
上海交通大学
网络学院

微机原理与应用

The Principle & Application of Microcomputer

王春香　副教授

wangcx@sjtu.edu.cn

第三章 8086/8088微处理器及其系统

主要内容

3.1 8086/8088微处理器

3.2 8086/8088系统的最小/最大工作方式

3.3 8086/8088的存储器

3.4 8086/8088的指令系统

第三章 8086/8088微处理器及其系统

- 透彻理解与熟练掌握8086/8088内部组成结构、寄存器结构与总线周期等。
- 深入理解存储器的分段设计。
- 正确理解与熟练掌握物理地址和逻辑地址关系。
- 理解堆栈及其操作。
- 理解“段加偏移”寻址机制。
- 掌握寻址方式。
- 掌握6大类指令系统的基本用法。

3.4.1 8086/8088 指令系统特点

8086与8088指令系统由8位8080／8085指令系统扩展而来的，同时又能在其后续的80x86系列的CPU上正确运行。
其主要特点是：

① 可变长指令：指令格式比较复杂。

② 寻址方式多样灵活，处理数据能力比较强（字节/字、有符号/无符号二进制数据、压缩型/非压缩型十进制数据）。

③ 有重复指令、乘除运算指令。扩充了条件转移、移位/循环指令。

④ 增设了加强软件中断功能和支持多处理器系统的相关指令。

3.4.2 8086/8088 的指令格式

指令由两部分构成：

操作码	操作数或操作数地址

操作码(OP-Code)字段：表示计算机所要执行的操作类型，由一组二进制代码表示。在汇编语言中用助记符代表。

操作数(Oprand)字段：指出指令执行的操作所需的操作数，可以是操作数本身，或是操作数地址，或是操作数地址计算方法。

3.4.2 8086/8088 的指令格式

操作码	操作数或操作数地址

无操作数：控制类指令

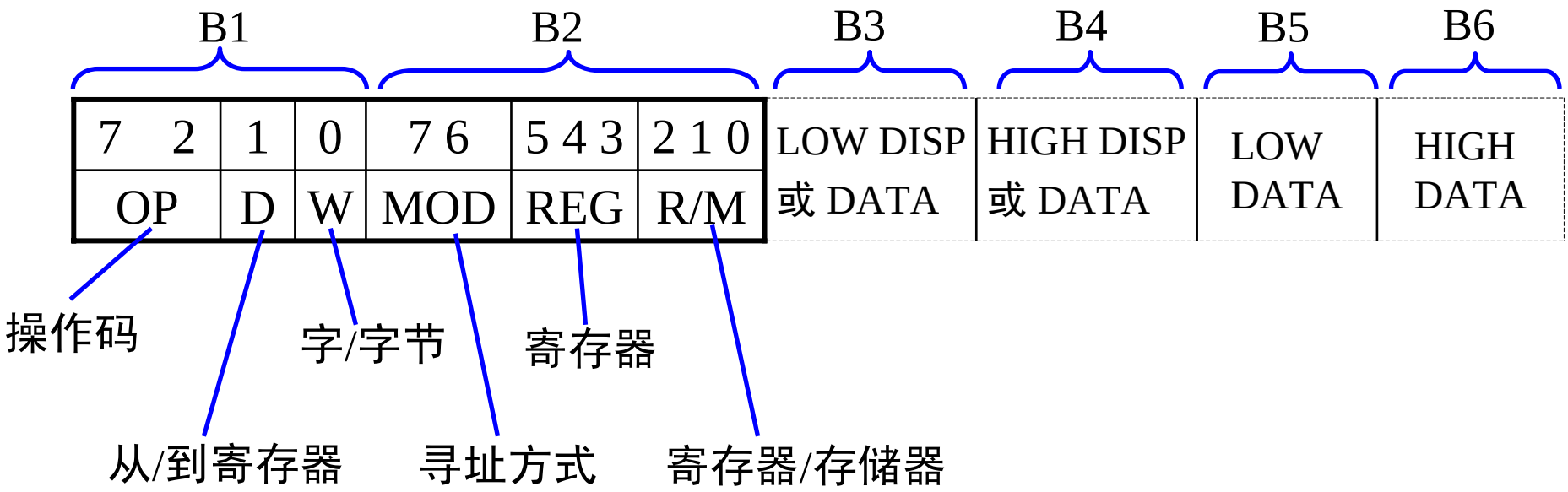
单操作数：只给出一个操作数地址，该操作数可在寄存器或存储器中，或指令中直接给出立即数。

双操作数：源操作数(source)，目的操作数(destination)一个操作数在寄存器中，另一个在寄存器或存储器中，或指令中直接给出立即数。不允许两个都在存储器中。

3.4.2 8086/8088 的指令格式

可变长指令，指令长度为1～6个字节。
其中B1和B2为基本字节，B3~B6根据不同指令作相应的安排。

指令格式：



3.4.2 8086/8088 的指令格式

① B1 字节

- **OP** — 指令操作码。
- **D** — 表示方向。

  D=1，REG字段所确定的寄存器为目的；
  D=0，则该寄存器为源。

- **W** — 表示字节或字处理方式。

  W=1，表示字；
  W=0，表示字节。

3.4.2 8086/8088 的指令格式

② B2 字节

- **MOD** — 给定指令的寻址方式。(2位)

MOD≠11：存储器方式，有一个操作数位于存储器中；

MOD＝00，没有位移量；
MOD＝01，只有低8位位移量，需将符号扩展8位，形成16位；
MOD＝10，有16位位移量。

MOD＝11：寄存器方式，两个操作数均在寄存器中，
一个由REG字段确定，
一个由R/M字段确定。

3.4.2 8086/8088 的指令格式

② B2 字节（续）

- **REG — 用来确定某一个操作数所在的寄存器编码。**
- **R/M — 受MOD制约。**

MOD＝11，某操作数所在的寄存器编码；

MOD ≠11，某操作数所在的存储单元有效地址EA。共24种计算方法。

3.4.2 8086/8088 的指令格式

	MOD=11			MOD≠11时存储器有效地址计算方法			
	R/M	W=0	W=1	R/M	MOD=00	MOD=01	MOD=10
	000	AL	AX	000	BX+SI	BX+SI+D8	BX+SI+D16
	001	CL	CX	001	BX+DI	BX+DI+D8	BX+DI+D16
	010	DL	DX	010	BP+SI	BP+SI+D8	BP+SI+D16
	011	BL	BX	011	BP+DI	BP+DI+D8	BP+DI+D16
	100	AH	SP	100	SI	SI+D8	SI+D16
	101	CH	BP	101	DI	DI+D8	DI+D16
	110	DH	SI	110	直接地址	BP+D8	BP+D16
	111	BH	DI	111	BX	BX+D8	BX+D16
D=1	目的操作数			源操作数			
D=0	源操作数			目的操作数			

注：D8为8位偏移量；D16为16位偏移量。
涉及BP寄存器寻址方式时，SS为默认的断寄存器；其它情况DS为默认的段寄存器。

3.4.2 8086/8088 的指令格式

无操作数指令

暂停指令 HLT

OP
1 1 1 1　0 1 0 0

F4H

单操作数指令

减1指令 DEC BX

OP	REG
0 1 0 0 1	0 1 1

4BH

双操作数指令

加法指令 ADD AX, BX

OP	d	w	mod	reg	r/M
000000	1	1	11	000	011

03C3H

3.4.3 寻址方式

1. 固定寻址
2. 立即数寻址
3. 寄存器寻址
4. 存储器寻址
   - 直接寻址
   - 间接寻址
     - 基址寻址
     - 变址寻址
     - 变址加变址寻址
5. 其它寻址方式
   - 串操作指令寻址
   - I/O端口寻址
   - 转移类指令寻址

3.4.3 寻址方式

1. 固定寻址

有些单字节指令，其操作是规定CPU对某个固定的寄存器进行的。

如：加法的ASCII调整指令AAA，规定被调整的数总位于AL中。

该指令用来调整AL中的结果，指令编码为：

OP	
0 0 1 1 0 1 1 1	37H

3.4.3 寻址方式

2. 立即数寻址

操作数就在指令中，当执行指令时，CPU直接从指令队列中取得立即数，而不必执行总线周期。

- 立即数可以是8位，或16位；
- 只能是整数类型的源操作数；
- 主要用来给寄存器赋初值；
- 指令执行速度快。

如：加法指令　MOV AX, 1680H

表示将1680H送AX，AH中为16H，AL中为80H；即高地址对应高字节，低地址对应低字节。

3.4.3 寻址方式

3. 寄存器寻址

操作数在CPU的寄存器中，指令中给出寄存器名。
源操作数和目的操作数均可采用寄存器寻址方式。

- 寻址的指令长度短；
- 操作数就在CPU内部进行，不需要使用总线周期；
- 指令执行速度快。

如： 加1指令　INC reg

表示将寄存器内容加1，指令编码：

7 … 3	2 … 0
OP	REG
0 1 0 0 0	rrr

16位：AX，CX，DX，BX，SP，BP，SI，DI
8位：AH，AL，BH，BL，CH，CL，DH，DL
000　001　010　011　100　101　110　111

3.4.3 寻址方式

4. 存储器寻址

寻找存储器操作数，必须经总线控制逻辑电路进行存取。

当EU单元需要读/写位于存储器的操作数时：

① 根据寻址方式(指令中B2字节)，由**EU计算**出操作数地址的偏移量，即有效地址**EA**；

② 将**EA送至BIU**单元，同时请求BIU执行一个总线周期；

③ **BIU**将某个段寄存器的内容左移4位，加上EU送来的EA，形成20位的实际地址，即**物理地址PA** ；

④ **执行总线周期**，读/写指令所需的操作数。

计算EA的通式为：

$$EA = \text{基址值}\begin{pmatrix}BX\\BP\end{pmatrix} + \text{变址值}\begin{pmatrix}SI\\DI\end{pmatrix} + \text{位移量D}\begin{pmatrix}0\\8\\16\end{pmatrix}$$

3.4.3 寻址方式

4. 存储器寻址：直接寻址

- 最简单、直观。
- 指令中直接以位移量形式，给出操作数的有效地址EA，即EA＝DISP
- 执行速度快，主要用于存取位于存储器中的简单变量。

编码格式为：4个字节

如：MOV AX, [1680H]

