M/

《微型计算机原理与接口技术》
第6版

第3章
8086的寻址方式和指令系统

§3.3　8086的指令系统

- 8086的指令共有六大类：数据传送指令、算术运算指令、逻辑运算和移位指令、字符串处理指令、控制转移指令、处理器控制指令。包含133条基本指令，与寻址方式，再加上不同的数据形式，可构成上千种指令

- 本章除详细介绍各类指令外，还将介绍部分伪指令，并给出许多短小的程序设计例子,以便更好理解指令功能。

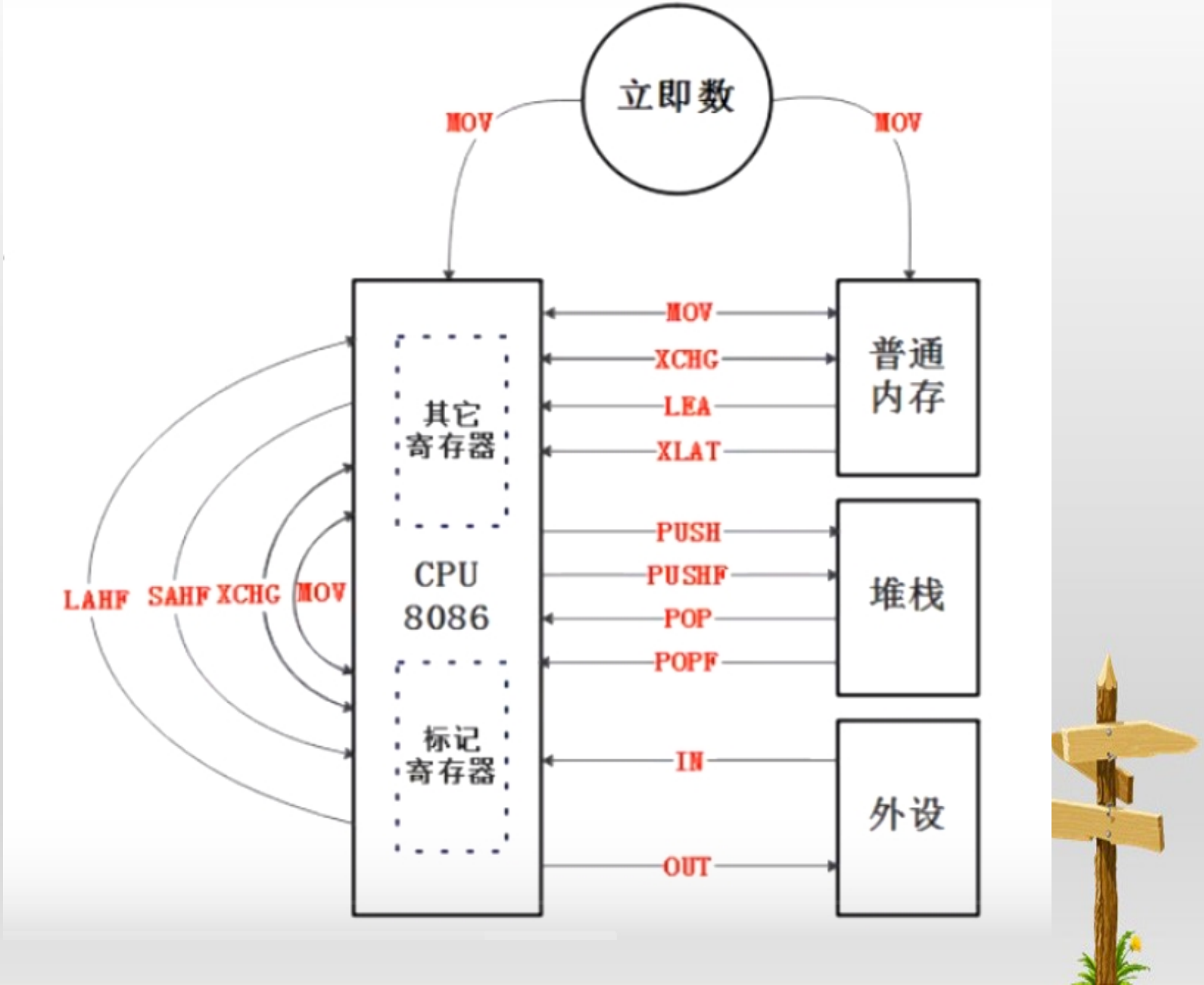
§3.3 8086的指令系统

