

3.1.7 相对基址变址寻址方式

3.1.8 其它寻址方式



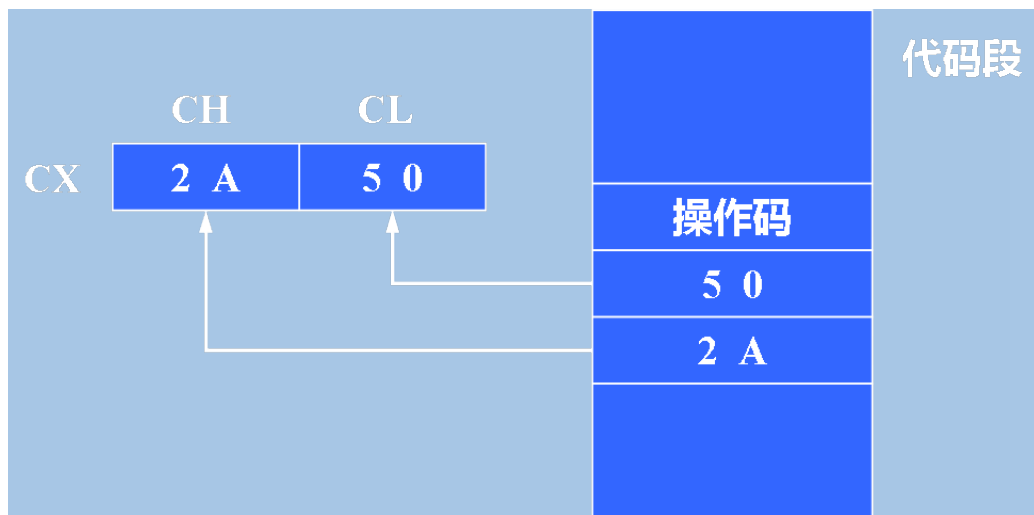
3.1.1 立即寻址方式

(Immediate Addressing)

□ 操作数直接包含在指令中，它是一个8位或16位的常数，也叫**立即数**。

例3.1 MOV AL, 26H

例3.2 MOV CX, 2A50H



□ 立即寻址规则

- 立即数可以送到寄存器中，还可送到一个存储单元(8位)中或两个连续的存储单元(16位)中去。
- 立即数只能作源操作数，不能作目的操作数。
- 以A~F打头的16进制数字出现在指令中时，前面一定要加一个数字0。

例如，将FF00H送到AX的指令必须写成：

MOV AX, 0FF00H



§ 3.1 8086的寻址方式

3.1.1 立即寻址方式

3.1.2 寄存器寻址方式

3.1.3 直接寻址方式

3.1.4 寄存器间接寻址方式

3.1.5 寄存器相对寻址方式

3.1.6 基址变址寻址方式

3.1.7 相对基址变址寻址方式

3.1.8 其它寻址方式



3.1.2 寄存器寻址方式

(Register Addressing)

❑ 操作数包含在寄存器中，由指令指定寄存器的名称。

➤ 16位寄存器可以是：

AX、BX、CX、DX、SI、DI、SP、BP

➤ 8位寄存器为：

AH、AL、BH、BL、CH、CL、DH、DL

例3.3 MOV DX, AX

设指令执行前 AX=3A68H, DX=18C7H

则指令执行后 DX=3A68H, AX=3A68H (保持不变)

□ 寄存器寻址规则

例3.4 MOV CL, AH

注意：源操作数的长度必须与目的操作数一致，否则会出错。

例如 MOV CX, AH 是错误的。



§ 3.1 8086的寻址方式

3.1.1 立即寻址方式

3.1.2 寄存器寻址方式

3.1.3 直接寻址方式

3.1.4 寄存器间接寻址方式

3.1.5 寄存器相对寻址方式

3.1.6 基址变址寻址方式

3.1.7 相对基址变址寻址方式

3.1.8 其它寻址方式



3.1.3 直接寻址方式

Direct Addressing

1.直接寻址方式

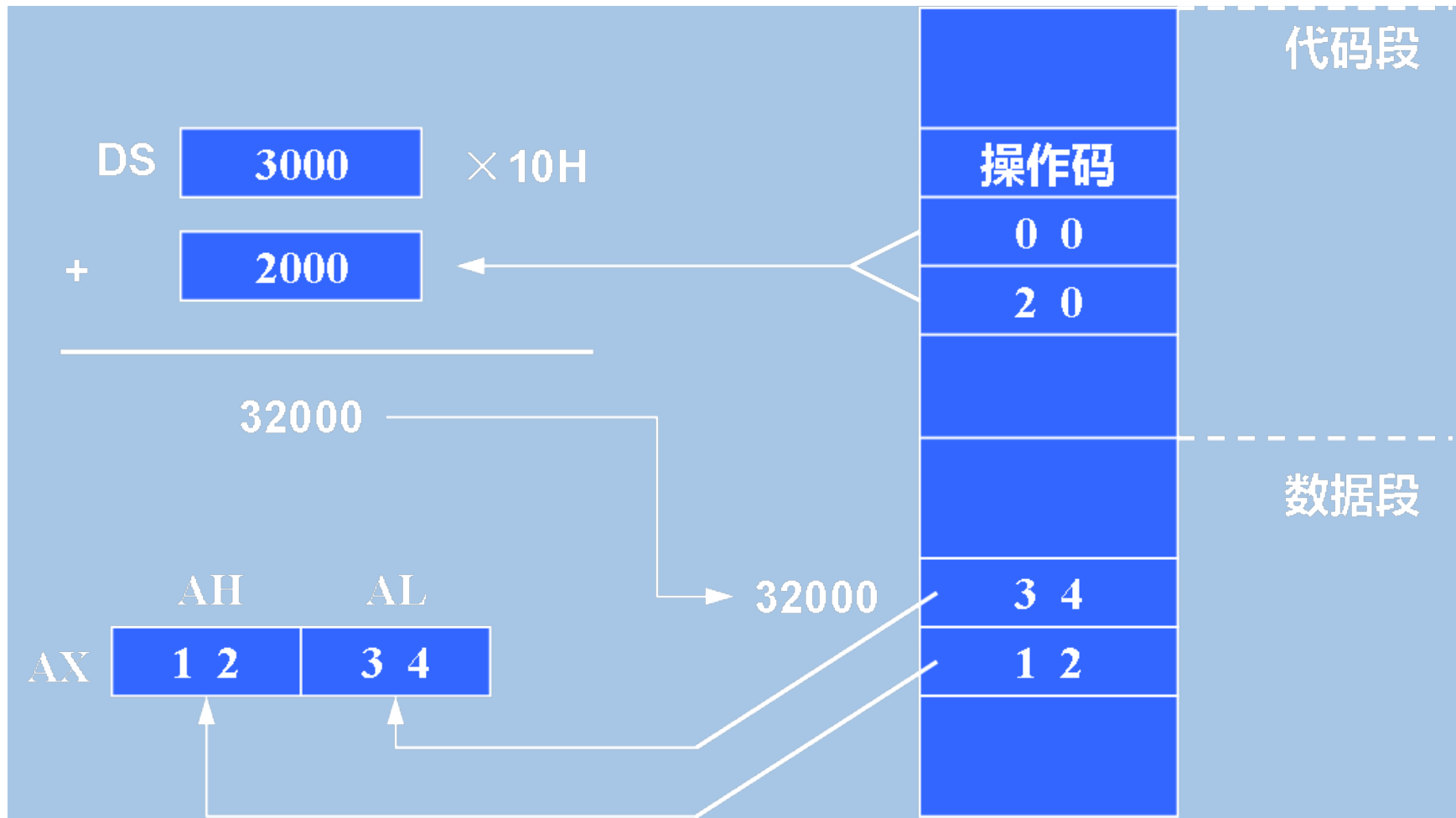
- 操作数的偏移地址也称为**有效地址EA**（Effective Address）。
- 在直接寻址方式下，存储单元的有效地址直接由指令给出，**默认**使用的段寄存器为**数据段寄存器DS**。
- 操作数的物理地址 = $16 \times DS + EA$



例3.5 MOV AX, [2000H]

- 设DS=3000H，则源操作数的物理地址
$$=16 \times 3000H + 2000H = 32000H$$
- 若(32000H)=34H，(32001H)=12H，则执行指令后AX=?
 - AX=1234H





例3.6 MOV AL, [2000H]



2. 段超越前缀

- 缺省为数据段，如果要对代码段、堆栈段或附加段寄存器所指出的存储区进行直接寻址，应在指令中指定**段超越前缀**。

例3.7 `MOV AX, ES:[500H]`

该指令的源操作数的物理地址= $16 \times \text{ES} + 500\text{H}$ 。



3. 符号地址

- 允许用**符号地址**代替**数值地址**，也就是给存储单元起一个名字，如AREA1，寻址时只要使用其名字，不必记住具体数值。

例3.8 MOV AX, AREA1

程序中事先应用**说明语句**也叫做**伪指令**定义符号地址

例3.10 AREA1 DW 0867H

...