表示将1680H和1681H两单元的取入AX中。

3.4.3 寻址方式

4. 存储器寻址：间接寻址

- 指寄存器寻址方式，操作数一定在存储器中；
- 存储单元的EA由寄存器指出：基址寄存器BX，基址指针寄存器BP，变址寄存器SI和DI；
- 书写时，寄存器带方括号；
- 根据所采用的寄存器不同，分为三种：

基址寻址：BX或BP＋位移量

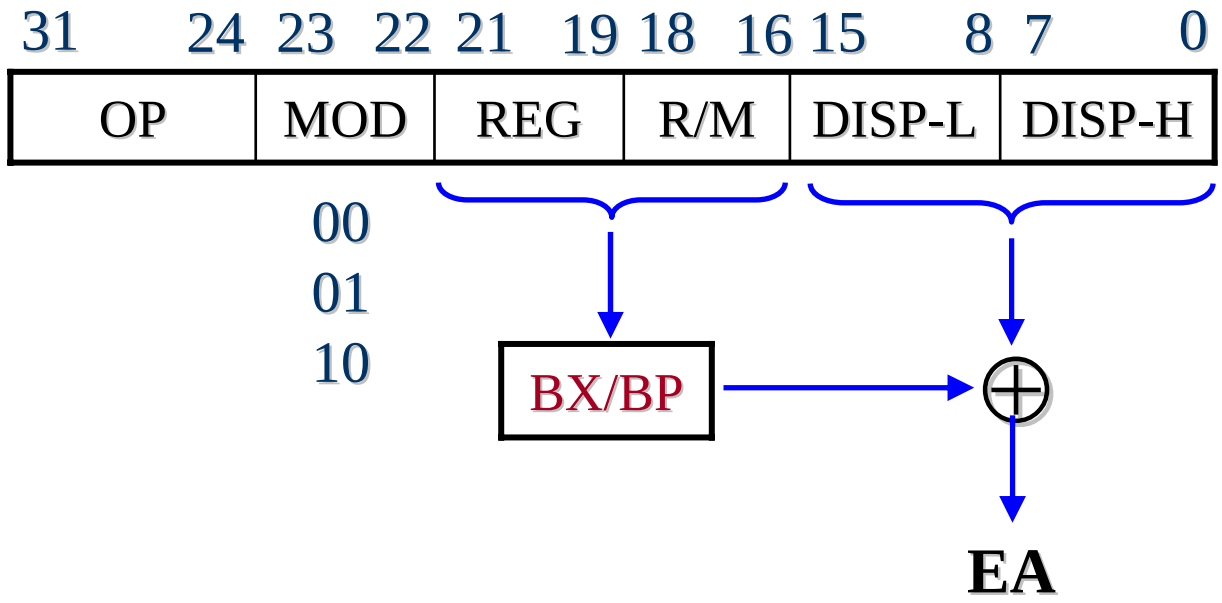
变址寻址：SI或DI＋位移量

基址加变址：BX或BP＋SI或DI＋位移量

3.4.3 寻址方式

4. 存储器寻址：间接寻址1

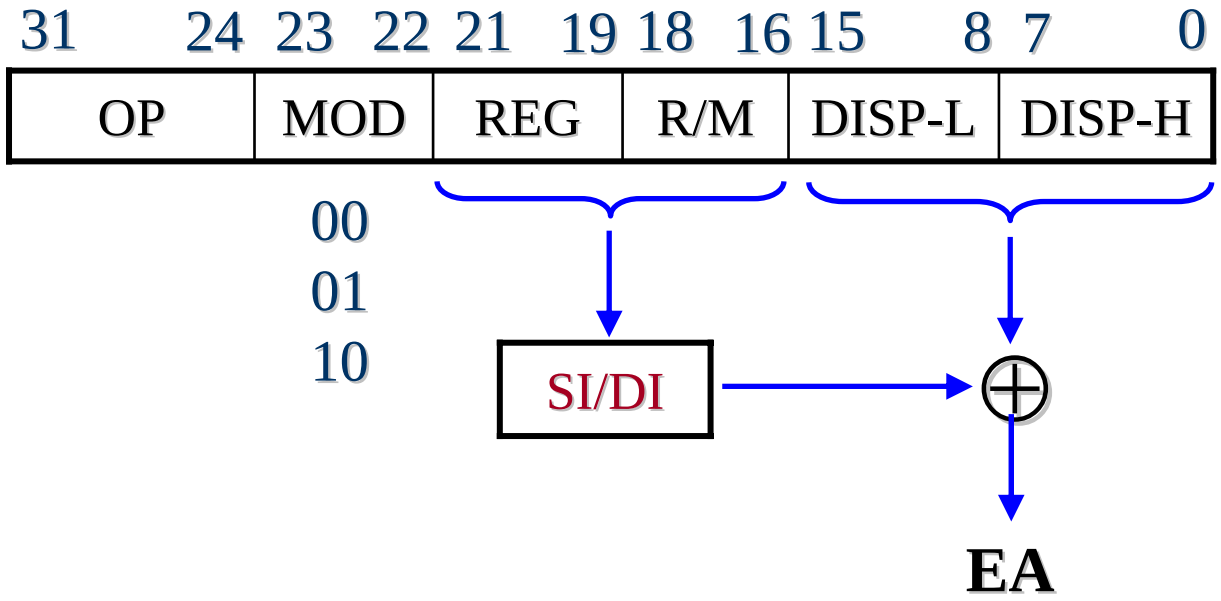
① 基址寻址



3.4.3 寻址方式

4. 存储器寻址：间接寻址2

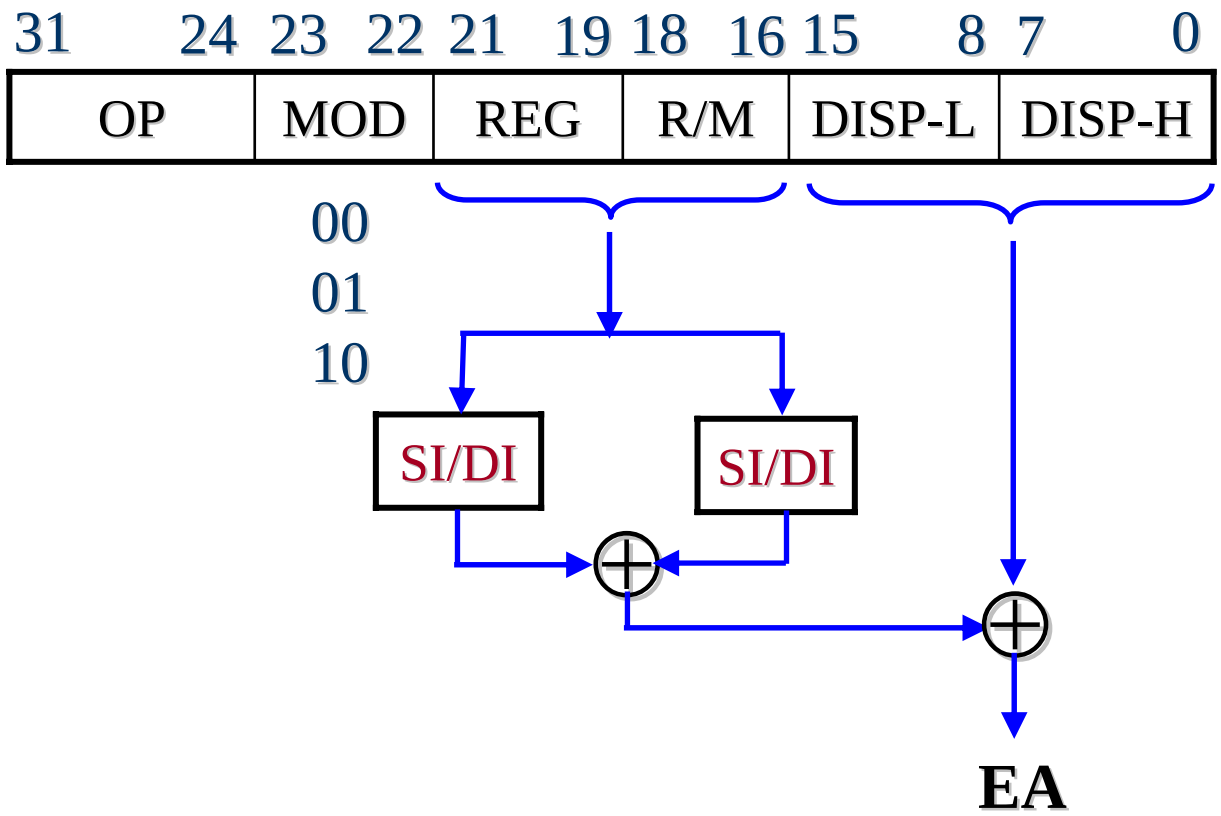
② 变址寻址



3.4.3 寻址方式

4. 存储器寻址：间接寻址3

③ 变址加变址寻址



3.4.3 寻址方式

4. 存储器寻址实例1

例：设DS=1200H，BX=05A6H，SS=5000H，BP=40A0H，SI=2000H，DI=3000H，位移量DISP=1618H。

试判断下列指令的寻址方式，并求出在各种寻址方式下的EA和PA，并说明指令执行的结果。

① **MOV AX，[0618H]**　　**直接寻址**

EA = DISP = 0618H

PA = DS左移4位 + EA
=12000H+0618H
=12618H

执行结果：将数据段12618H和12619H两单元的内容取出送AX。

3.4.3 寻址方式

4. 存储器寻址实例2

② MOV AX，[BX]　间接寻址：基址寻址

EA = BX = 05A6H

PA = DS左移4位＋EA
=12000H+05A6H
=125A6H

执行结果：将数据段125A6H和125A7H两单元的内容取出送AX。

3.4.3 寻址方式

4. 存储器寻址实例3

③ MOV AX，[BP]　间接寻址：基址寻址

EA = BP = 40A0H

PA = SS左移4位 + EA
= 50000H + 40A0H
= 540A0H

执行结果：将堆栈段540A0H和540A1H两单元的内容取出送AX。

3.4.3 寻址方式

4. 存储器寻址实例4

④ MOV AX，[DI]　间接寻址：变址寻址

EA=DI=3000H

PA= DS左移4位+EA
= 12000H+3000H
= 15000H

执行结果：将数据段15000H和15001H两单元的内容取出送AX。

3.4.3 寻址方式

4. 存储器寻址实例5

⑤ MOV AX，[BX+DI]　间接寻址：基址加变址寻址

EA=BX+DI
　　=05A6H+3000H=35A6H

PA=DS左移4位+EA
　　=12000H+35A6H=155A6H

执行结果：将数据段155A6H和154A7H两单元的内容取出送AX。

3.4.3 寻址方式

4. 存储器寻址实例6

⑥ MOV AX，[BP+SI+DISP]　间接寻址：基址加变址寻址

EA=BP＋SI+DISP
=40A0H+2000H+1618H=76B8H

PA=SS左移4位+EA
=50000H+78B8H=576B8H

执行结果：将堆栈段576B8H和576B9H两单元的内容取出送AX。

3.4.3 寻址方式

5. 其它寻址方式1

5.1 串操作指令寻址方式

- ✓ 源串操作数第1个字节/字有效地址存放在源变址寄存器SI中。
- ✓ 目标串操作数第1个字节/字有效地址存放在目标变址寄存器DI中。
- ✓ 重复串操作时，自动修改SI和DI的内容，指向后面的字节/字。
- ✓ 指令中，不必给出SI或DI的编码，故串操作指令是隐含寻址方式。