3.3.1 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

- 数据传送指令见表3.3。
- 除SAHF和POPF 指令外，对标志位均没有影响。

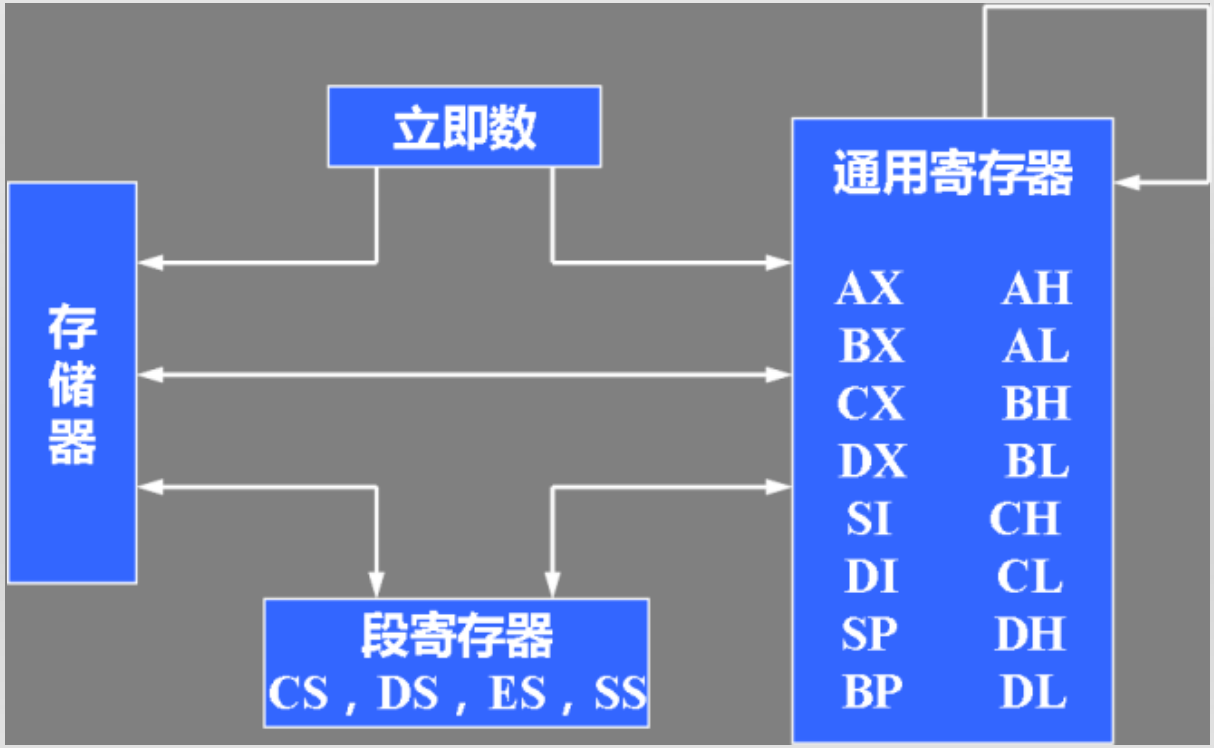

立即数
MOV
MOV
其它
寄存器
CPU
8086
标记
寄存器
LAHF
SAHF
XCHG
MOV
MOV
XCHG
LEA
XLAT
PUSH
PUSHF
POP
POPF
IN
OUT
普通
内存
堆栈
外设