MOV AX, AREA1

这里的DW伪指令语句用来定义变量。MOV指令执行后将AREA1单元中内容送到AX，结果AX=0867H

§ 3.1 8086的寻址方式

3.1.1 立即寻址方式

3.1.2 寄存器寻址方式

3.1.3 直接寻址方式

3.1.4 寄存器间接寻址方式

3.1.5 寄存器相对寻址方式

3.1.6 基址变址寻址方式

3.1.7 相对基址变址寻址方式

3.1.8 其它寻址方式



3.1.4 寄存器间接寻址方式

Register Indirect Addressing

- 指令中给出的寄存器中的值不是操作数本身，而是操作数的**有效地址**EA。
- 寄存器名称外必须加**方括号**，可用的寄存器有：BX、BP、SI、DI。

□ 应遵守以下约定：

约定1：如果指令中指定的寄存器是BX、SI或DI，则**默认操作数**存放在**数据段**中：

物理地址=16×DS+BX

或=16×DS+SI

或=16×DS+DI

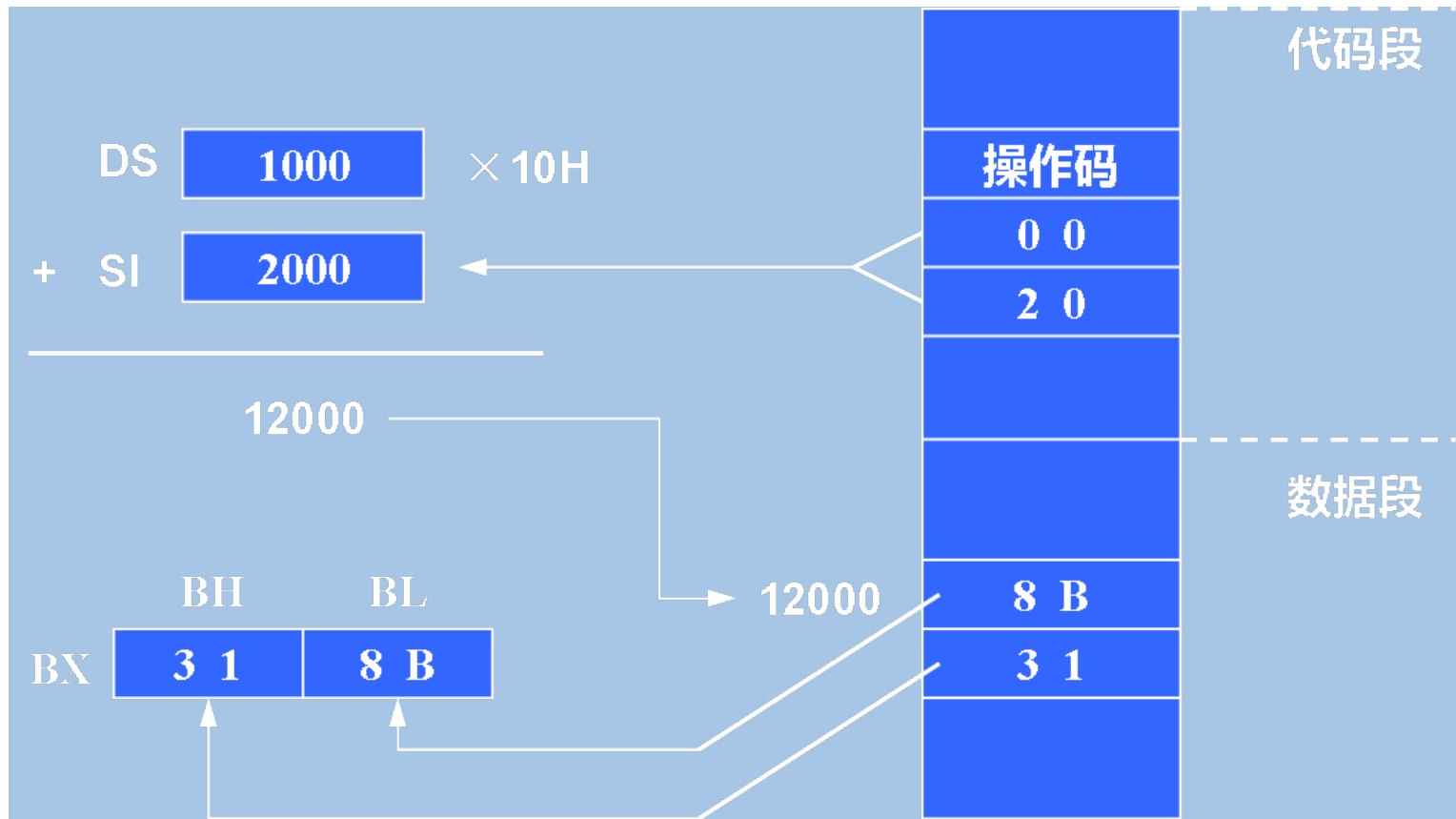


例3.11 MOV BX, [SI]

设DS=1000H, SS=3000H, SI=2000H,
(12000H)=318BH, 则物理地址=?

$$= 16 \times \text{DS} + \text{SI} = 10000\text{H} + 2000\text{H} = 12000\text{H}$$

指令执行后, **BX = ?**。



约定2: 如果指令中用**BP**进行间接寻址，则默认操作数在堆栈段中

例如:

`MOV AX, [BP]` ; 操作数的物理地址
; $=16 \times \text{SS} + \text{BP}$

指令中也可以指定段超越前缀

例如:

`MOV BX, DS:[BP]` ; 源操作数物理地址
; $=16 \times \text{DS} + \text{BP}$

`MOV AX, ES:[SI]` ; 源操作数物理地址
; $=16 \times \text{ES} + \text{SI}$



§ 3.1 8086的寻址方式

3.1.1 立即寻址方式

3.1.2 寄存器寻址方式

3.1.3 直接寻址方式

3.1.4 寄存器间接寻址方式

3.1.5 寄存器相对寻址方式

3.1.6 基址变址寻址方式

3.1.7 相对基址变址寻址方式

3.1.8 其它寻址方式



3.1.5 寄存器相对寻址方式

Register Relative Addressing

- 它与寄存器间接寻址十分相似，但在有效地址上还要加一个8/16位的**位移量**

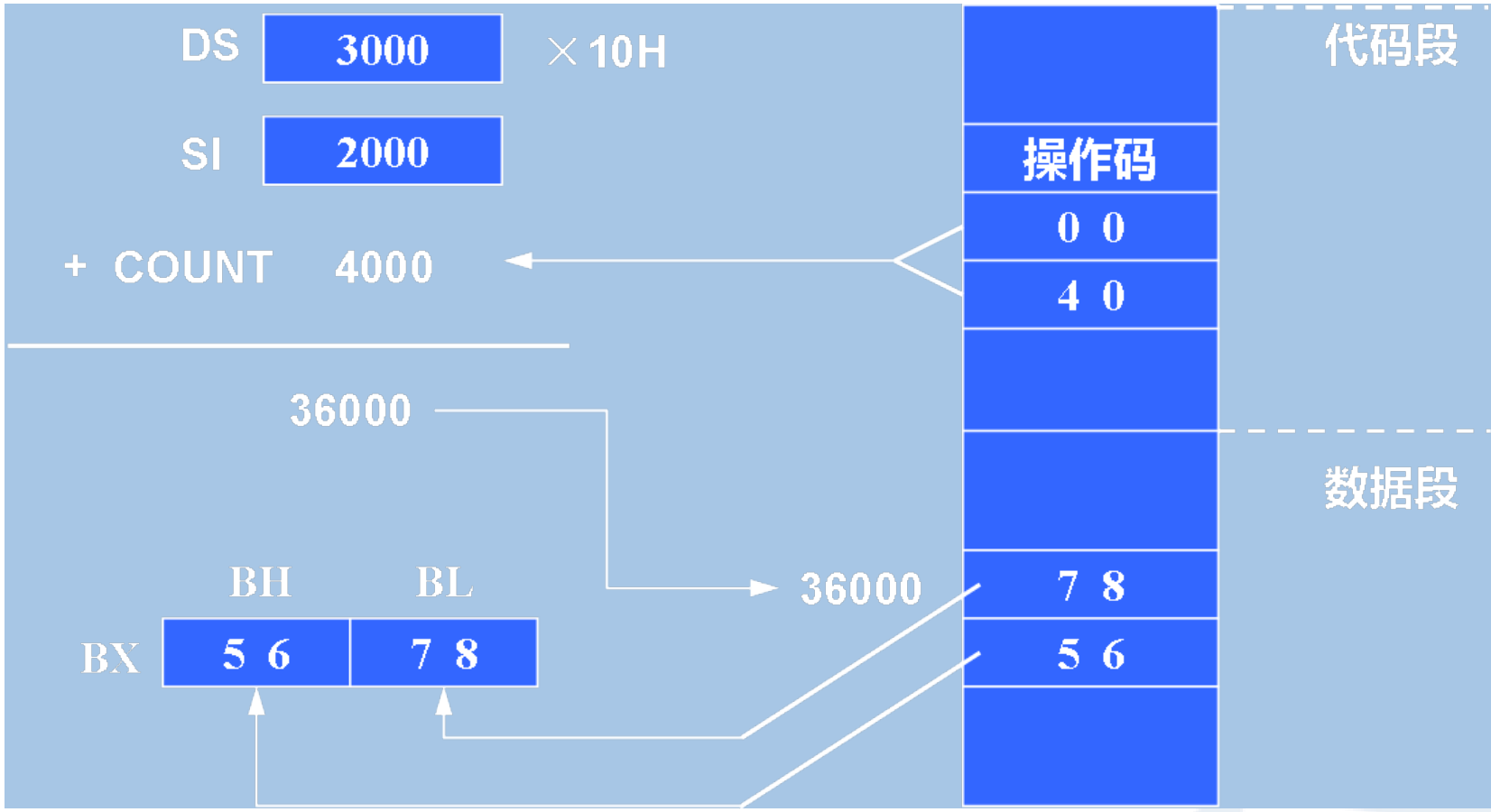
例3.12 MOV BX, COUNT[SI]

设 DS=3000H, SS=1000H, SI=2000H, 位移量
COUNT=4000H, (36000H)=5678H, 则**物理地址**=?

$$\begin{aligned}\text{物理地址} &= 16 \times \text{DS} + \text{SI} + \text{COUNT} \\ &= 30000\text{H} + 2000\text{H} + 4000\text{H} \\ &= 36000\text{H}\end{aligned}$$

执行结果 **BX**=?