3.4.3 寻址方式

5. 其它寻址方式2

5.2 I/O端口寻址方式

① 直接端口寻址：

以8位立即数方式在指令中直接给出。

端口号范围 0～255

例如：IN AL，n

② 间接端口寻址：

通过DX间接寻址，16位端口地址放在DX中。

端口号范围 0～65535

例如：OUT DX，AL

3.4.3 寻址方式

5. 其它寻址方式3

5.3 转移类指令的寻址方式

8086/8088系统中，存储器采用分段结构，转移类指令有段内转移和段间转移。

条件转移指令：

只允许实现段内转移，且段内短转移，即转移地址范围为-128～+127字节，由指令直接给出8位地址偏移量。

无条件转移指令和调用指令：

段内短转移，段内直接转移，段内间接转移，段间直接转移，段间间接转移

3.4.4 指令的分类

8086指令系统中，包含133条基本指令，与寻址方式结合，再加上不同的数据形式，可构成上千种指令。

按功能指令可分为6类：

① 数据传送类
② 算术运算类
③ 逻辑运算与移位类
④ 串操作类
⑤ 处理器控制类

1. 数据传送类

可完成寄存器与寄存器之间、寄存器与存储器之间，寄存器与I/O端口之间的字节或字传送。

特点：除SAHF和POPF不影响标志寄存器内容。

共14条，分为4小类：

- 通用数据传送（5条）
- 目标地址传送（3条）
- 标志位传送（4条）
- I/O数据传送（2条）

1. 数据传送类

指令类型	指令功能	指令书写格式
通用数据传送	字节或字传送 字压入堆栈 字弹出堆栈 字节或字交换 字节翻译	MOV d,s PUSH s POP d XCHG d,s XLAT
目标地址传送	装入有效地址 装入DS寄存器 装入ES寄存器	LEA d,s LDS d,s LES d,s
标志位传送	将FR低字节装入AH寄存器 将AH内容装入FR低字节 将FR内容压入堆栈 从堆栈弹出FR内容	LAHF SAHF PUSHF POPF
I/O数据传送	输入字节或字 输出字节或字	IN 累加器,端口 OUT 端口,累加器

1. 数据传送类

① 通用数据传送指令（5条）

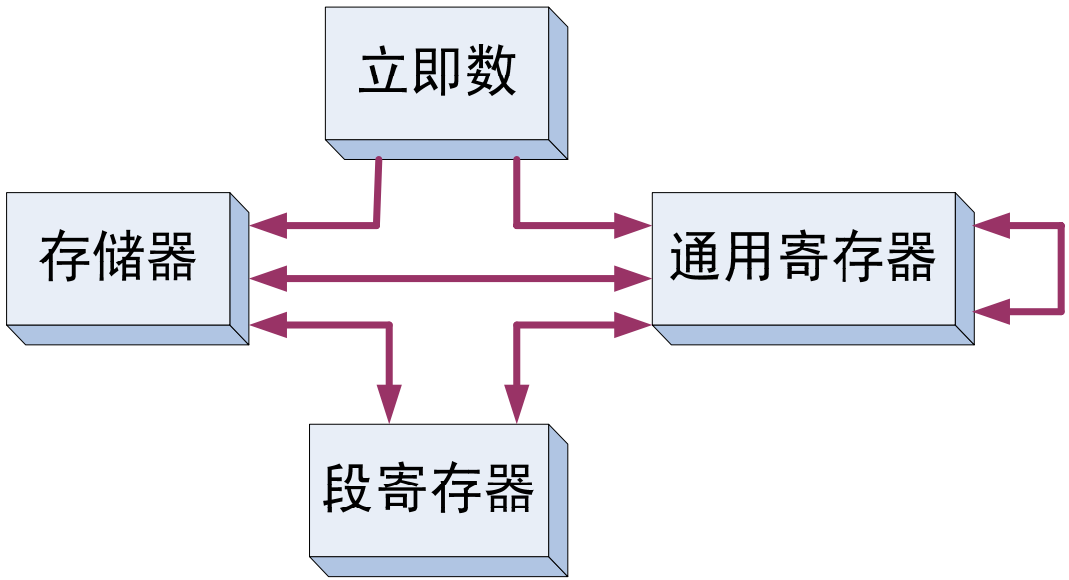
- ☆ 字节或字传送：MOV 目的，源
- ☆ 字压入堆栈：PUSH 源
- ☆ 字弹出堆栈：POP 目的
- ☆ 字节或字交换：XCHG 目的，源
- ☆ 字节翻译：XLAT

1. 数据传送类：通用数据传送

- **MOV d, s** ；将源操作数s指定的内容送到目的d

源操作数： 8/16位寄存器、存储器中的某个字节/字，或者8/16位立即数；

目标操作数： 8/16位寄存器、存储器中的某个字节/字，不能为立即数；



1. 数据传送类：通用数据传送

MOV 指令需注意问题：

- ✓ MOV指令可传送8位数据，也可传送16位数据。
- ✓ MOV 指令中的两操作数s和d，必用1个寄存器，不允许同为存储单元。
- ✓ 不能用CS和IP做目的操作数。
- ✓ 不允许段寄存器之间直接数据传送。
- ✓ 不允许立即数做目的操作数。
- ✓ 不能向段寄存器送立即数。

1. 数据传送类：通用数据传送

例1： 8/16位立即数送累加器AL和AX的指令。

```
MOV AX,0        ；AX清0，字传送
MOV AL,12H      ；AL ← 12H，字节传送
```

例2： 两个存储单元(SI)和(DI)之间传送数据。

```
MOV AL, [SI]
MOV [DI],AL
```

1. 数据传送类：通用数据传送

例3： CPU内部寄存器之间(除CS, IP外)实现数据的任意传送。

MOV AH,AL ；AH ← AL，字节传送

MOV DS,AX ；DS ← AX，字传送

MOV SI,BP ；SI ← BP，字传送

1. 数据传送类：通用数据传送

例4： CPU内部通用寄存器与存储器之间实现的数据传送。

```
MOV AL,BUFFER
        ；AL ← 内存单元BUFFER中字节内容

MOV AX,[SI]
        ；AX ← SI间接寻址指示的内存两单元字内容

MOV [DI],CX
        ；[DI] ← CX的内容
```

1. 数据传送类：通用数据传送

例5： 将数据段首地址DATA填入DS中。

```
MOV AX, DATA
        ; AX ← 数据段首地址DATA

MOV DS, AX
        ; DS ← AX
```

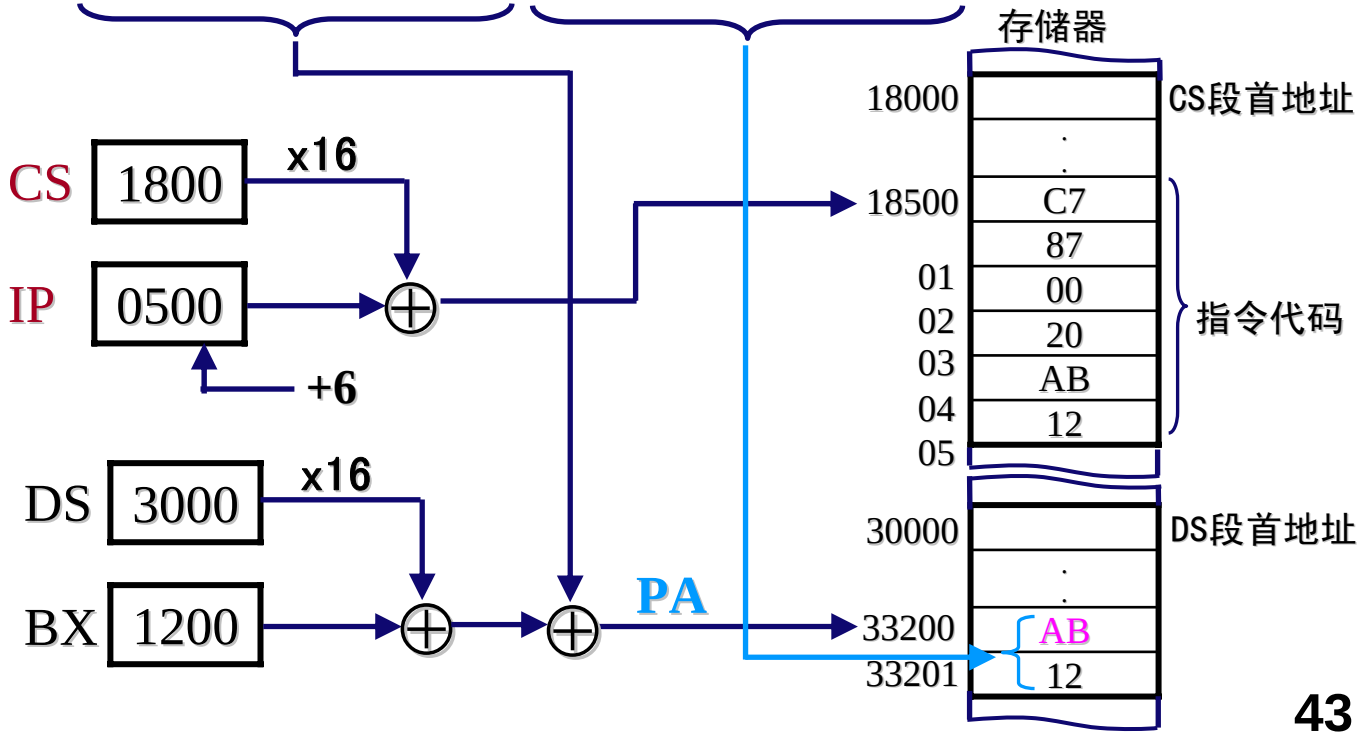
1. 数据传送类：通用数据传送

MOV [BX+2000H],12ABH

其中：BX=1200H, CS=1800H, IP=0500H, DS=3000H

指令编码格式与操作过程如下：

7 0	7 0	7 0	7 0	7 0	7 0
OP	W	MOD	OP	R/M	DISP-L	DISP-H	DATA-L	DATA-H
1100 011	1	10	000	111	0000 0000	0010 0000	1010 1011	0001 0010