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



- MOV 指令允许数据传送的途径如图3.12。但CS不能做目的操作数。
- 指令中至少要有一项明确说明传送的是字节还是字。

- 注意：
- 源、目的不能同时为段寄存器
- 立即数汉有直接送段寄存器
- 目的不能为是立即数和CS
- 目的、源不能同时为存储器寻址
- 不影响 标志位

1）MOV指令

例3.24

```
MOV   AL，‘B’
            ；AL←将字符B的ASCII码(42H)
```

例3.25

```
MOV   AX，DATA
MOV   DS，AX
```

- 由于DATA表示数据段的段址，是一个16位立即数，不能被直接送进DS，需要先送进另一个数据寄存器(如AX)，再传到DS中。

数据段

- 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   3 DUP(0)
ARRAY   DW   3100H，01A6H
STRING  DB   ‘GOOD’
DATA    ENDS                       ；数据段结束
```

数据段

- 数据段以段说明符SEGMENT开始，ENDS结束，DATA是数据段的段名。
- DB伪操作符用来定义字节变量。
- DW定义字变量，低字节在前，高字节在后。
- DUP复制操作符，前面的“3”说明在存储器中保留3个字节单元，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。
- 数据占用存储空间的情况如图3.13。➔
- 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

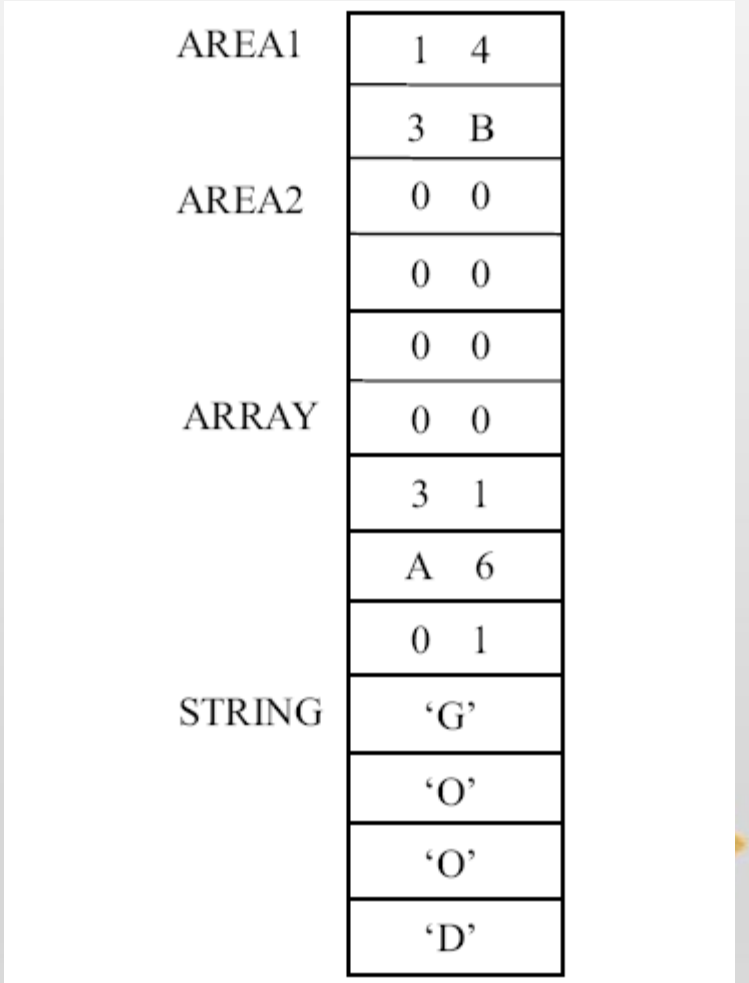


图 3.13　数据段占用存储空间的情况

1）MOV指令

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，符号地址ARRAY的偏移量0005H被送到了DX中。

例3.27

设AREA1和AREA2的值如图3.13，说明以下指令功能

MOV AL，AREA1；AL←AREA1中的内容14H

MOV AREA2，AL；0002H单元←14H

1）MOV指令

例3.28