§ 3.1 8086的寻址方式

3.1.1 立即寻址方式

3.1.2 寄存器寻址方式

3.1.3 直接寻址方式

3.1.4 寄存器间接寻址方式

3.1.5 寄存器相对寻址方式

3.1.6 基址变址寻址方式

3.1.7 相对基址变址寻址方式

3.1.8 其它寻址方式



3.1.6 基址变址寻址方式

Based Indexed Addressing

➤ 有效地址是一个**基址寄存器(BX或BP)**和一个**变址寄存器(SI或DI)**的内容之和，两个寄存器均由指令指定。

➤ 若基址寄存器为**BX**时，段址寄存器用**DS**，则：

$$\text{物理地址} = 16 \times \text{DS} + \text{BX} + \text{SI}$$

$$\text{或} = 16 \times \text{DS} + \text{BX} + \text{DI}$$

➤ 若基址寄存器为**BP**时，段址寄存器应使用**SS**，则：

$$\text{物理地址} = 16 \times \text{SS} + \text{BP} + \text{SI}$$

$$\text{或} = 16 \times \text{SS} + \text{BP} + \text{DI}$$

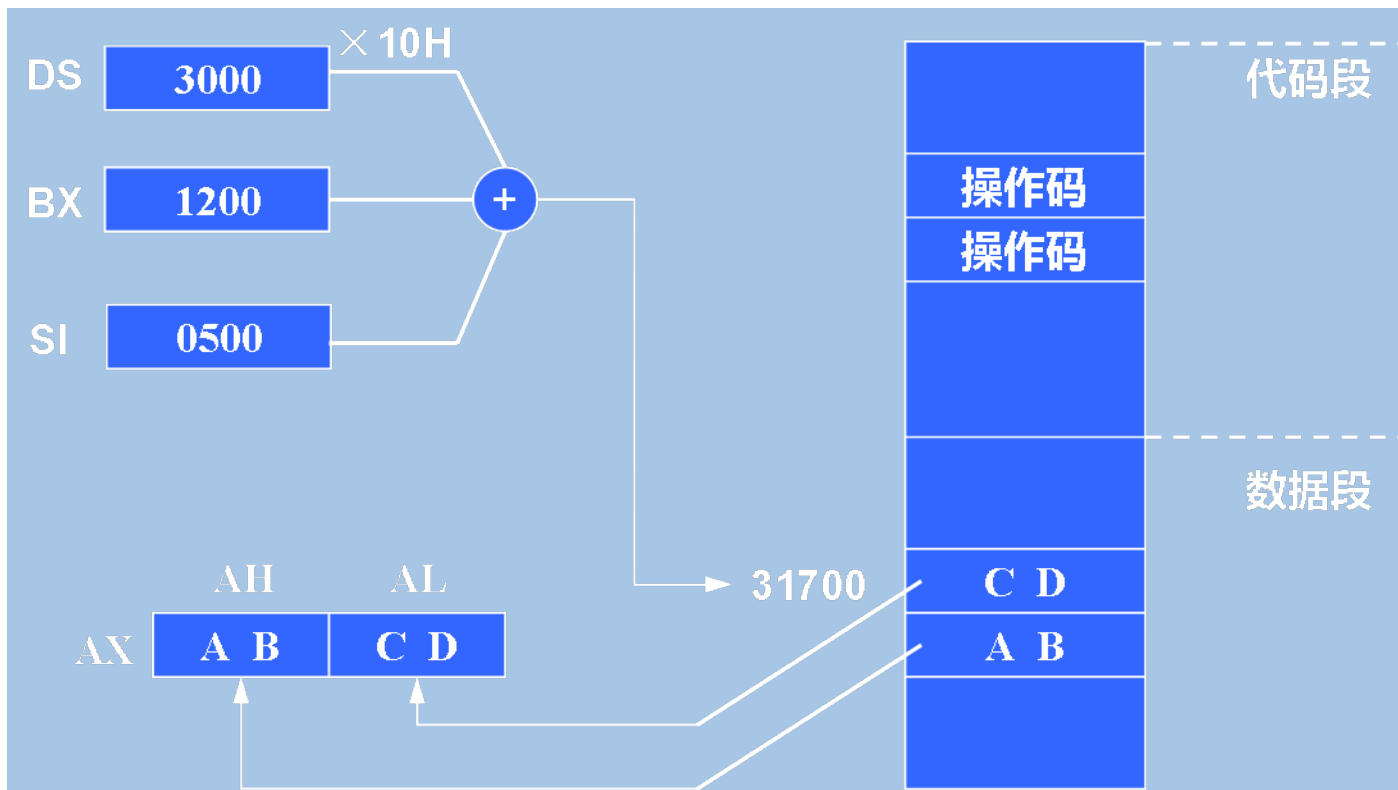
例3.13 MOV AX, [BX][SI]

设 DS=3000H, SS=2000H, BX=1200H, SI=0500H,
(31700H)=ABCDH, 则物理地址=?

物理地址 = $16 \times \text{DS} + \text{BX} + \text{SI}$

$$= 30000\text{H} + 1200\text{H} + 0500\text{H} = 31700\text{H}$$

执行结果: AX=?。



§ 3.1 8086的寻址方式

3.1.1 立即寻址方式

3.1.2 寄存器寻址方式

3.1.3 直接寻址方式

3.1.4 寄存器间接寻址方式

3.1.5 寄存器相对寻址方式

3.1.6 基址变址寻址方式

3.1.7 相对基址变址寻址方式

3.1.8 其它寻址方式



3.1.7 相对基址变址寻址方式

Relative Based Indexed Addressing

- 有效地址是基址和变址寄存器的内容，再加上8/16位位移量之和。
- 当基址寄存器为BX时，用DS作段寄存器，则：
物理地址 = $16 \times DS + BX + SI + 8\text{位或}16\text{位位移量}$
或 = $16 \times DS + BX + DI + 8\text{位或}16\text{位位移量}$
- 当基址寄存器为BP时，用SS作段寄存器，则：
物理地址 = $16 \times SS + BP + SI + 8\text{位或}16\text{位位移量}$
或 = $16 \times SS + BP + DI + 8\text{位或}16\text{位位移量}$

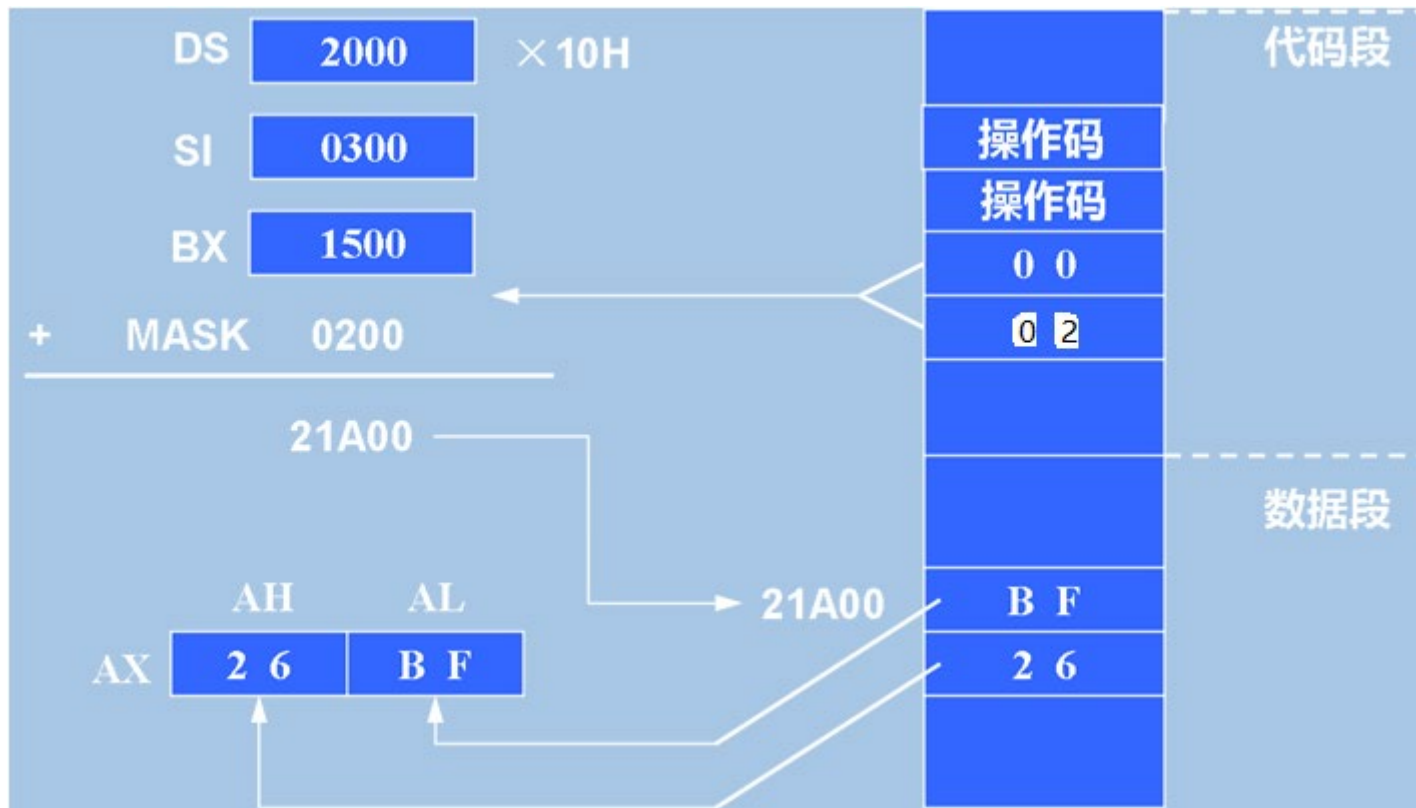
例3.14 `MOV AX, MASK[BX][SI]`

设DS=2000H, SS=3000H, BX=1500H, SI=0300H,
MASK=0200H, (21A00H)=26BFH, 则物理地址=?

物理地址=16×DS+BX+SI+MASK

=20000H+1500H+0300H+0200H=21A00H

执行结果：AX=?。



□ 涉及操作数的地址时，常使用方括号，带[]的地址必须遵循下列规则：

- (1) 立即数可以出现在方括号内，表示直接地址，例如[2000H]。
- (2) 只有BX、BP、SI、DI可以出现在[]内，既可单独出现，也可几个寄存器组合(只能相加)，或寄存器与常数相加，但BX和BP不允许出现在同个[]内，SI和DI也不能同时出现。
- (3) 方括号有相加的含义，故下面几种写法是等价的：
$$6[BX][SI] / [BX+6][SI] / [BX+SI+6]$$
- (4) 若[]内包含BP，则隐含使用SS提供基地址，它们的物理地址=16×SS+EA

§ 3.1 8086的寻址方式

3.1.1 立即寻址方式

3.1.2 寄存器寻址方式

3.1.3 直接寻址方式

3.1.4 寄存器间接寻址方式

3.1.5 寄存器相对寻址方式

3.1.6 基址变址寻址方式

3.1.7 相对基址变址寻址方式

3.1.8 其它寻址方式



3.1.8 其它寻址方式

1. 隐含寻址

指令中不指明操作数，但具有隐含规定的寻址方式。

例如， `MUL DL` ;表示 $AX \leftarrow AL * DL$



2. I/O端口寻址

8086有直接端口和间接端口两种寻址方式:

1) 直接端口寻址方式

端口地址由指令直接提供，它是一个**8位立即数** $n=00 \sim FFH$

例3.15 IN AL, 63H ; AL←端口63H中的内容

2) 间接端口寻址方式

被寻址的端口号由寄存器DX提供，端口号=0000~ FFFFH

例3.16 `MOV DX, 213H` ; DX=口地址号213H
 `IN AL, DX` ; AL←端口213H中的内容

§ 3.2 8086的指令系统

3.2.1 数据传送指令

3.2.2 算术运算指令

3.2.3 逻辑运算和移位指令

3.2.4 字符串处理指令

3.2.5 控制转移指令

3.2.6 处理器控制指令



3.2.1 数据传送指令

表 3.3 数据传送指令

通用数据传送指令	
MOV	字节或字的传送
PUSH	入栈指令
POP	出栈指令
XCHG	交换字或字节
XLAT	表转换
输入输出指令	
IN	输入
OUT	输出
地址目标传送指令	
LEA	装入有效地址
LDS	装入数据段寄存器
LES	装入附加段寄存器
标志传送指令	
LAHF	标志寄存器低字节装入 AH
SAHF	AH 内容装入标志寄存器低字节
PUSHF	标志寄存器入栈指令
POPF	出栈,并送入标志寄存器