1. 数据传送类：通用数据传送

- **PUSH s**；将源操作数(16位)压入堆栈
- **POP d** ；将堆栈中当前栈顶两相邻单元数据字弹出到d

特点：

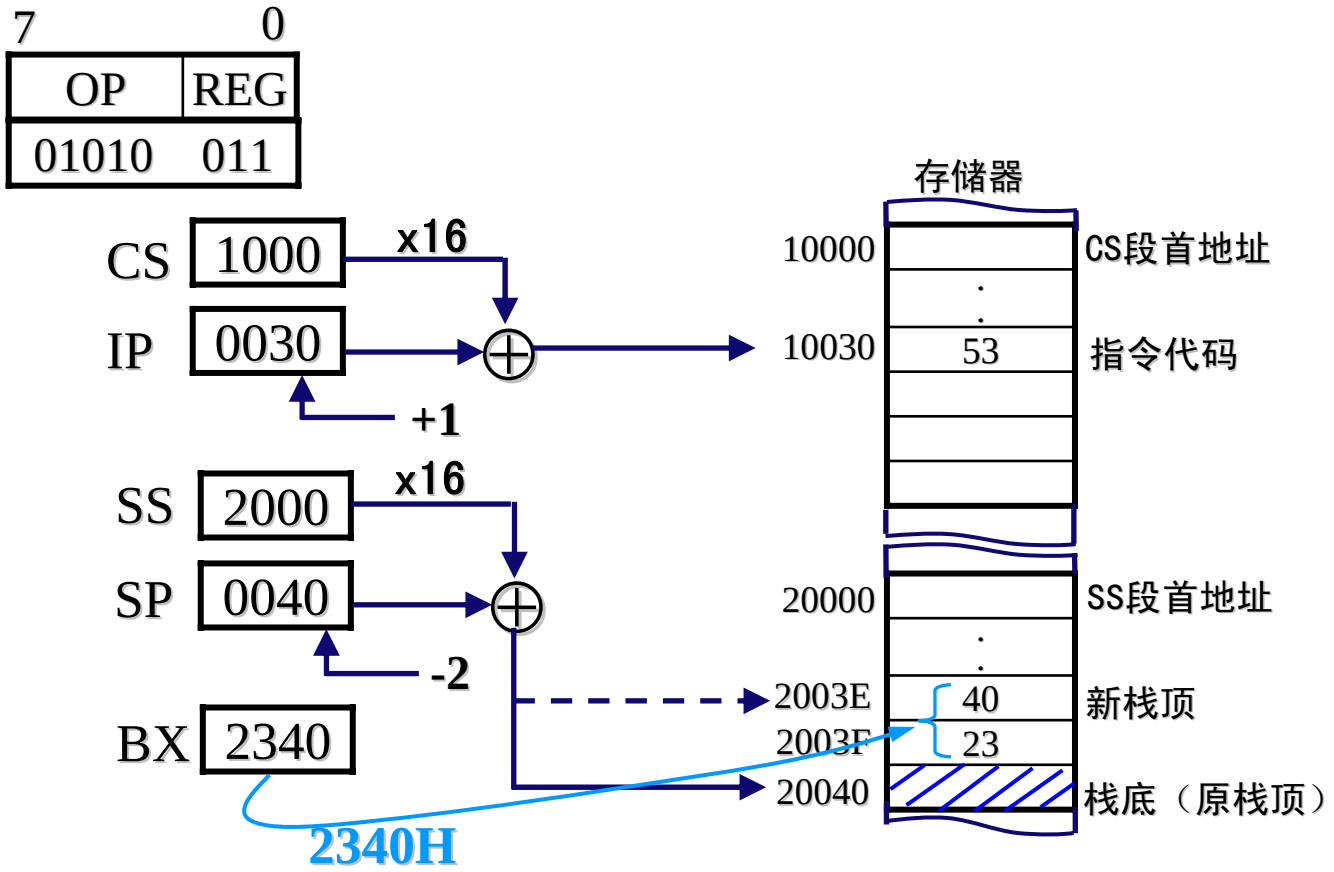
- s和d可以是16位寄存器或存储器两相邻单元；
- 堆栈按字操作；
- 每执行一条入栈指令，堆栈地址指针SP减2，入栈的数据位于栈顶；
- 高位字节先入栈，放在较高地址单元，低位字节后入栈，放在较低地址单元；“先进先出原则”
- 执行弹出指令时，过程相反，栈顶指针的值加2；
- CS段寄存器值可以入栈，但不能反过来弹出一个字到CS。

1. 数据传送类：通用数据传送

PUSH BX

其中：CS=1000H, IP=0030H, SS=2000H, SP=0040H, BX=2340H

指令编码格式与操作过程

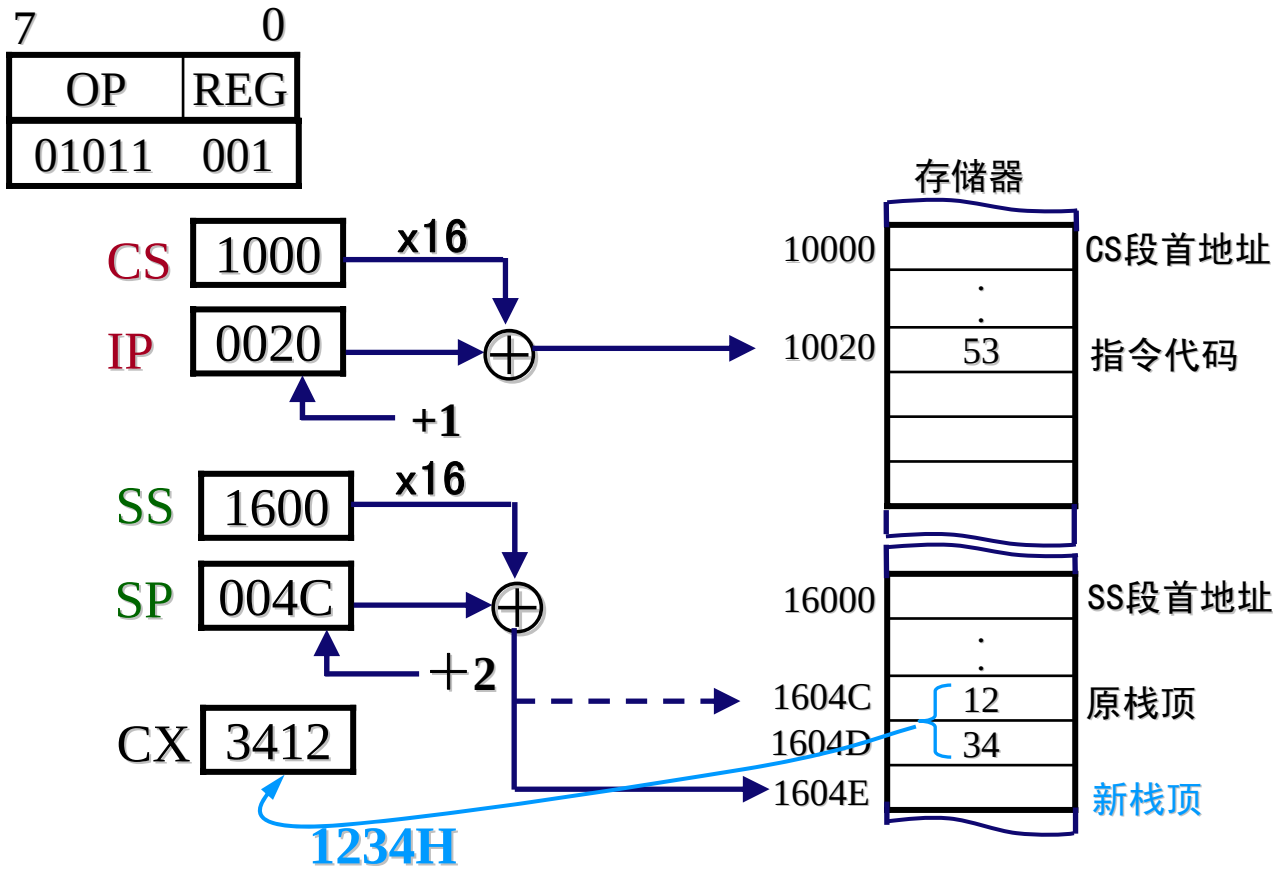


1. 数据传送类：通用数据传送

POP CX

其中：CS=1000H, IP=0020H, SS=1600H, SP=004CH

指令编码格式与操作过程



1. 数据传送类：通用数据传送

- XCHG d, s ；将源操作数和目的操作数(字或字节)
  ；相互对应交换位置。

特点：

- 可以在通用寄存器与累加器之间，通用寄存器之间交换；
- 通用寄存器与存储器之间进行；
- 两个存储单元之间不能交换；
- 不能使用立即数；
- 段寄存器与IP不能作为一个源或目的操作数。

1. 数据传送类：通用数据传送

例如： XCHG AX, [SI+0400H]