```
MOV  AX，TABLE［BP］［DI］
```

；将地址为 16×SS+BP+DI+TABLE 的字单元中的内容送进AX。

1. 通用数据传送指令

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，使SP←SP-2，再把源操作数压入SP指示的位置上。

1. 通用数据传送指令

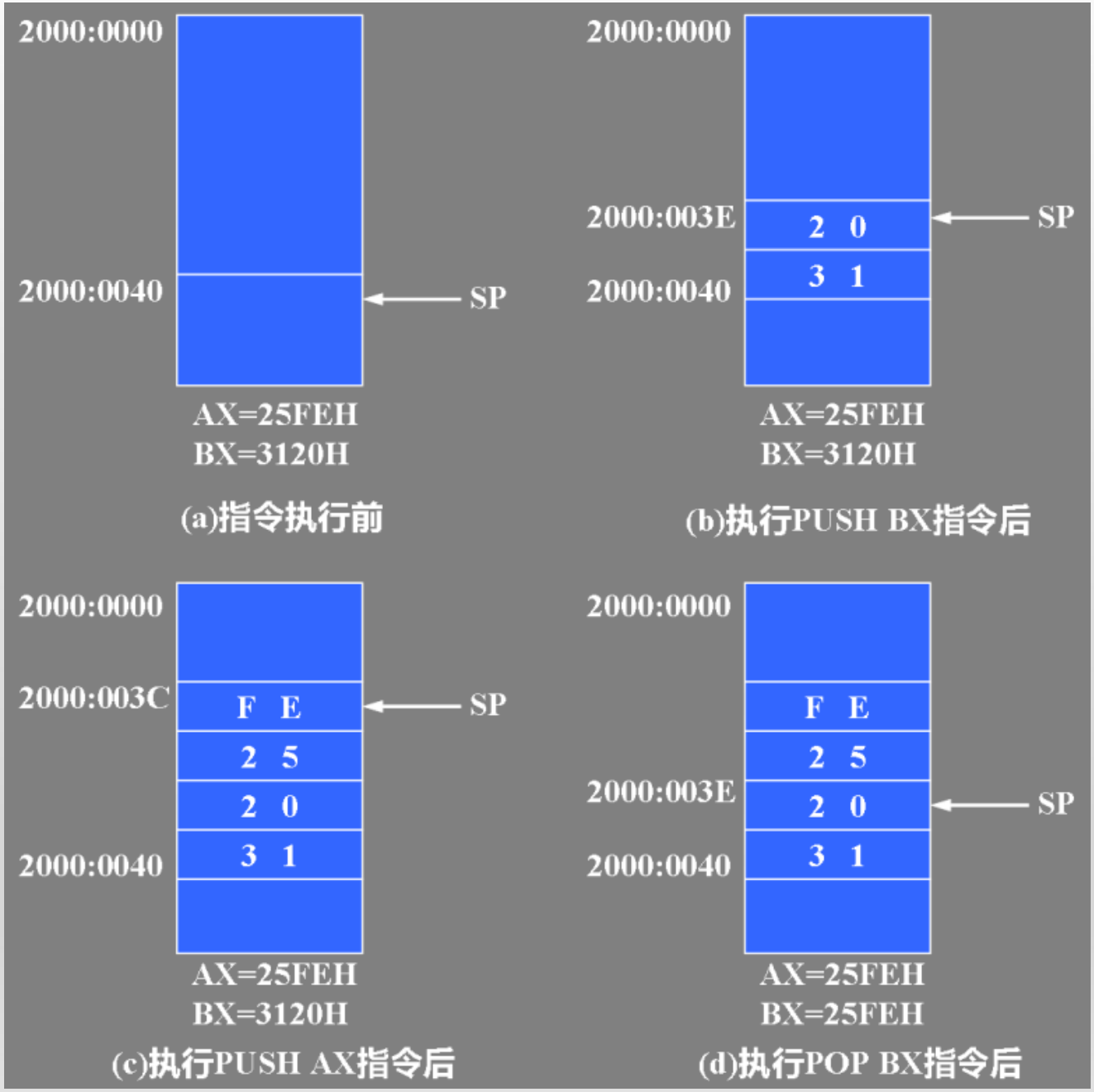
3) 出栈指令POP (Pop Word off Stack)

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，SP←SP+2，SP向高地址方向移动，指向新的栈顶。

- 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
- 目的不能是CS　　POP CS



例3.29

设SS=2000H

SP=40H

AX=25FEH

BX=3120H

依次执行指令：

PUSH BX

PUSH AX

POP BX

堆栈中的数据和SP的变化情况如图3.14所示。

- 例：PUSH AX
- PUSH BX
- PUSH CX
- …… ;期间用到AX、BX、CX、DX
- POP CX
- POP BX
- POP AX
- 保护现场

1. 通用数据传送指令

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=3000×10H+1800H +200H=31A00H其中数据＝1995H

指令执行后，AX=1995H，(31A00H)=2000H

- 交换两个存储单元（DA_BYTE1和DA_BYTE2）之间的数据。
- MOV AL,DA_BYTE1;
- XCHG AL,DA_BYTE2;
- XCHG AL,DA_BYTE1;MOV DA_BYTE1,AL;

1. 通用数据传送指令

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

指令格式：

XLAT 转换表　；“转换表”为表格首地址

或　XLAT　　；“转换表”可省略不写

指令功能：

将1个字节从一种代码转换成另一种代码

- 使用XLAT指令前，应建立一个表格，最多256个字节，且置：BX←转换表始址，AL← 表头地址到要找的某项间的位移量；
- 指令执行时，根据位移量从表中查到转换后的代码值，送入AL中。（AL）←（（BX）+（AL））

1. 通用数据传送指令

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

- 先用DB伪指令建1个表格，存放0~9的七段码值。
- 表格起始地址为TABLE，数字0~9的七段码存放在相对于TABLE的位移量为0~9的单元中。

程序如下：

```
TABEL  DB40H，79H，24H，30H，19H；七段码
       表格
            DB      12H，02H，78H，00H，18H
              …
MOV   AL，5                          ；AL←数字5的位移量
MOV   BX，OFFSET  TABLE；BX←表格首地址
XLAT  TABLE                          ；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。

2. 输入输出指令

例3.32

用IN指令从输入端口读取数据的例子。

```
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
```

2. 输入输出指令

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
```

- 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

2. 输入输出指令

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

2. 输入输出指令

例3.34 用OUT指令对输出端口进行操作的例子。

```
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

不影响 标志位

- 源操作数必须是存储单元，目的操作数是一个除段寄存器之外的16位寄存器。

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，不能直接用MOV指令来实现，必须使用下面指令：

LEA　BX，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，

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)