除SAHF和POPF 指令外，对标志位均没有影响

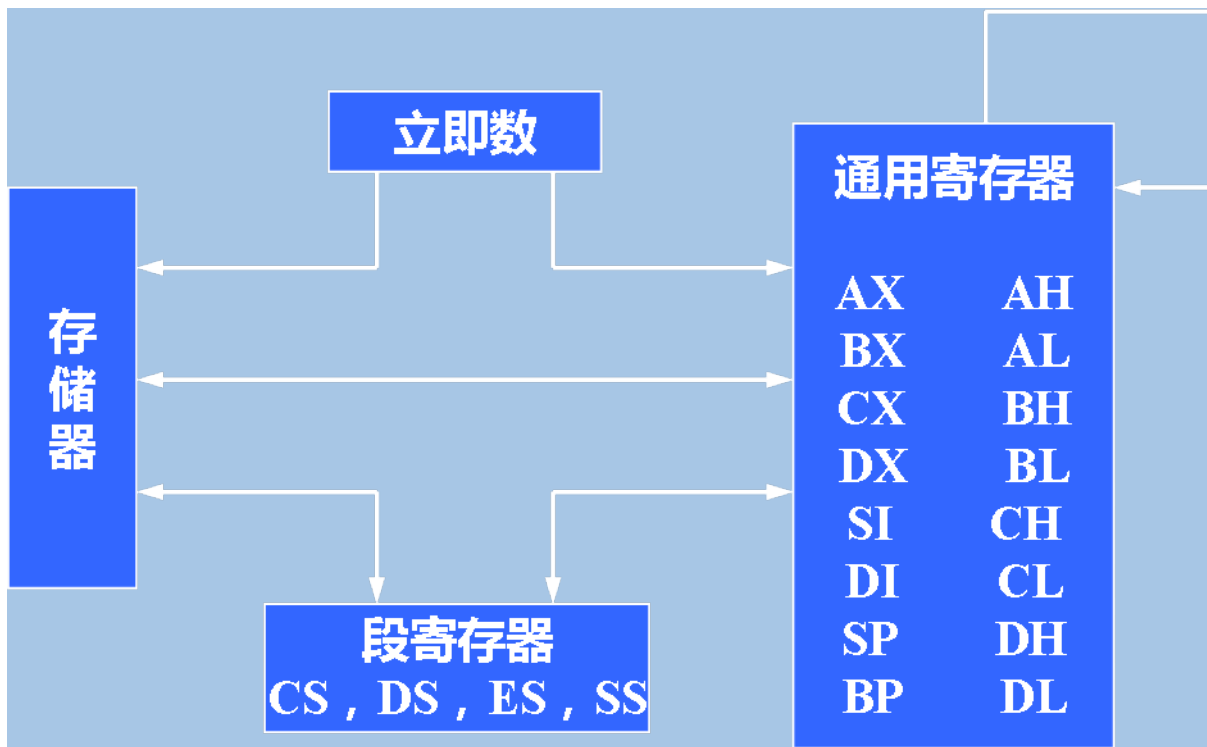


1. 通用数据传送指令 (General Purpose Data Transfer)

1) MOV 传送指令(Move)

指令格式: **MOV** 目的, 源

指令功能: 目的操作数 ← 源操作数



注意:

CS 不能做目的操作数。

立即数 不能做目的地址

指令中至少要有一项明确说明传送的是**字节**还是**字**

例3.24

```
MOV AL, 'B'
```

例3.25

```
MOV AX, DATA
```

```
MOV DS, AX
```



什么是数据段？（提前学点伪指令）

下面将举例介绍数据段的基地址、段中各变量的偏移地址、变量定义等概念

在汇编语言程序中，数据通常存放在**数据段**中

例如，下面是某个程序的**数据段**：

DATA SEGMENT ;数据段开始

AREA1 DB 14H, 3BH

AREA2 DB 3DUP(0)

ARRAY DW 3100H, 01A6H

STRING DB 'GOOD'

DATA ENDS ;数据段结束

数据段 **SEGMENT** 开始，**ENDS** 结束，**DATA** 是段名

DB 伪操作符定义**字节变量**。

DW 定义**字变量**，低字节在低地址单元，高字节在高地址单元

DUP 复制操作符，“3”说明在存储器中保留3个单元，初值均为0

数据段

- 汇编后，DATA被赋予具体的段地址，各变量将自偏移地址0000H开始依次存放，各符号地址也被赋予确定的值，等于它们在数据段中的偏移量。

AREA1的偏移地址为0000H

AREA2的偏移地址为0002H

ARRAY的偏移地址为0005H

字符串‘GOOD’从0009H开始存放

AREA1	1	4
	3	B
AREA2	0	0
	0	0
	0	0
ARRAY	0	0
	3	1
	A	6
	0	1
STRING	‘G’	
	‘O’	
	‘O’	
	‘D’	

图 3.13 数据段占用存储空间的情况

例3.26 MOV DX, OFFSET ARRAY

- 将ARRAY的**偏移地址**送到DX，其中，**OFFSET**为属性操作符，表示应把其后的符号地址的值(而不是内容)作为操作数
- 若ARRAY的值如图3.13，则指令执行后，**DX=?**

例3.27

设AREA1和AREA2的值如图3.13，说明以下指令功能

MOV AL, AREA1

MOV AREA2, AL



2) 进栈指令PUSH (Push Word onto Stack)

指令格式: **PUSH 源**

指令功能: 将源操作数推入堆栈

- 源操作数可以是16位通用寄存器、段寄存器或存储器中的数据字，但**不能是立即数**。
- 执行PUSH操作后，**先**使 $SP \leftarrow SP - 2$ ，**再**把源操作数压入SP指示的位置上。

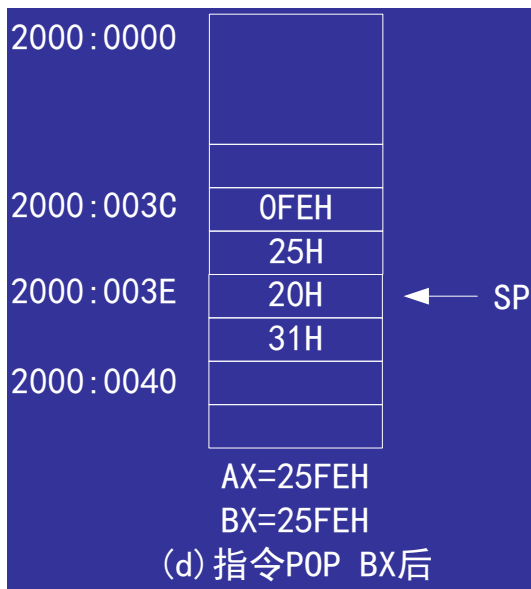
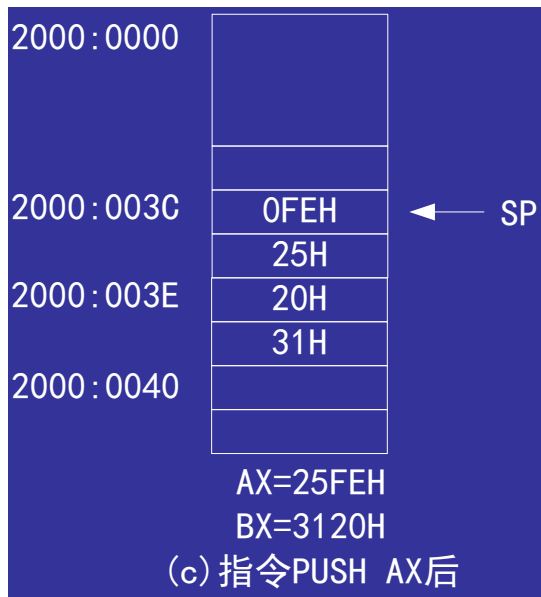
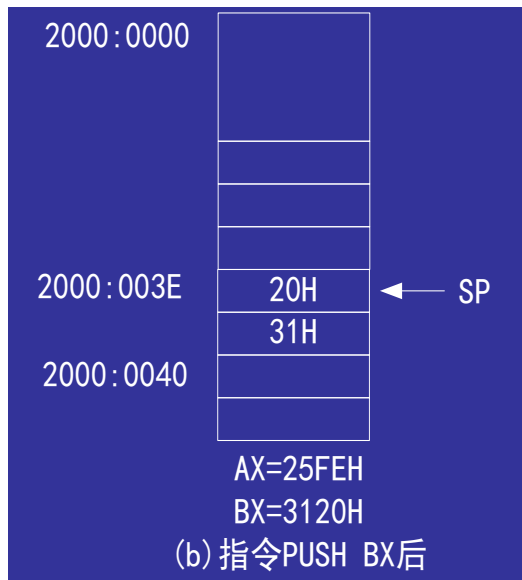
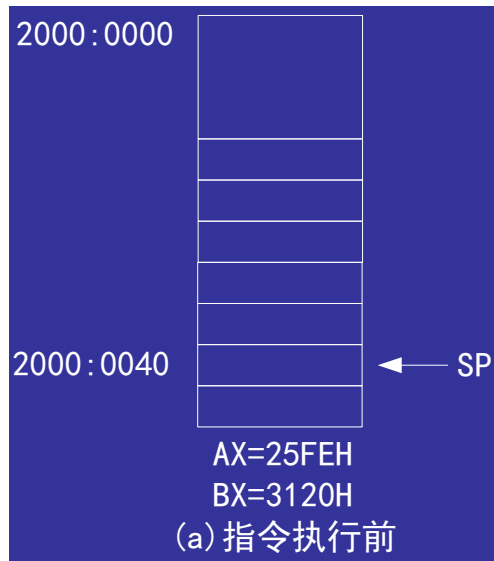


3) 出栈指令POP (Pop Word off Stack)

指令格式: POP 目的

- 指令功能: 把当前SP所指向的一个字送到目的操作数中
- 目的操作数可以是16位通用寄存器、段寄存器或存储单元, 但**不能是CS**
- 每执行一次出栈操作, **先**将SP指向的数据送到目的操作数, **再** $SP \leftarrow SP + 2$, SP向高地址方向移动, 指向新的栈顶





例3.29

设SS=2000H

SP=40H

AX=25FEH

BX=3120H

依次执行指令：

PUSH BX

PUSH AX

POP BX



4) 交换指令XCHG (Exchange)

指令格式: XCHG 目的, 源

指令功能: 源操作数和目的操作数相交换

- 交换可以在寄存器之间、寄存器与存储器之间进行，但段寄存器不能作为操作数，也不能直接交换两个存储单元中的内容。

例3.30 设AX=2000H, DS=3000H, BX=1800H, (31A00H)=1995H, 执行指令

XCHG AX, [BX+200H]