其中：CS=1000H, IP=0064H, DS=2000H, SI=3000H,
AX=1234H

物理地址：

PA=DSx16+SI+0400H
=20000H+3000H+0400H=23400H；

将AX内容1234H，与23400H、23401H两单元内容ABCDH相互交换位置。

执行结果： AX＝ABCDH

(23400H)=34H， (23401H)=12H

1. 数据传送类：通用数据传送

- **XLAT** ；通过查表来完成代码转换，
  ；用于实现字节翻译功能的指令。

将寄存器AL中设定的一个字节数值变换为内存一段连续表格中的另一个相应代码，以实现编码制的转换。

对于一些无规律的代码转换比较方便。

使用指令前：BX寄存器指向表的首地址

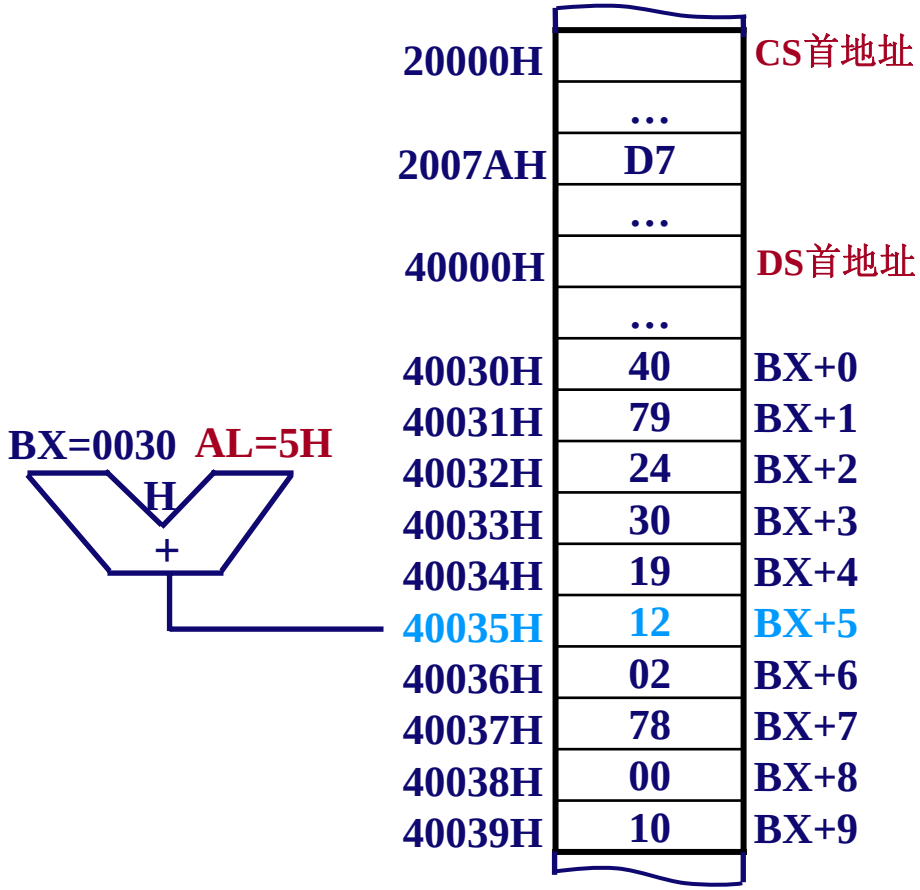
AL中存放待查的码，即某一项与表首址的距离。

1. 数据传送类：通用数据传送

例如：十进制数0～9的七段显示码表，被定位在当前数据段中，其起始地址的偏移地址值为0030H，将AL中待转换的十进制数5转换成对应的七段码。CS=2000H，IP=007AH，DS=4000H。

操作步骤：

- 建立代码转换表：最大容量256字节，将该表定位到内存中某个逻辑段的一片连续地址中，并将表首地址的偏移量置入BX。
  MOV BX, 0030H
- 将待转换的一个十进制数在表中的序号送入AL中。
  MOV AL, 5
- 执行XLAT命令。
  XLAT



1. 数据传送类

② 目标地址传送指令（3条）

- ✪ 装入有效地址：LEA 目的，源
- ✪ 装入DS寄存器：LDS 目的，源
- ✪ 装入ES寄存器：LES 目的，源

1. 数据传送类：目标地址传送

- **LEA d, s** ；取有效地址指令

用于指定源操作数(需是M操作数)的**16位偏移地址EA**，传送到一个指定的16位通用寄存器中。

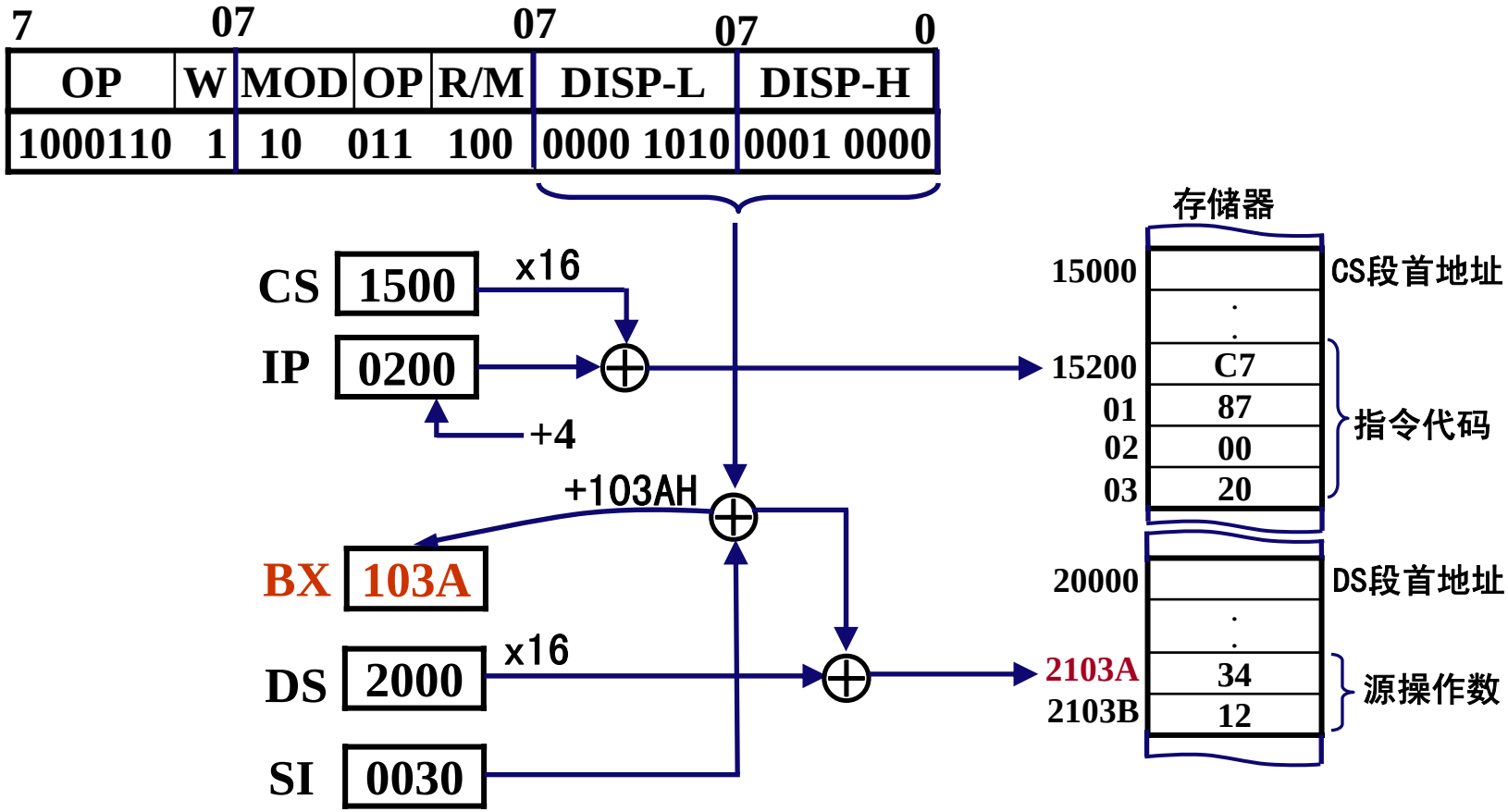
通常用来建立串操作指令所需的寄存器指针。

1. 数据传送类：目标地址传送

例如：LEA BX, [SI+100AH]

CS=1500H，IP=0200H，DS=2000H，SI=0030H。

源操作数为1234H。



1. 数据传送类：目标地址传送

注意和LEA和MOV指令区别。

例如：

```
LEA AX, [0618H]    ；将内存单元的偏移量0618H送AX，
                   ；指令执行后，AX中的内容为0618H

LEA SP, [3768H]    ；使堆栈指针SP为3768H

LEA BX, [BP＋DI]   ；将内存单元的偏移量BP+DI送BX，
                   ；指令执行后，BX中内容为BP+DI的值
```

1. 数据传送类： 目标地址传送

✪ **LDS d, s** ； 取某变量的32位地址指针的指令

从源操作数所指定的存储单元开始，由4个连续存储单元中取出某变量的地址指针(共4个字节)。

将前两个字节(偏移地址)传送到目标操作数所指定的16位通用寄存器中，后两字节(段地址)传送到DS段寄存器中。

1. 数据传送类：目标地址传送

例如：LDS SI, [DI+100AH]

CS=1000H，IP=0604H，DS=2000H，DI=2400H，待传送的某变量的地址指针，其偏移地址为0180H，段地址为2230H。

执行结果：

PA=DSx16+EA=20000H+2400H+100AH=2340AH

将物理地址2340AH单元开始的4个字节中，

前两个字节(偏移地址)0180H传送到SI寄存器中，

后两个字节(段地址)2230H传送到DS寄存器中，

并取代它的原值2000H。

1. 数据传送类： 目标地址传送

- **LES d, s** ； 取某变量的32位地址指针的指令

从源操作数所指定的存储单元开始，由4个连续存储单元中取出某变量的地址指针(共4个字节)。

将前两个字节(偏移地址)传送到目标操作数所指定的16位通用寄存器中，后两字节(段地址)传送到ES段寄存器中。

1. 数据传送类： 目标地址传送

例如：LES DI, [BX]