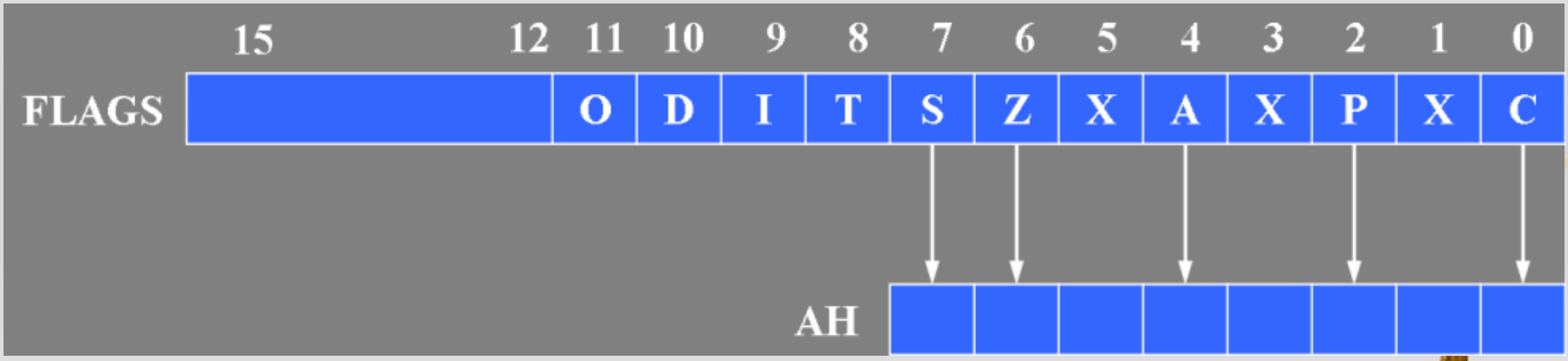
1) LAHF 标志送到AH指令

(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。

操作示意图如图3.15。



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： 把整个标志寄存器的内容推入堆栈，并使SP←SP-2

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←SP+2。

§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

3.3.2 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。

算术运算指令

- 上述4种类型数的表示方法见表3.5。

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

算术运算指令

- 系统提供加、减、乘、除四种基本运算指令，还有各种调整指令，见表3.6。

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

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←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←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，存储单元可以有表3.2中所示的24种表示方法。
- 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 列举上述两加法指令的实例，说明其用法。

```
ADD AL，18H              ；AL←AL+18H
ADC BL，CL               ；BL←BL+CL+CF
ADC AX，DX               ；AX←AX+DX+CF
ADD AL，COST［BX］
   ；将AL内容和物理地址=DS：(COST+BX)
   ；的存储字节相加，结果送到AL中
ADD COST［BX］，BL
   ；将BL与物理地址=DS：(COST+BX)的存储
   ；字节相加，结果留在该存储单元
```

- **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
```

相加过程的算式表示：

```
    0101  1110
+   0011  1100
-------------
    1001  1010
```

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，由两个数以及它们结果的符号决定，当两个加数符号相同，而结果的符号与之相反时，OF=1。

- 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。详见教材。

- 假定数据数组ARRAY包括从元素0到9共10个字节数。现要求累加元素3、元素5和元素7
- 程序为：
- MOV AL,0
- MOV SI,3
- ADD AL,ARRAY[SI]
- ADD AL,ARRAY[SI+2]
- ADD AL,ARRAY[SI+4]

1. 加法指令

3) INC 增量指令 (Increment)

指令格式： INC 目的

指令功能： 目的 ← 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。

1. 加法指令

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］
                  ；内存字节单元内容增1
INC   WORD  PTR［BX］
                  ；内存字单元内容增1
```

求内存数据段中M1为首和M2为首的两个20字节数之和，并将结果写入M2为首的区域中。

```
        LEA  SI，M1
        LEA  DI，M2
        MOV CX，20
        CLC
NEXT：  MOV AL，[SI]
        ADC [DI]，AL
        INC SI
        INC DI
        DEC CX
        JNZ NEXT
        HLT
```

使CF=0

按“字”运算

```
        LEA  SI，M1
        LEA  DI，M2
        MOV CX，10
        CLC
NEXT：  MOV AX，[SI]
        ADC [DI]，AX
        ADD SI，2
        ADD DI，2
        DEC CX
        JNZ NEXT
        HLT
```

1. 加法指令

4) AAA 加法的ASCII调整指令

(ASCII Adjust for Addition)