源操作数物理地址= $3000 \times 10H + 1800H + 200H = 31A00H$

指令执行后, AX= 1995H , (31A00H)= 2000H

5) 表转换指令XLAT (Table Lookup-Translation)

不要求掌握查表指令

需要了解什么是查表以及如何实现查表



□ 查表的实现

例3.31 表3.4是十进制数字0~9的LED七段码对照表，试求数字5的七段码值。

表 3.4 十进制数的七段显示码表

十进制数字	七段显示码	十进制数字	七段显示码
0	40H	5	12H
1	79H	6	02H
2	24H	7	78H
3	30H	8	00H
4	19H	9	18H



程序如下：

```
TABLE DB 40H, 79H, 24H, 30H, 19H ;七段码表格  
      DB 12H, 02H, 78H, 00H, 18H
```

...

```
MOV  SI, 5                ; SI←数字5的位移量  
MOV  BX, OFFSET TABLE   ; BX←表格首地址  
MOV  AL, [BX+SI]          ; 查表得AL=12H
```



2. 输入输出指令 (Input and Output)

1) IN 输入指令 (Input)

指令格式:

- ① **IN AL, 端口地址** ; AL←从8位端口读入1字节
- 或 **IN AX, 端口地址** ; AX←从16位端口读入1个字
- ② **IN AL, DX** ; 端口地址存放在DX中
- 或 **IN AX, DX**

- 格式①，**端口地址(00~FFH)**直接包含在IN指令里，共允许寻址256个端口
- 当端口地址大于FFH时，必须用格式②寻址，即先将端口号送入DX，再执行输入操作，**DX允许范围0000~FFFFH**



例3.32

IN AL, 0F1H ; AL←从F1H端口读入1字节

IN AX, 80H ; AL←80H端口内容
 ; AH←81H端口内容

MOV DX, 310H ; 端口地址310H先送入DX

IN AL, DX ; AL←310H端口内容



例3.33

IN指令中也可用符号表示地址。例如，要求从一个模/数(A/D)转换器读入1字节数字量到AL中

ATOD EQU 54H	; A/D转换器端口地址为54H
IN AL, ATOD	; 将54H端口的内容读入AL中



2) OUT 输出指令 (Output)

指令格式:

- ① OUT 端口地址, AL ; 8位端口←AL内容
或 OUT 端口地址, AX ; 16位端口←AX内容

- ② OUT DX, AL ; DX=端口地址
或 OUT DX, AX



例3.34

OUT 85H, AL ; 85H端口←AL内容

MOV DX, 0FF4H ; DX指向端口0FF4H

OUT DX, AL ; FF4H端口←AL内容

MOV DX, 300H ; DX指向16位端口

OUT DX, AX ; 300H端口←AL内容

; 301H端口←AH内容



3. 地址目标传送指令 (Address Object Transfers)

这是一类专用于传送地址码的指令，可以用来传送操作数的段地址和偏移地址

1) LEA 取有效地址指令 (Load Effective Address)

指令格式：LEA 目的, 源

指令功能：取源操作数地址的偏移量，送到目的操作数



例3.35 设：SI=1000H，DS=5000H，(51000H)=1234H，
指令执行结果如下：

LEA BX, [SI]

; [SI]的偏移地址为1000H，BX←1000H

MOV BX, [SI]

; 偏移地址为1000 H单元的内容为1234H，

; 指令执行后，BX←1234H

例3.36 下面两条指令是等价的，它们都取TABLE的偏移地址，送到BX中。

LEA BX, TABLE

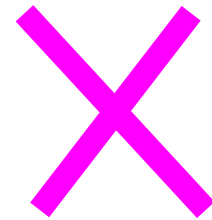
MOV BX, OFFSET TABLE



例3.37 某数组含20个元素，每个元素占一个字节，序号为0~19。设DI指向数组开头处，如要把序号为6的元素的偏移地址送到BX中：

LEA BX, 6[DI]

MOV BX, OFFSET 6[DI]



2) LDS 将双字指针送到寄存器和DS指令

(Load Pointer using DS)

指令格式: **LDS** 目的, 源

指令功能: 从源操作数指定的存储单元中, 取出1个4字节地址指针, 送进目的寄存器DS和指令中指定的目的寄存器中。

➤ 源操作数必须是存储单元, 目的操作数必须是16位寄存器, **常用SI寄存器**, 但**不能用段寄存器**。

例3.38 设: DS=1200H, (12450H)=F346H,
(12452H)=0A90H。执行指令:

LDS SI, [450H]

结果: 存储单元前2字节内容为F346H, **SI←F346H**
后2字节内容为0A90H, **DS←0A90H**

3) LES 将双字指针送到寄存器和ES指令 (Load Pointer using ES)

指令格式: **LES** 目的, 源

指令功能: 与 LDS 指令的操作基本相同, 但段寄存器为 **ES**, 目的操作数常用**DI**。

例3.39 设DS=0100H, BX=0020H,
(01020H)=0300H, (01022H)=0500H

LES DI, [BX]

; 存储单元前2字节内容为0300H, **DI←0300H**

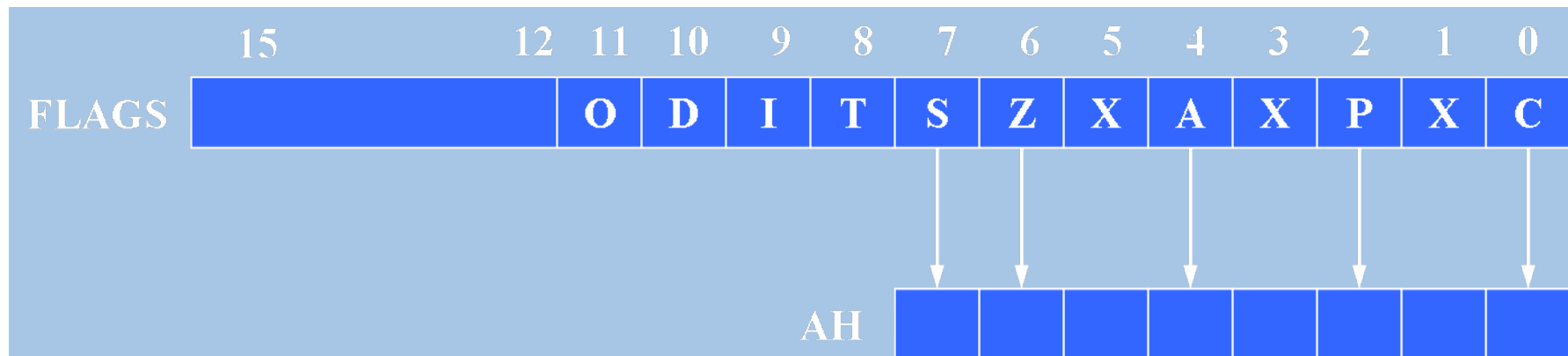
; 后2字节内容为0500H, **ES←0500H**

4. 标志传送指令(Flag Transfers)

1) LAHF 标志送到AH指令 (Load AH from Flags)

指令格式: **LAHF**

指令功能: 把标志寄存器的**SF**、**ZF**、**AF**、**PF**和**CF**传送到AH寄存器的相应位。



2) SAHF AH送标志寄存器 (Store AH into Flags)

指令格式: **SAHF**

指令功能: 把AH内容存入标志寄存器。指令功能与LAHF 的操作相反。

3) PUSHF 标志入栈指令 (Push Flags onto Stack)

指令格式: **PUSHF**

指令功能: $SP \leftarrow SP - 2$, 把整个标志寄存器的内容推入堆栈。



4) POPF 标志出栈指令 (Pop Flags off Stack)

指令格式: **POPF**

指令功能: 把SP所指的一个字, 传送给标志寄存器
FLAGS, 并使 $SP \leftarrow SP + 2$ 。



§ 3.2 8086的指令系统

3.2.1 数据传送指令

3.2.2 算术运算指令

3.2.3 逻辑运算和移位指令

3.2.4 字符串处理指令

3.2.5 控制转移指令

3.2.6 处理器控制指令



3.2.2 算术运算指令

- 算术运算指令可处理4种类型的数：
 - 无符号二进制整数
 - 带符号二进制整数
 - 无符号压缩十进制整数(Packed Decimal)
 - 无符号非压缩十进制整数(Unpacked Decimal)
- 二进制数可以是8位或16位，如果是**带符号数**，则用**补码**表示。
- **压缩十进制数** 在一个字节中存放两个BCD码十进制数
- **非压缩十进制数** 低半字节存放一个十进制数，高半字节为全零。



表 3.5 4 种类型数的表示方法

二进制码(B)	十六进制(H)	无符号二进制(D)	带符号二进制(D)	非压缩十进制	压缩十进制
0000 0111	07	7	+7	7	07
1000 1001	89	137	-119	无效	89
1100 0101	C5	197	-59	无效	无效



表 3.6 算术逻辑指令

加 法	
ADD	加法
ADC	带进位的加法
INC	增量
AAA	加法的 ASCII 调整
DAA	加法的十进制调整
减 法	
SUB	减法
SBB	带借位的减法
DEC	减量
NEG	取负
CMP	比较
AAS	减法的 ASCII 调整
DAS	减法的十进制调整

乘 法	
MUL	无符号数乘法
IMUL	整数乘法
AAM	乘法的 ASCII 调整
除 法	
DIV	无符号数除法
IDIV	整数除法
AAD	除法的 ASCII 调整
CBW	把字节转换成字
CWD	把字转换成双字

十进制调整指令不做介绍



1. 加法指令 (Addition)

1) ADD 加法指令

指令格式: **ADD** 目的, 源

指令功能: 目的 \leftarrow 源+目的

2) ADC 带进位的加法指令 (Addition with Carry)

指令格式: **ADC** 目的, 源

指令功能: 目的 \leftarrow 源+目的+CF

- 它们的**源操作数**可以是寄存器、存储器或立即数。
- **目的操作数**只能用寄存器和存储单元。
- 源和目的操作数**不能同时为存储器**, 而且它们的**类型必须一致**, 即都是字节或字

例3.40

ADD AL, 18H	; $AL \leftarrow AL + 18H$
ADC BL, CL	; $BL \leftarrow BL + CL + CF$
ADC AX, DX	; $AX \leftarrow AX + DX + CF$
ADD AL, COST[BX]	
ADD COST[BX], BL	

它们影响标志位： **CF、OF、PF、SF、ZF和AF**



例3.41 试用加法指令对两个8位16进制数5EH和3CH求和，分析指令执行后对标志位的影响。

程序如下：

MOV	AL, 5EH	; AL=5EH (94)
MOV	BL, 3CH	; BL=3CH (60)
ADD	AL, BL	; 结果AL=9AH



例3.41

运算后的标志位：

ZF=0，运算结果非0

AF=1，低4位向高4位有进位

CF=0，D7位没有向前产生进位

SF=1，D7=1

PF=1，结果中有偶数个1

OF=1

$$\begin{array}{r} 0101 \quad 1110 \\ + \quad 0011 \quad 1100 \\ \hline 1001 \quad 1010 \end{array}$$