DS=B000H，BX=080AH，

B080AH单元指定的存储字为05A2H，

B080CH单元指定的存储字为4000H。

执行结果：

PA=DSx16+EA=B0000H+080AH=B080AH

前两个字节(偏移地址)05A2装入DI，

后两个字节(段地址)4000H装入ES，

即 DI=05A2H，ES=4000H。

1. 数据传送类

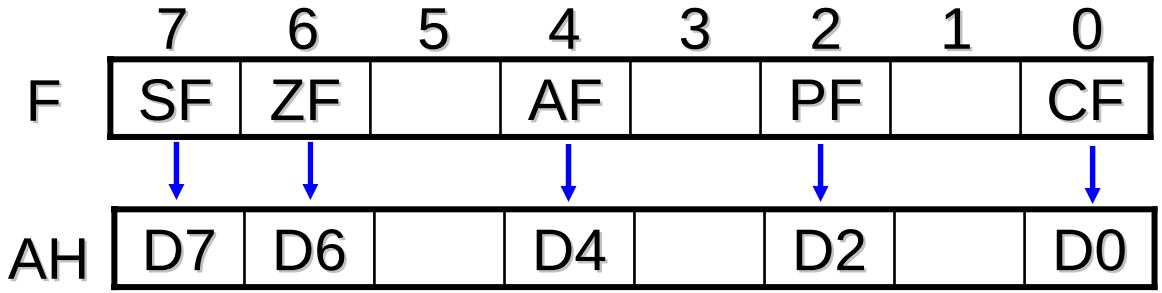
③ 标志位传送指令（4条）

- 将FR低字节装入AH寄存器：LAHF
- 将AH内容装入FR低字节：SAHF
- 将FR内容压入堆栈：PUSHF
- 从堆栈弹出FR内容：POPF

1. 数据传送类：标志位传送

- **LAHF** ；将标志寄存器F的低字节传送到AH寄存器中

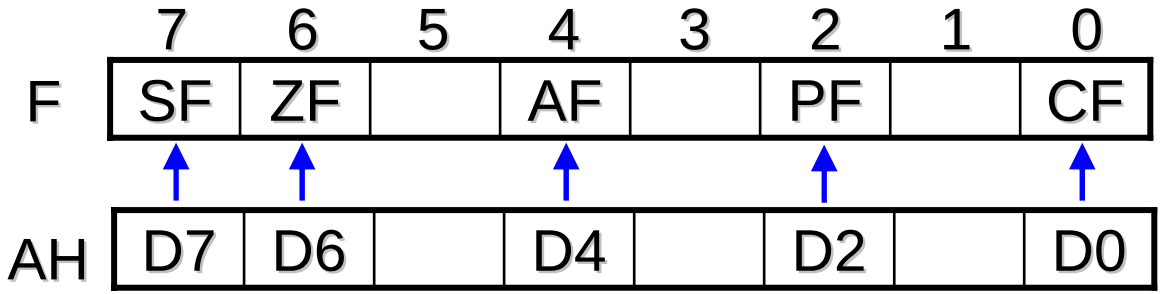
通过AH对标志寄存器的SF、ZF、AF、PF、CF标志位复位



1. 数据传送类：标志位传送

- **SAHF** ；将AH寄存器内容传送到标志寄存器F的低字节

通过AH对标志寄存器的SF、ZF、AF、PF、CF标志位置位



1. 数据传送类：标志位传送

- **PUSHF** ；将16位标志寄存器F内容入栈保护
- **POPF** ；将当前栈顶和次栈顶中的数据字弹出，
  ；送回标志寄存器中

经常**成对出现**，用在子程序和中断处理程序的首尾，用来保护和恢复主程序涉及的标志寄存器内容。

必要时可用来修改标志寄存器内容。

1. 数据传送类

④ I/O数据传送指令（2条）

- ✪ 输入字节或字：IN 累加器，端口
- ✪ 输出字节或字：OUT 端口，累加器

特点：

- I/O指令只能用累加器作为执行I/O数据传送的机构；
- 直接寻址I/O指令：寻址范围0～255；
- 间接寻址I/O指令：寻址范围0～65535；
- I/O设备地址两种形式：固定端口和可变端口。

1. 数据传送类：I/O数据传送

- **IN 累加器，端口号**

；指定端口中内容输入到累加器AL/AX

端口号可由8位立即数直接给出；
也可由DX寄存器间接给出16位端口号。

IN AL, PORT ；AL ← (端口PORT)

IN AX, PORT ；AX ← (端口PORT)

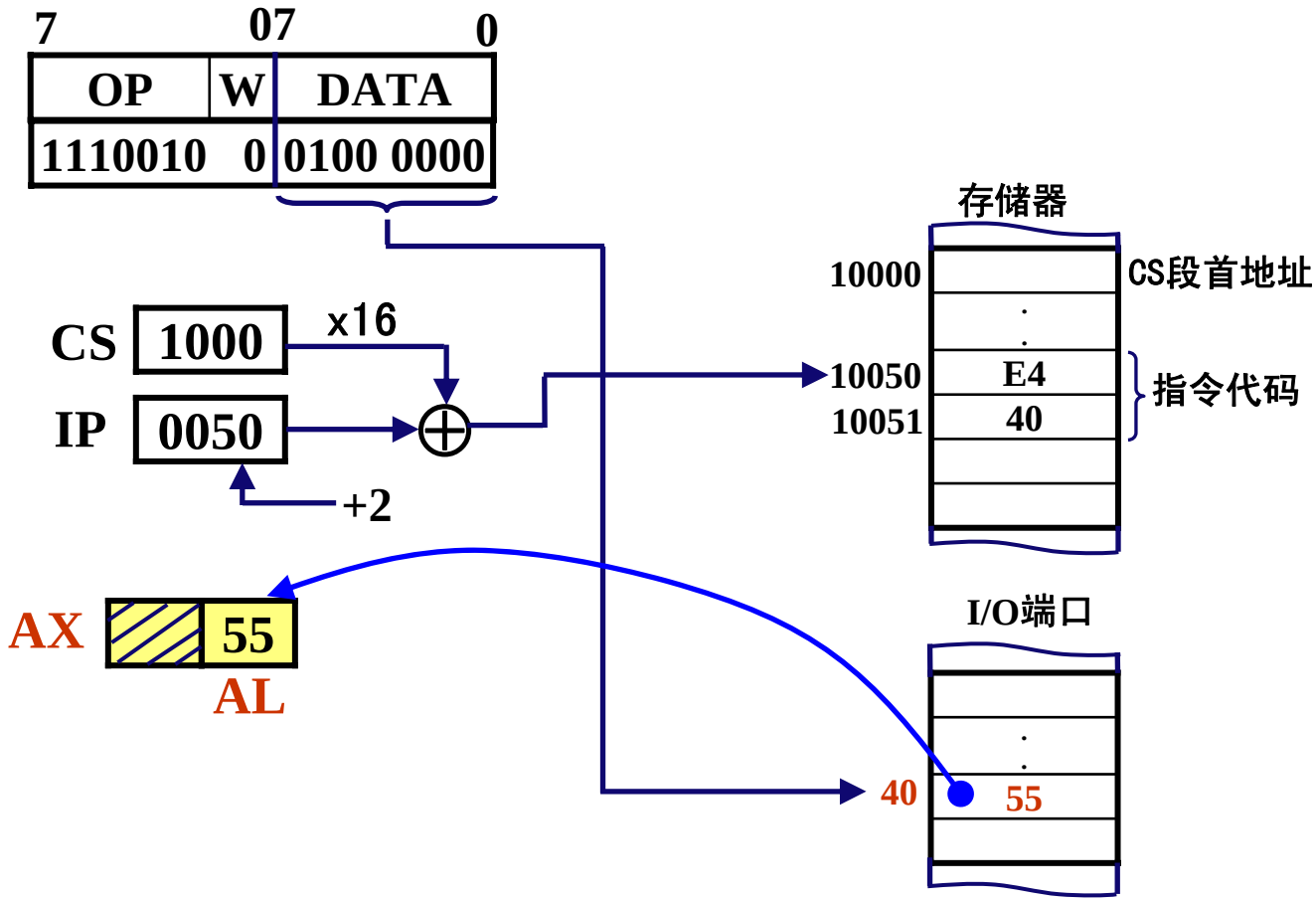
OUT DX,AL ；端口(DX) ← AL

OUT DX,AX ；端口(DX) ← AX

1. 数据传送类：I/O数据传送

例如：IN AL, 40H

CS=1000H，IP=0050H，8位端口40H中内容为55H。



1. 数据传送类：I/O数据传送

✪ **OUT 端口号，累加器**

；累加器AL/AX中内容输出到指定端口

端口号可由8位立即数直接给出；
也可用DX寄存器间接给出16位端口号。

OUT PORT,AL ；端口PORT ← AL

OUT PORT,AX ；端口PORT ← AX

OUT DX,AL ；端口(DX) ← AL

OUT DX,AX ；端口(DX) ← AX

1. 数据传送类：I/O数据传送

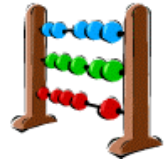
例如：OUT DX, AL

CS=4000H，IP=0020H，DX=6A10H，AL＝66H。

执行结果：

将累加器AL中的数据字节66H，输出到DX指定的端口6A10H中。

2. 算术运算类



- 无符号/有符号、8/16位二进制数运算：加减乘除
- 无符号压缩型/非压缩型十进制运算：十进制调整
- 根据运算结果影响状态标志，有时要利用某些标志才能得到正确的结果；使用时请留心有关状态标志。