指令格式： AAA

指令功能： 用ADD或ADC指令对两个**非压缩BCD**数数或以ASCII码表示的十进制做加法后，结果在AL中，用此指令将AL中的结果进行调整。

另外，若AF=1，表示有进位，则进到AH中。

例3.44 非压缩十进制数的9可表示成0000 1001，5则为0000 0101，高4位均为0。设AH=0，若AL= BCD 9，BL= BCD 5，求两数之和。

运算过程：

```
ADD   AL,BL      ;        0000  1001 … 9
                 ;   +    0000  0101 … 5
                 ;  ------------------------------
AAA              ;        0000  1110 … 低 4 位>9
                 ;   +    0000  0110 … 加 6 调整
                 ;  ------------------------------
                 ;        0001  0100
                 ;   ∧    0000  1111 … 清高 4 位
                 ;  ------------------------------
                 ;        0000  0100 … AL=4
                 ;        CF=1,AF=1,AH=1
                 ;  结果为 AX=0104H,表示非压缩十进制数 14
```

例3.45　求ASCII码表示的数9(39H)与5(35H)之和。设AH=0，则运算过程：

```
MOV   AL,'9'      ;    AL =39H
MOV   BL,'5'      ;    BL =35H
ADD   AL,BL       ;        0011   1001 … '9'
                  ;    +   0011   0101 … '5'
                  ;    ------------------------------
AAA               ;        0110   1110 … 低 4 位>9
                  ;    +   0000   0110 … 加 6 调整
                  ;    ------------------------------
                  ;        0111   0100
                  ;    ∧   0000   1111 … 清高 4 位
                  ;    ------------------------------
                  ;        0000   0100 … AL=4
                  ;        CF=1,AF=1,AH=1
                  ;  结果为 AX=0104H,即非压缩十进制数 14
```

如想把AX中的结果表示成ASCII码，只要在AAA 指令后加一条指令：

OR AX,3030H

就可使AX中的结果变成了ASCI码3134H。

1. 加法指令

5) DAA 加法的十进制调整指令

(Decimal Adjust for Addition)

指令格式： DAA

指令功能： 对两个压缩BCD数相加后的结果（已在AL中）进行调整。

注意：要对AL中高半字节和低半字节分别进行调整

1. 加法指令

例3.46 若AL=BCD 38，BL=BCD 15，求两数之和.

运算过程：

```
ADD   AL,BL      ;        0011   1000 … 38
                 ;    +   0001   0101 … 15
                 ;   ------------------------------
DAA              ;        0100   1101 … 低 4 位>9
                 ;    +   0000   0110 … 加 6 调整
                 ;   ------------------------------
                 ;        0101   0011 … 结果为 AL=BCD 53,CF=0
```

1. 加法指令

例3.47 若AL=BCD 88，BL=BCD 49，求两数之和。

注意：要对AL中高半字节和低半字节分别进行调整。

运算过程：

```
ADD   AL,BL      ;        1000   1000 … 88
                 ;   +    0100   1001 … 49
                 ;  ------------------------------
DAA              ;        1101   0001 … AF=1
                 ;   +    0000   0110 … 加 6 调整
                 ;  ------------------------------
                 ;        1101   0111 … 调整后高半字节>9
                 ;   +    0110   0000 … 加 60H 调整
                 ;  ------------------------------
                 ;        0011   0111 … 结果为 AL=BCD 37,CF=1
```

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 ← 目的 – 源

例3.48

```
SUB   AX，BX          ；AX←AX–BX
SUB   DX，1850H       ；DX←DX–1850H
SUB   BL，［BX］
      ；BL中内容减去物理地址=DS：BX
      ；处的字节，结果存入BL
```

2. 减法指令

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 ← 目的 – 源 – CF

例3.49

SBB AL，CL；AL←AL – CL – CF

- SBB主要用于多字节减法中

2. 减法指令

3) DEC 减量指令 (Decrement)

指令格式： DEC 目的

指令功能： 目的 ← 目的 – 1

例3.50

```
DEC   BX                    ; BX←BX – 1
DEC   WORD PTR［BP］
                            ; 堆栈段中位于［BP］
                            ; 偏置处的字减1
```

2. 减法指令

4) NEG 取负指令 (Negate)

指令格式： NEG 目的

指令功能： 目的 ← 0 – 目的

例3.51

```
NEG   AX
      ；将AX中的数取负(改变数的符号位)

NEG   BYTE PTR［BX］
      ；对数据段中位于［BX］偏置处的字节取负
```

2. 减法指令

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                ；AL与80H作比较
CMP   BX，DATA1   ；BX与数据段中偏移量
                  ；为DATA1处的字比较
```

- 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。

2. 减法指令

6) AAS 减法的ASCII调整指令

(ASCII Adjust for Subtraction)

指令格式： AAS

指令功能： 在用SUB或SBB指令，对两个非压缩BCD数，或以ASCII码表示的十进制数相减后，对AL中所得结果进行调整，如有借位，则CF置1。

例3.54 设AL=BCD 3，CL=BCD 8，求两数之差。很显然，结果为BCD 5，但要向高位借位。

运算过程：