如何对这些标志进行解释，取决于编写的程序，或者说是人为决定的。



3) INC 增量指令 (Increment)

指令格式: INC 目的

指令功能: 目的 \leftarrow 目的 + 1

- 目的操作数可以是通用寄存器或内存。指令执行后影响AF、OF、PF、SF和ZF，但进位标志CF 不受影响



例3.42 INC指令的例子

INC BL ; BL寄存器中内容增1

INC CX ; CX寄存器中内容增1

指令中只有一个操作数，如果是**内存单元**，则要用PTR操作符说明是字还是字节

例3.43

INC **BYTE PTR**[BX]

INC **WORD PTR**[BX]



2. 减法指令 (Subtraction)

1) SUB 减法指令 (Subtraction)

指令格式: SUB 目的, 源

指令功能: 目的 \leftarrow 目的 - 源

例3.48

SUB AX, BX ; AX \leftarrow AX - BX

SUB DX, 1850H ; DX \leftarrow DX - 1850H

SUB BL, [BX]



2) SBB 带借位的减法指令 (Subtract with Borrow)

指令格式: SBB 目的, 源

指令功能: 目的 \leftarrow 目的 - 源 - CF

例3.49

SBB AL, CL ; $AL \leftarrow AL - CL - CF$

► SBB主要用于多字节减法中



3) DEC 减量指令 (Decrement)

指令格式: DEC 目的

指令功能: 目的 \leftarrow 目的 - 1

例3.50

DEC BX

DEC WORD PTR[BX]



4) NEG 取负指令 (Negate)

指令格式: NEG 目的

指令功能: 目的 $\leftarrow 0 - \text{目的}$

例3.51

NEG AX

NEG BYTE PTR[BX]



5) CMP 比较指令 (Compare)

指令格式: **CMP** 目的, 源

指令功能: 目的 – 源

- 结果不回送到目的, 仅反映在标志位上

例3.52

CMP AL, 80H

CMP BX, DATA1

比较指令主要用在希望比较两个数的大小, 而又不破坏原操作数的场合



3. 乘法指令 (Multiply)

1) MUL 无符号数乘法指令 (Multiply)

指令格式: **MUL 源**

指令功能: 把源操作数和累加器中的数, 都当成无符号数, 然后将两数相乘。

- 其中有一个操作数**一定是累加器**
- 如果源操作数是1个字节, 则 $AX \leftarrow AL * \text{源}$
- 若源操作数是1个字, 则 $(DX, AX) \leftarrow AX * \text{源}$
- 源操作数可以是寄存器或存储单元, **不能是立即数**
- 当源操作数是存储单元时, 应在操作数前加BYTE或WORD, **说明是字节还是字**



例3.56

```
MUL    DL                ; AX←AL*DL
MUL    CX                ; (DX, AX)←AX*CX
MUL    BYTE PTR[SI]      ; AX←AL*(内存中某字节),
                        ; BYTE说明字节乘法
MUL    WORD PTR[BX]      ; (DX, AX)←AX*(内存中
                        ; 某字), WORD说明字乘法
```

- MUL指令执行后影响**CF**和**OF**标志
- 如果结果的高半部分不全为**0**，则CF、OF均置1。否则，CF、OF均清0
- 通过测试这两个标志，可检测并去除结果中的**无效前导零**

例3.57

设AL=55H， BL=14H， 计算它们的积。

只要执行下面这条指令：

`MUL BL`

结果：AX=06A4H

由于AH=06H≠0， 高位部分有效， 所以置CF=1
和OF=1



2) IMUL 整数乘法指令 (Integer Multiply)

指令格式: **IMUL 源**

指令功能: 把源操作数和累加器中的数, 都作为带符号数, 进行相乘。

例3.59 设AL=-28, BL=59, 试计算它们的乘积。

IMUL BL ; AX=F98CH= - 1652, CF=1, OF=1



4. 除法指令 (Division)

1) DIV 无符号数除法指令 (Division, unsigned)

指令格式: **DIV** 源

指令功能: 对两个无符号二进制数进行除法操作。

- 如果源操作数为**字节**, 被除数必须放在**AX**中, 并且有:

$AL \leftarrow AX / \text{源(字节)的商}$

$AH \leftarrow AX / \text{源(字节)的余数}$

- 要是被除数只有8位, 必须把它放在AL中, 并将AH清0, 然后相除。



- 若源操作数为字，被除数必须放在DX和AX中，并且有：

$AX \leftarrow (DX, AX) / \text{源(字)的商}$

$DX \leftarrow (DX, AX) / \text{源(字)的余数}$

- 要是被除数只有16位，除数也是16位，则必须将16位被除数送入AX，再将DX清0，然后相除。
- 与被除数和除数一样，商和余数都是无符号数



2) IDIV 整数除法指令 (Integer Division)

指令格式: IDIV 源

指令功能: 功能与DIV相同, 但操作数都必须是带符号数, 商和余数也都是带符号数, 而且规定余数的符号和被除数的符号相同

- 进行除法操作时, 如果商超过了目标寄存器AL或AX所能存放数的范围, 计算机会自动产生**除法错中断**, 相当于执行了除数为0的运算, 所得的商和余数都不确定



□ 除法指令溢出举例

例3.61 两个无符号数7A86H和04H相除的商，应为1EA1H。若用DIV指令进行计算，即

```
MOV  AX, 7A86H
```

```
MOV  BL, 04H
```

```
DIV  BL
```

这时，由于BL中的除数04H为字节，而被除数为字，商1EA1H大于AL中能存放的最大无符号数FFH，结果将产生**除法错误中断**



- 对于带符号数除法指令，字节操作时要求被除数为16位，字操作时要求被除数为32位
- 如果被除数不满足这个条件，不能简单地将高位置0，而应该先用下面的符号扩展指令 (Sign Extension)将被除数转换成除法指令所要求的格式，**CBW**、**CWD**，再执行除法指令

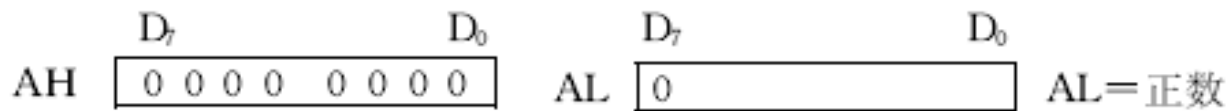


3)CBW 把字节转换为字指令 (Convert Byte to Word)

指令格式: **CBW**

指令功能: 把AL中字节的符号位扩充到AH的所有位, 这时AH被称为是AL 的符号扩充。

如果AL中的D7=0, 就将这个0扩展到AH中去, 使AH=00H, 即



若AL中的D7=1, 则将这个1扩展到AH中去, 使AH=FFH, 即



CBW指令执行后, 不影响标志位。



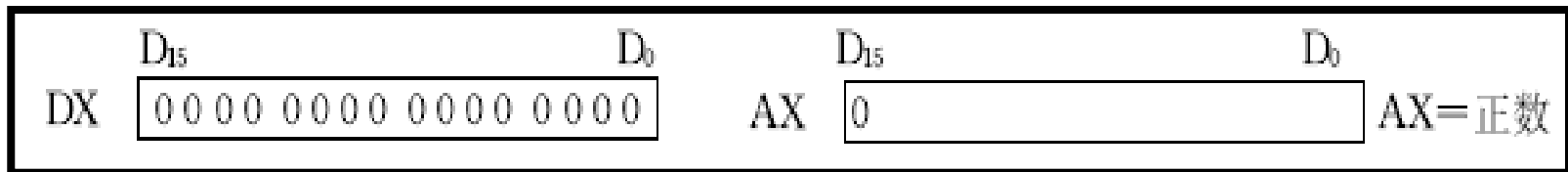
4) CWD 把字转换成双字指令

(Convert Word to Double Word)

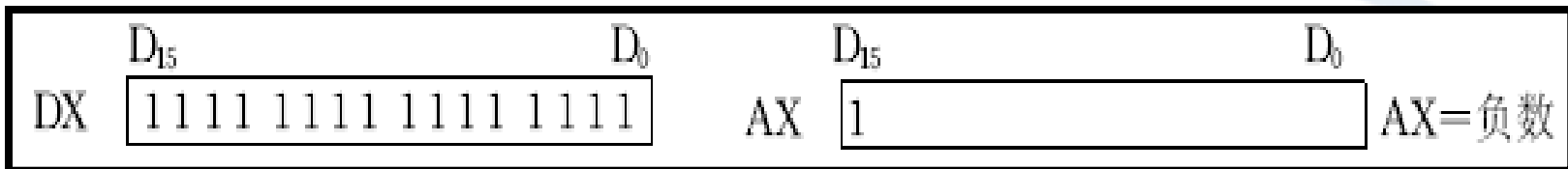
指令格式: CWD

指令功能: 把AX中字的符号位扩充到DX寄存器的所有位中去。

若AX中的D15=0, 则 $DX \leftarrow 0000H$, 即




若AX中的D15=1, 则 $DX \leftarrow FFFFH$, 即



例3.62 编程求-38/3的商和余数

```
MOV  AL, 11011010B      ; 被除数-38
MOV  CH, 00000011B      ; 除数+3
CBW                          ; 将AL符号扩展到AH中
                        ; 使AX=1111 1111 1101 1010B
IDIV  CH                  ; AX/CH
                        ; AL=1111 0100B = -12 (商)
                        ; AH=1111 1110B = -2(余数)
```



§ 3.2 8086的指令系统

3.2.1 数据传送指令

3.2.2 算术运算指令

3.2.3 逻辑运算和移位指令

3.2.4 字符串处理指令

3.2.5 控制转移指令

3.2.6 处理器控制指令



3.2.3 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 AX

NOT BL

NOT BYTE PTR[BX]



- 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 \leftarrow 目的 \wedge 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。

AND AX, 0F0FH ; AX \leftarrow 0508H。

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

OR AX, 3030H ; AX \leftarrow 3538H



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 \otimes 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

5) TEST 测试指令 (Test)