共20条，分为5小类：

- 加法（3条）
- 减法（5条）
- 乘法（2条）
- 除法（4条）
- 十进制调整（6条）

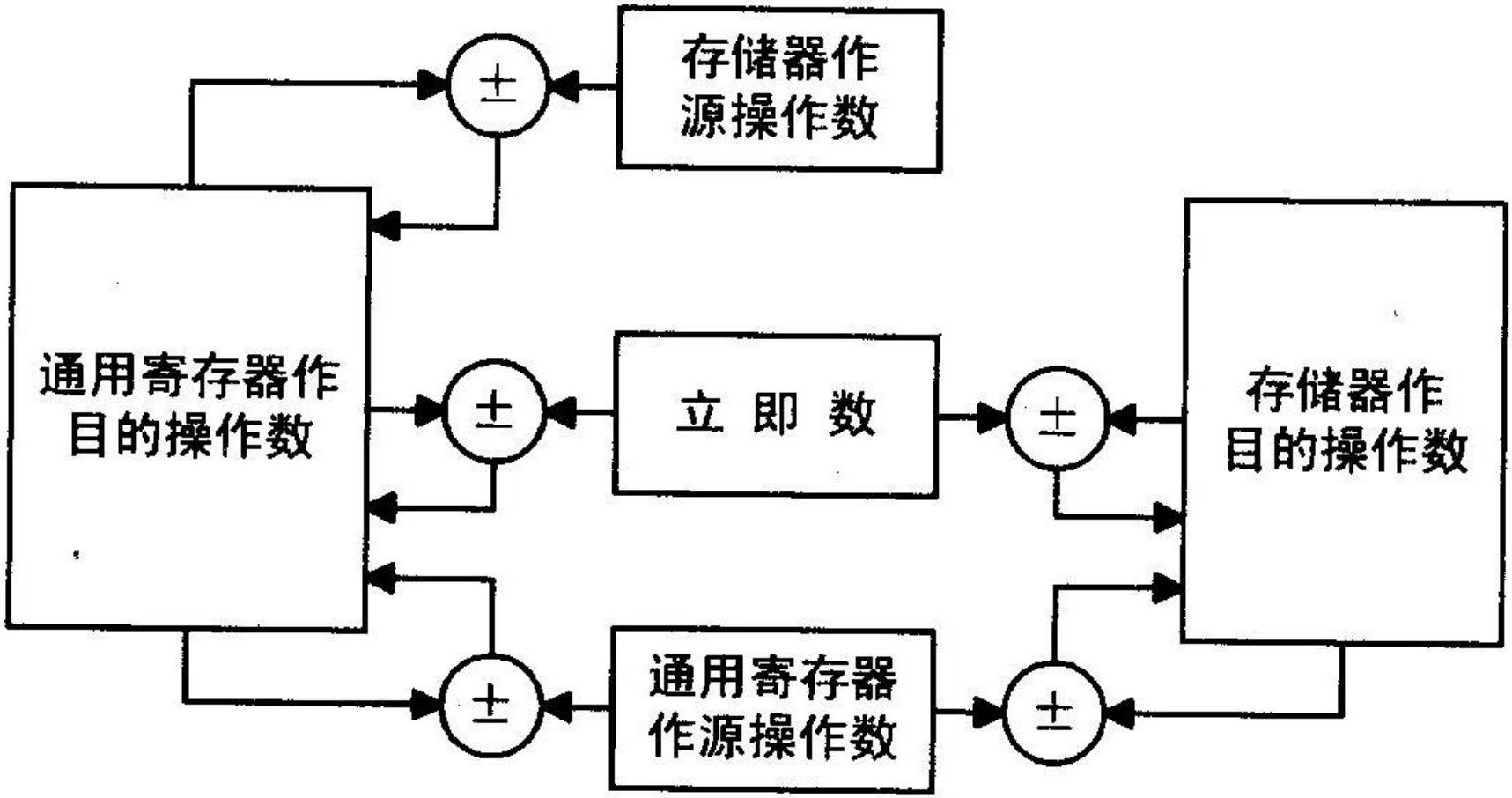
2. 算术运算类：加法

指令名称	指令书写格式 (助记符)	状态标志位 O S Z A P C
加法(字/字节)	ADD **d, s**	↑ ↑ ↑ ↑ ↑ ↑
带进位加法(字/字节)	ADC **d, s**	↑ ↑ ↑ ↑ ↑ ↑
加1(字/字节)	INC **d**	↑ ↑ ↑ ↑ ↑ •

↑ — 表示运算结果影响标志位

• — 表示运算结果不影响标志位

2. 算术运算类：加法



2. 算术运算类：加法

✪ **ADD d, s** ；d←d＋s

功能： 源操作数和目的操作数相加，结果送到目的操作数。

源操作数： 8/16位通用寄存器，存储器，立即数

目的操作数： 8/16位通用寄存器，存储器

提示：

目的操作数不能为立即数；

源操作数和目的操作数不能同时为存储器。

2. 算术运算类：加法

例1： ADD [BX+106BH], 1234H

设： CS=1000H, IP=0300H, DS=2000H, BX=1200H

执行结果：

EA=BX+106BH=226BH

PA=DSx16+EA=2226BH

将立即数1234H，与存储器单元2226BH和2226CH中的3344H相加，和为4578H，结果保留在目的地址2226BH和2226CH单元中。

运算结果影响标志位。

O	S	Z	A	P	C
0	0	0	0	1	0

2. 算术运算类：加法

例2：寄存器加法。

若将AX、BX、CX和DX内容累加，再将所得的16位的和数存入AX。

程序为：

```
ADD AX,BX    ;  AX←AX＋BX
ADD AX,CX    ;  AX←AX＋BX＋CX
ADD AX,DX    ;  AX←AX＋BX＋CX＋DX
```

2. 算术运算类：加法

例3：立即数加法。常数和立即数相加时。

若将立即数12H取入DL，然后用立即数加法指令再将34H加到DL中的12H上，所得的结果放在DL。

程序为：

```
MOV DL,12H    ;  DL←12H
ADD DL,34H    ;  DL←DL＋34H
```

O	S	Z	A	P	C
0	0	0	0	0	0

2. 算术运算类：加法

例4：存储器与寄存器的加法。

将存储在数据段中，偏移地址为NUMB和NUMB+1连续单元的字节数据累加到AL。

程序为：

```
MOV DI,OFFSET NUMB  ；偏移地址NUMB装入DI
ADD AL,0            ；AL清零
ADD AL,[DI]         ；将NUMB单元的字节内容加上
                    ；AL，和数存AL
ADD AL,[DI＋1]      ；累加NUMB＋1单元中的字节
                    ；内容，累加结果存放在AL
```

2. 算术运算类：加法

例5：数组加法。存储器数组是一个按顺序排列的数据表。

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                ；存放和数的AL清0
MOV SI,3                ；将SI指向元素3
ADD AL,ARRAY[SI]        ；加元素3
ADD AL,ARRAY[SI+2]      ；加元素5
ADD AL,ARRAY[SI+4]      ；加元素7
```

2. 算术运算类：加法

✪ **ADC d, s** ；d←d＋s＋CF

功能：源操作数和目的操作数相加外，再加上进位标志CF，结果送到目的操作数。

CF将重新根据结果置成新的状态。

ADC指令一般用于16位以上的多字节数字相加。

例如：

```
ADC AX,BX          ；AX=AX+BX+C
ADC BX,[BP+2]      ；由BX+2寻址的堆栈段存储单元的字
                   ；内容，加上BX和进位位，结果存入BX。
```

2. 算术运算类：加法

例1：假定实现BX与AX中4字节数据，与DX和CX中4字节数据相加，其结果存入BX和AX中。

程序为：

```
ADD AX,CX        ;  AX←AX+CX

ADC BX,DX        ;  BX←BX+DX+CF
```

2. 算术运算类：加法

✪ **INC d** ；d←d+1

功能：将目的操作数当作无符号数，加1后，结果送到目的操作数。

目的操作数：8/16位通用寄存器，存储器；
但不允许为立即数。

间接寻址的存储器单元加1时，数据长度需用伪指令说明；

INC指令只影响OF、SF、ZF、AF、PF5个标志，不影响CF。

循环程序中，常用该指令对地址指针和循环计数值进行修改。

2. 算术运算类：加法

例如：

```
INC SP；SP=SP+1

INC BYTE PTR[BX+1000H]  ；将数据段中由BX+1000H
                        ；寻址的存储单元的字内容
                        ；加1

INC WORD PTR[SI]        ；将数据段中由SI寻址的存储单元
                        ；的字内容加1

INC DATA1               ；将数据段中DATA1存储单元的内容加1
```

2. 算术运算类：减法

指令名称	指令书写格式(助记符)	状态标志位					
		O	S	Z	A	P	C
减法(字/字节)	ADD **d, s**	↑	↑	↑	↑	↑	↑
带进位减法(字/字节)	ADC **d, s**	↑	↑	↑	↑	↑	↑
减1(字/字节)	INC **d**	↑	↑	↑	↑	↑	•
取负	NEC **d**	↑	↑	↑	↑	↑	1
比较	CMP **d, s**	↑	↑	↑	↑	↑	↑

↑ — 表示运算结果影响标志位

• — 表示运算结果不影响标志位

2. 算术运算类：减法

✪ **SUB d, s** ；d←d - s

功能： 目的操作数减去源操作数，结果送到目的操作数。

源操作数： 8/16位通用寄存器，存储器，立即数

目的操作数： 8/16位通用寄存器，存储器

提示：

目的操作数不能为立即数；

源操作数和目的操作数不能同时为存储器。

不允许进行段寄存器减法。

2. 算术运算类：减法

例1：SUB AX, [BX]

设： CS=1000H, IP=60C0H, DS=2000H, BX=970EH
AX=8811H

执行结果：

EA=BX=970EH

PA=DSx16+EA=2970EH

将AX寄存器中的目的操作数8811H，减去存储器单元2970EH和2970FH中的源操作数00FFH相减，结果8712H送回AX。

运算结果影响标志位。

O	S	Z	A	P	C
0	1	0	1	1	0

2. 算术运算类：减法

✪ **SBB d, s** ； d←d – s - CF

功能： 目的操作数减去源操作数外，再减去进位标志位CF，结果送到目的操作数。

CF将重新根据结果置成新的状态。

SBB指令一般用于16位以上的多字节数字相减。

例如：

```
SBB AX,BX                    ; AX=AX-BX-CF
SBB WORD PTR[DI],50A0H       ; 由DI寻址的数据段存储
                             ; 单元的内容，减去50A0H
                             ; 及CF值。
```

2. 算术运算类：减法

例1：假定从存于BX和AX中的4字节数据，与存于DX和CX中4字节数据相减，其结果存入BX和AX中。

程序为：