```
SUB   AL,CL      ;        0000  0011   BCD 3
                 ;   —    0000  1000   BCD 8
                 ;  ------------------------
AAS              ;        1111  1011   低 4 位>9
                 ;   —    0000  0110   减 6 调整
                 ;  ------------------------
                 ;        1111  0101
                 ;   ∧    0000  1111   高 4 位清 0
                 ;  ------------------------
                 ;        0000  0101   AL=5
                 ;   结果为 5,CF=1,表示有借位
```

2. 减法指令

7) DAS 减法的十进制调整指令

(Decimal Adjust for Subtraction)

指令格式： DAS

指令功能： 在用SUB或SBB指令，对两个**压缩BCD数**相减（结果已存在AL中）后，进行调整。

- 同样，它也要对AL中高半字节和低半字节分别进行调整。

2. 减法指令

例3.55 设AL=BCD 56，CL=BCD 98，求两数之差。

运算过程：

```
SUB   AL,CL      ;        0101   0110 … BCD 56
                 ;   —    1001   1000 … BCD 98
                 ;  ──────────────────────────
                 ;        1011   1110 … 低 4 位＞9,CF＝AF＝1
DAS              ;   —    0000   0110 … 减 6 调整
                 ;  ──────────────────────────
                 ;        1011   1000 … 高 4 位＞9
                 ;   —    0110   0000 … 减 60H 调整
                 ;  ──────────────────────────
                 ;        0101   1000 … BCD 58
                 ;   结果为 AL＝BCD 58,CF＝1,表示有借位
```

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 ← AL * 源
- 若源操作数是1个字， 则 (DX，AX) ← AX * 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。

3. 乘法指令

例3.56

```
MUL   DL                    ; AX←AL*DL
MUL   CX                    ; (DX，AX)←AX*CX
MUL   BYTE［SI］   ; AX←AL*(内存中某字节)，
                            ; BYTE说明字节乘法
MUL   WORD［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为零，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。

3. 乘法指令

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。

- 存放结果的方式与MUL相同，最后给乘积赋予正确的符号。
- 指令执行后，如果乘积的高半部分不是全0或全1，则置CF=1，OF=1。 若结果高半部分为全0或全1，则使CF=0，OF=0。这样来决定是否需要保存积的高半部分。

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
```

3) AAM 乘法的ASCII调整指令

(ASCII Adjust for Multiply)

指令格式： AAM

指令功能：对存于AL的两个非压缩BCD数相乘的积进行调整，结果在AX中，高位放AH，低位在AL。

- 两个ASCII码数相乘前，应先屏蔽掉每个数字的高半字节。
- 调整过程： 把AL内容除以10，商放在AH中，余数在AL中。即

  AH ← AL/10所得的商

  AL ← AL/10所得的余数

  指令执行后，将影响ZF、SF和PF。

3. 乘法指令

例3.60 求两个非压缩十进制数09和06之乘积。

可用如下指令实现：

```
MOV     AL，09H     ；置初值
MOV     BL，06H
MUL     BL          ；AL←09与06之乘积36H
AAM                 ；调整得AH=05H(十位)，
                    ；AL=04H(个位)
```

最后，可在AX中得到正确结果AX=0504H，即BCD数54。

3. 乘法指令

- 如果AL和BL中分别存放9和6的ASCII码，则求两数之积时要用以下指令实现：