指令格式: TEST 目的, 源

指令功能: 目的 \wedge 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。



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

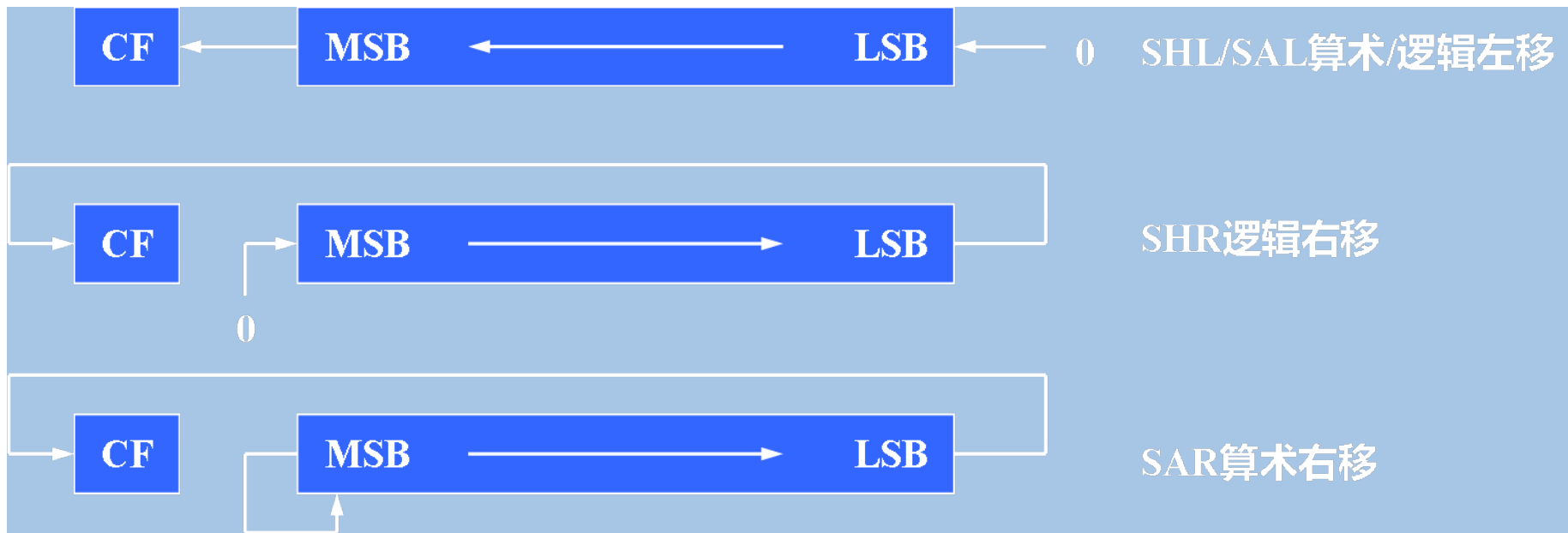
TEST AL, 40H ; D7=0, D6=1?

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，如图下图。



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)



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



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SAL/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

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**

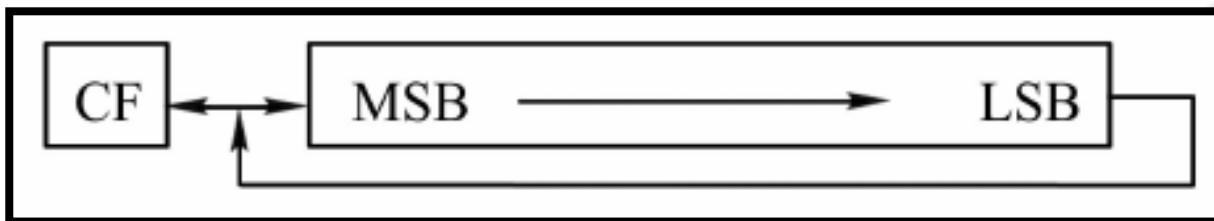
1) ROL 循环左移指令 (Rotate Left)

指令格式： ROL 目的, 计数值



2) ROR 循环右移指令 (Rotate Right)

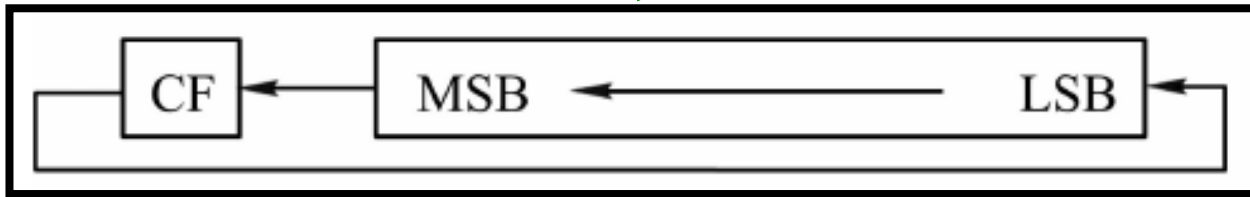
指令格式： ROR 目的, 计数值



3) RCL 通过进位位循环左移

(Rotate through Carry Left)

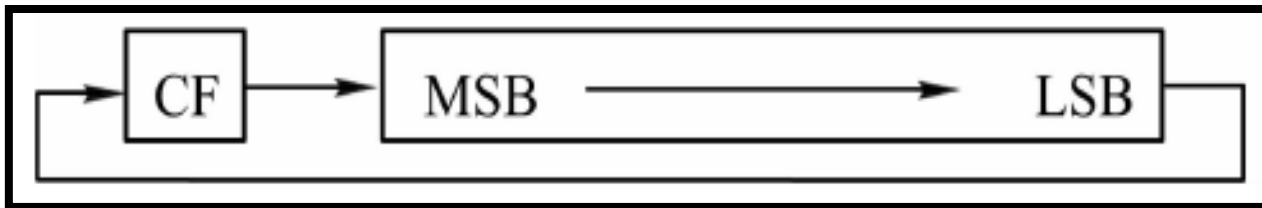
指令格式: RCL 目的, 计数值



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



□ 循环移位指令特点

- 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

□ 循环移位指令举例

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

§ 3.2 8086的指令系统

3.2.1 数据传送指令

3.2.2 算术运算指令

3.2.3 逻辑运算和移位指令

3.2.4 字符串处理指令 不做介绍

3.2.5 控制转移指令

3.2.6 处理器控制指令



§ 3.2 8086的指令系统

3.2.1 数据传送指令

3.2.2 算术运算指令

3.2.3 逻辑运算和移位指令

3.2.4 字符串处理指令

3.2.5 控制转移指令

3.2.6 处理器控制指令



3.2.5 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，如下表。



表 3.9 控制转移指令

无条件转移和过程调用指令	
JMP	无条件转移
CALL	过程调用
RET	过程返回
条 件 转 移	
JZ/JE 等 10 条指令	直接标志转移
JA/JNBE 等 8 条指令	间接标志转移
条 件 循 环 控 制	
LOOP	CX≠0 则循环
LOOPE/LOOPZ	CX≠0 和 ZF=1 则循环
LOOPNE/LOOPNZ	CX≠0 和 ZF=0 则循环
JCXZ	CX=0 则转移
中 断	
INT	中断
INTO	溢出中断
IRET	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**。CS的值不变
- **段间转移**, 又称**远(FAR)转移**。CS和IP的值都要改变



- 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

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]



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
- 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



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



- 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



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← PROG_F的段内偏移量

CS←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

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



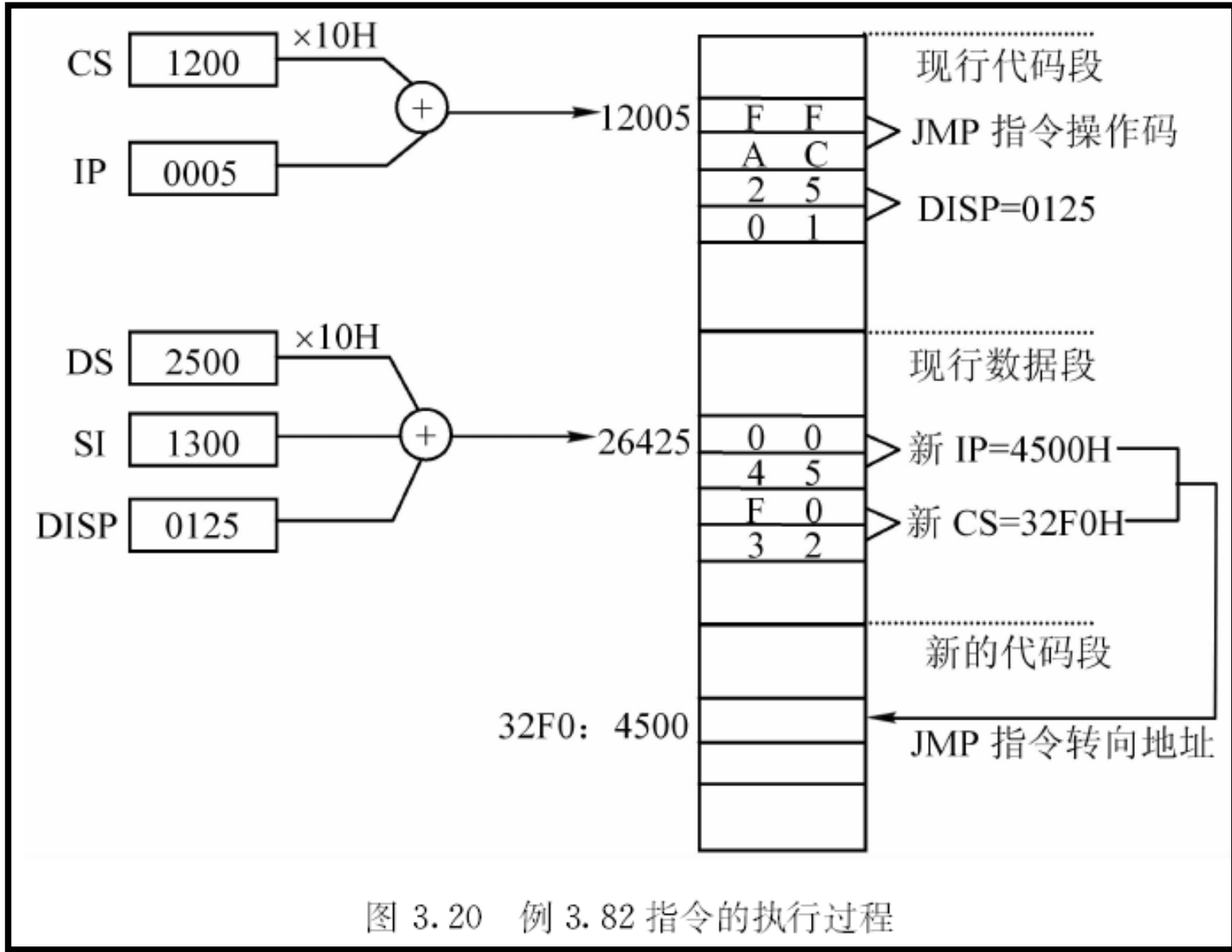


图 3.20 例 3.82 指令的执行过程

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

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 $SP \leftarrow SP + 2$



①段内直接调用和返回

例3.84

CALL PROC_N



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)