```
AND     AL，0FH          ；屏蔽高半字节
AND     BL，0FH
MUL     BL               ；相乘
AAM                      ；调整
```

- 如要将结果转换成ASCII码，可再用指令ORAX，3030H来实现，使AX=3534H。
- 8086/8088指令系统中，不允许采用压缩十进制数做乘法，乘法调整指令仅此一条。

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 ← AX/源(字节)的商

  AH ← AX/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。

4. 除法指令

1) DIV 无符号数除法指令

- 若源操作数为字，被除数必须放在DX和AX中，并且有：

  AX ← (DX，AX)/源(字)的商

  DX ← (DX，AX)/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。

4. 除法指令

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。

4. 除法指令

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。

4. 除法指令

- 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，再执行除法指令。

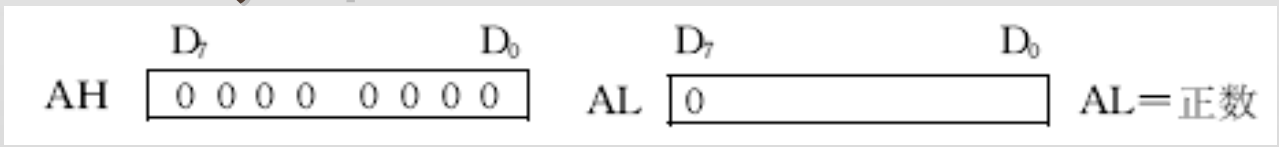
4. 除法指令

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

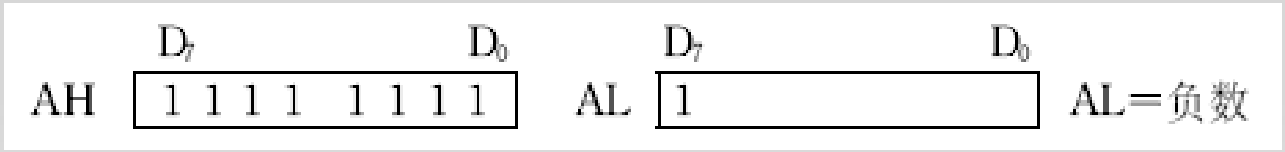
指令格式： CBW

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. 除法指令

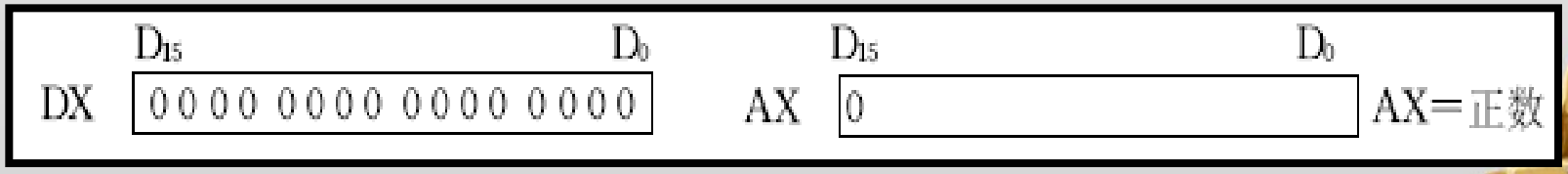
4) CWD 把字转换成双字指令

(Convert Word to Double Word)

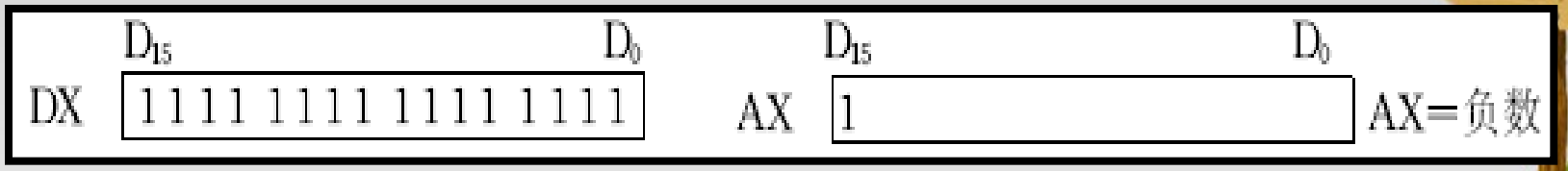
指令格式： CWD

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←0000H，即



- 若AX中的D15=1，则DX←FFFFH，即



4. 除法指令

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)
```

4. 除法指令

5) AAD 除法的ASCII调整指令

(ASCII Adjust for Division)

指令格式：AAD

指令功能：在做除法前把BCD码转换成二进制数。

- 前面介绍的调整指令，都是在用加法、减法和乘法指令后，紧跟着用一条AAA、AAS或AAM指令，对运算结果进行调整。
- 而除法的ASCII调整指令不同，它是在除法之前进行的。

4. 除法指令

5) AAD 除法的ASCII调整指令

- 在把AX中的两位非压缩BCD数除以一个非压缩BCD数之前，先用AAD指令，把AX中的被除数调整成二进制数，并存入AL，然后才能用DIV指令进行运算。调整的过程为：

  AL←AH×10+AL

  AH←00

- 本指令根据AL寄存器的结果影响SF、ZF和PF。

4. 除法指令

例3.63　设AX中存有两个非压缩BCD数0307H，即十进制数37，BL中存有一个非压缩BCD数05H，若要完成AX/BL的运算，可用以下指令：

```
AAD
DIV   BL
```

- 第1条指令先将AX中的两个BCD数转换成二进制数，03×10+7=37=25H，并将25H→AL，显然经调整后的被除数25H才真正代表37，再用DIV指令做除法，可得正确的结果：

AL=7 (商)

AH=2 (余数)