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



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)。



指令CALL PROG_N的执行过程如图3.21(a)

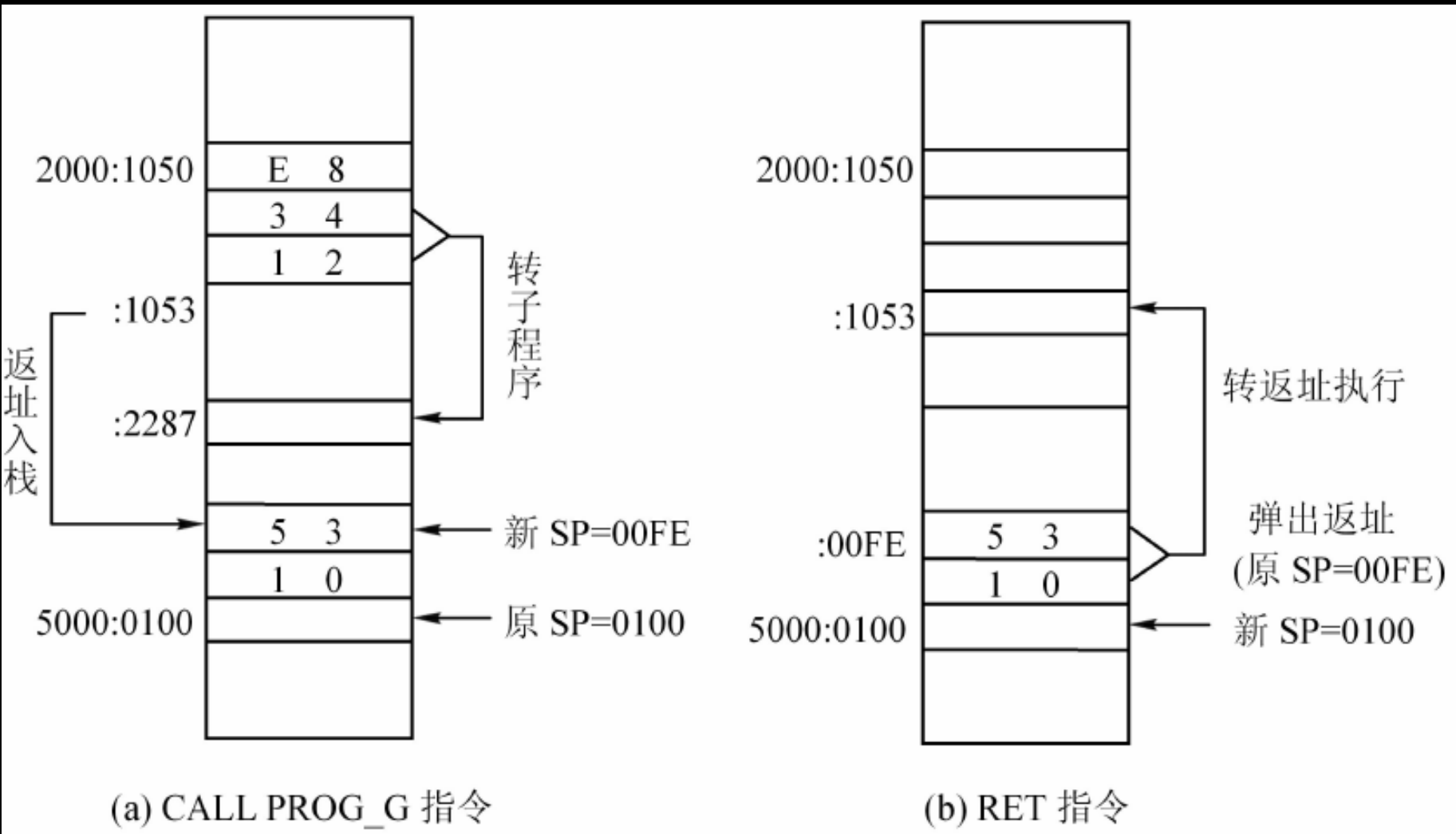


图 3.21 段内直接调用和返回指令执行情况

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]

$SP \leftarrow SP - 2$

IP入栈

$IP \leftarrow EA$ ，计算出目的地址的有效地址EA，送入IP



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

$$(16 \times DS + BX + SI) = (10000H + 0200H + 0300H)$$

$$= (10500H) = 3210H, \text{ 即 } EA = 3210H$$

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

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

执行远调用CALL指令的过程：

SP←SP-2 即SP=0050H-2=004EH

CS入栈 即CS=1000H入栈

SP←SP-2 即SP←004CH

IP入栈 即IP=205FH入栈

转子程序入口 将PROG-F的段地址和
偏移地址送CS：IP

即CS←3000H, IP←0500H

执行子程序PROG-F



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：

$IP \leftarrow \text{栈中内容}$

$IP \leftarrow 205FH$

$SP \leftarrow SP + 2$

$SP \leftarrow 004CH + 2 = 004EH$

$CS \leftarrow \text{栈中内容}$

$CS \leftarrow 1000H$

$SP \leftarrow SP + 2$

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

所以程序转返回地址CS: IP=1000: 205FH处执行



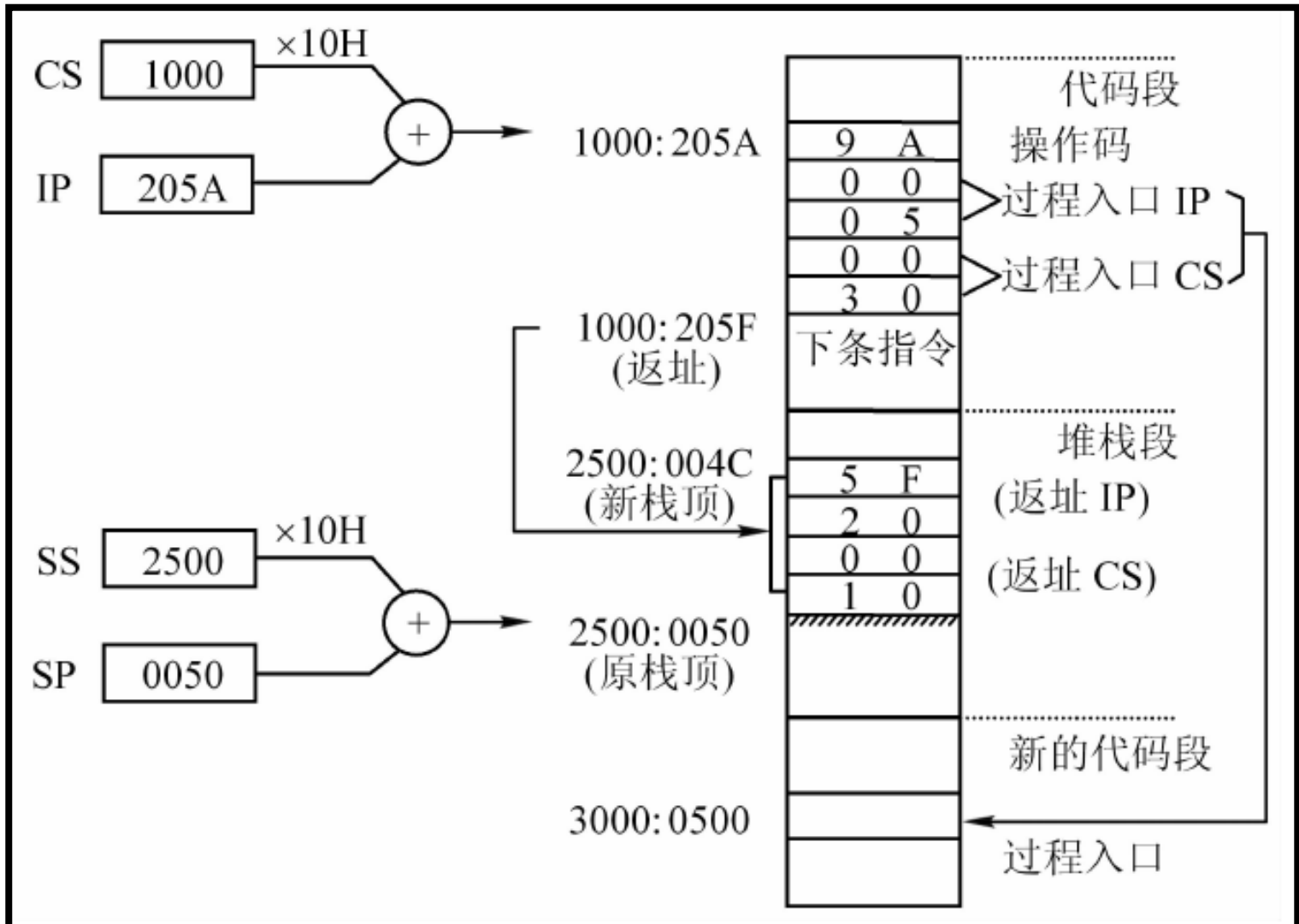


图 3.22 段间调用指令执行过程

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



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
- 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

CF、ZF、SF、OF、PF

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

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) 间接标志转移

- **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，每条指令都有两种不同的助记符



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



例3.89 设 AL=F0H, BL=35H, 执行指令

CMP AL, BL ; AL-BL, 但是不改变AL的值

JAE NEXT ; AL大于等于BL, 则转到NEXT



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

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°C时，打开加热器；温度升到100°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, 00H	; 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

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
```



AGAIN:

```
CMP  BYTE PTR TABLE[BX], 0    ; 取一个数与0比
JGE   GRET_EQ                  ;  $\geq 0$ , 转GRET_EQ
INC   NEG1                        ;  $< 0$ , 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 循环控制类指令
- 不要求掌握



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



2) 中断指令

①INT 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。

②INTO 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



③IRET (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。



§ 3.2 8086的指令系统

3.2.1 数据传送指令

3.2.2 算术运算指令

3.2.3 逻辑运算和移位指令

3.2.4 字符串处理指令

3.2.5 控制转移指令

3.2.6 处理器控制指令



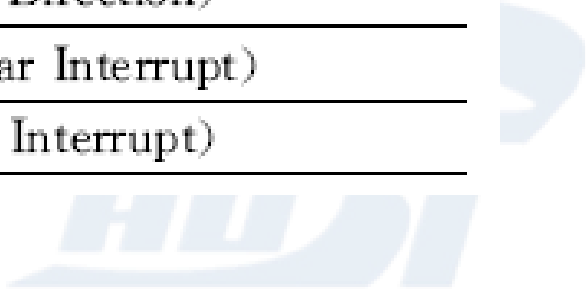
3.2.6 处理器控制指令

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

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) 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

不要求掌握



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