```
SUB AX,DI        ;  AX←AX-CX
SBB BX,SI        ;  BX←BX-SI-CF
```

2. 算术运算类：减法

✪ DEC d　　；d←d - 1

功能：将目的操作数减1后，结果送到目的操作数。

目的操作数：8/16位通用寄存器，存储器；
但不允许为立即数。

间接寻址的存储器单元减1时，数据长度需用TYPE PTR类型伪指令来标识数据长度。

DEC指令只影响OF、SF、ZF、AF、PF5个标志，不影响CF。

循环程序中，常用该指令对地址指针和循环计数值进行修改。

2. 算术运算类：减法

例如：

```
DEC BL                  ；BL=BL-1

DEC CX                  ；CX=CX-1

DEC BYTE PTR[DI]        ；由DI寻址的数据段中，字节存储
                        ；单元的字内容减1

DEC WORD PTR[BP]        ；由BP寻址的堆栈段中，字存储
                        ；单元的内容减1
```

2. 算术运算类：减法

- NEG d ；d← $\overline{d}$ +1 求补码指令
- NEG d ；d←0 - d

功能：将目的操作数取负后，结果送到目的操作数。

目的操作数：8/16位通用寄存器，存储器；
但不允许为立即数。

NEG指令对标志的影响与用0作被减数的SUB指令一样。
但进位标志CF=1

2. 算术运算类：减法

例1：NEG BYTE PTR[BX]

设：CS=1000H, IP=200AH, DS=2000H, BX=3000H

EA=BX=3000H

PA=DSx16+EA=23000H

假设23000H存储单元内容为字节变量FDH

即　FDH=$[-3]_{补}$

则　+3被送回物理地址23000H单元中。

2. 算术运算类：减法

✪ CMP d, s　　；d - s　只置标志位

功能：目的操作数减去源操作数，不送回结果。
只根据运算结果置标志位。

源操作数：8/16位通用寄存器，存储器，立即数

目的操作数：8/16位通用寄存器，存储器

提示：

目的操作数不能为立即数；

源操作数和目的操作数不能同时为存储器；

不允许进行段寄存器比较。

2. 算术运算类：减法

例1：

```
CMP BL,CL          ；BL-CL

CMP AX,SP          ；AX-SP

CMP AX,1000H       ；AX-1000H

CMP [DI],BL        ；由DI寻址的数据段存储单元的字节内容
                   ；减BL

CMP CL,[BP]        ；由CL减由BP寻址的堆栈段存储单元的
                   ；字节内容

CMP SI,TEMP[BX]    ；由SI减由TEMP+BX寻址的数据段
                   ；存储单元的字
```

2. 算术运算类：减法

例2 要将CL的内容与20H作比较，当CL ≥64H时，程序转向存储器地址SUBER处继续执行。

程序为：

```
CMP CL,64H      ；  CL与64H作比较
JAE SUBER       ；  如果等于或大于则跳转
```

2. 算术运算类：减法

例3 用比较指令判断两个数大小。

无符号数：根据借位标志CF判断。

ZF=1，d=s； ZF=0，d≠s

CF=0，表示无借位，即d ≥ s；

CF=1，表示有借位，即d<s；

有符号数：根据溢出标志OF和符号标志SF两者的异或运算结果来判断。

OF⊕SF=0时，则d≥s；

OF⊕SF=1时，则d<s；

通常，比较指令后面跟一条条件转移指令，检查标志位的状态决定程序的转向。

2. 算术运算类：乘法

指令名称	指令书写格式(助记符)	状态标志位					
		O	S	Z	A	P	C
不带符号乘法(字/字节)	MUL s	↑	×	×	×	×	↑
带符号整数乘法(字/字节)	IMUL s	↑	×	×	×	×	↑

x — 表示标志位为任意值

↑ — 表示运算结果影响标志位

2. 算术运算类：乘法

✪ MUL s ；无符号乘法指令

功能：完成两个无符号的8/16位二进制数相乘。

被乘数：隐含在累加器AL/AX中

乘数：指令中由s指定的源操作数；
8/16位通用寄存器或存储器操作数。

积：相乘后得到双倍长的积

8位二进制数乘法：其16位积的高8位存于AH，
低8位存于AL；

16位二进制数乘法：其32位积的高16位存于DX，
低16位存于AX；

2. 算术运算类：乘法

利用CF和OF标志可判断相乘结果的高位字节或高位字是否有效：

高位字节或高位字有效：即AH≠0或DX≠0，则将CF和OF两标志位同时置1

高位字节或高位字无效：即AH=0或DX=0，则将CF和OF两标志位同时置0

2. 算术运算类：乘法

例1：MUL BYTE PTR[BX+2AH]

设：CS=3000H, IP=0250H, DS=2000H, BX=0234H
AL=12H，源操作数定义为字节变量66H。

EA=BX+2AH=025EH

PA=DSx16+EA=2025EH

被乘数：12H

乘数：66H

乘积：12Hx66H=072CH

AH≠0，故CF=1，OF=1；其余标志为任意状态。

2. 算术运算类：乘法

✪ IMUL src　　；有符号乘法指令

功能：完成两个带符号的8/16位二进制数相乘。

IMUL指令除计算对象为带符号二进制数以外，其它都与MUL一样的，但结果不同。

IMUL指令对OF和CF影响：

若乘积的高一半是低一半的符号扩展，
则OF=CF=0；否则均为1。

2. 算术运算类：乘法

例1：

```
IMUL CL       ；AX ← (AL)x(CL)

IMUL CX       ；DX、AX ← (AX)x(CX)

IMUL BYTE PTR[BX]
              ；AX ← (AL)x[BX]
              ；即AL中的、和BX所指内存单元中的
              ；两个8位有符号数相乘，结果送AX中。

IMUL WORD PTR[DI]
              ；DX、AX ← (AX)x[DI]
              ；即AX中的、和DI与DI+1所指内存单元中的
              ；两个8位有符号数相乘，结果送DX和AX中。
```

2. 算术运算类：除法

指令名称	指令书写格式(助记符)	状态标志位
		O S Z A P C
不带符号除法(字/字节)	DIV s	x x x x x x
带符号整数除法(字/字节)	IDIV s	x x x x x x
字节转换成字	CBW	• • • • • •
字转换成双字	CWD	• • • • • •

x — 表示标志位为任意值

• — 表示运算结果不影响标志位

2. 算术运算类：除法

✪ DIV src　　；无符号除法指令

功能：完成两个无符号二进制数相除。

被除数：隐含在累加器AX(字节除)或DX、AX(字除)中

除数：指令中由s指定的源操作数
8/16位通用寄存器或存储器操作数

商：字节除法－商存于AL中，余数存于AH中
字除法－商存于AX中，余数存于DX中

余数：根据8086约定，余数符号应与被除数符号一致。

2. 算术运算类：除法

若运算所得商数超出累加器容量，则系统将其当作除数为0处理，自动产生类型0中断，CPU将转去执行类型0中断服务程序作适当处理，此时所得商数和余数均无效。

类型0中断处理时：

- 先将标志位进堆栈，IF和TF清零；
- 接着CS和IP的内容进堆栈；
- 然后将0,1两单元内容填入IP，2,3两单元内容填入CS；
- 最后，再进入0号中断的处理程序。

2. 算术运算类：除法

例1：DIV BYTE PTR[BX+SI]

设：CS=1000H, IP=0406H, DS=3000H, BX=2000H
SI=050EH，AX=1500H，存储器中的源操作数
定义为字节变量22H。

EA=BX+SI=2000H+050EH=250EH

PA=DSx16+EA=3250EH

被除数：1500H

除数：22H

商：AL=9EH

余数：AH=04H

2. 算术运算类：除法

✪ IDIV src　　；有符号除法指令

功能：完成两个带符号二进制数相除。

与DIV指令主要区别：对符号位处理的约定。

如果源操作数是字节/字数据：

- 被除数应为字/双字数据，并隐含存放于AX/DX、AX中。
- 如被除数也是字节/字数据在AL/AX中，应将AL/AX的符号位(AL_7)/(AX_{15})扩展到AH/DX中，才能开始字节/字除法运算。

2. 算术运算类：除法

例1：

```
IDIV BX    ；将DX和AX中的32位数，除以BX中的16位数，
           ；所得的商在AX中，余数在DX中。

IDIV BYTE PTR[SI]
           ；将AX中的16位数，除以SI所指内存
           ；单元的8位数，所得的商在AL中，余数在AH中。
```

2. 算术运算类：除法

✪ CBW和CWD是两条专门为IDIV指令设置的符号扩展指令。

- 符号扩展是指用一个操作数的符号位(最高位)形成另一 个操作数，后一个操作数的高位是全0(正数)或全1(负数)。
- 符号扩展虽然使数据位数加长，但数据大小并没有改变，扩展的高部分仅是低部分的符号扩展。
- 符号扩展指令有两条，用来将字节转换为字，字转换为双字。

```
CBW     ；AL符号扩展成AX
CWD     ；AX符号扩展成DX
```

2. 算术运算类：除法

```
MOV AL,64H
    ；AL＝64H （机器数），表示10进制数100（真值）
CBW
    ；将符号0扩展，AX=0064H，仍然表示100

MOV AX,0FF00H
    ；AX=FF00H，表示有符号10进制数 - 256
CWD
    ；将符号位“1”扩展，DX.AX=FFFFFF00H
    ；仍然表示－256
```

2. 算术运算类：除法

例如：在B1，B2，B3字节类型变量中，分别存有8位带符号数a，b，c，实现(a*b＋c)/a运算。

程序如下:

```
MOV   AL,B1     ；a→(AL)
IMUL  B2        ；实现a*b→(AX)
MOV   CX,AX     ；(AX)→(CX)
MOV   AL,B3     ；C→(AL)
CBW             ；扩展符号位至AH中
ADD   AX,CX     ；(AX)＋(CX)→(AX),完成a*b＋c
IDIV  B1        ；完成(a*b＋c)/a,商→(AL),余数→(AH)
```

2. 算术运算类：十进制调整

指令名称	指令书写格式(助记符)	状态标志位					
		O	S	Z	A	P	C
加法的ASCII码调整	AAA	x	x	x	↑	x	1
加法的十进制调整	DAA	x	↑	↑	↑	↑	x
减法的ASCII码调整	AAS	x	x	x	↑	x	↑
减法的十进制调整	DAS	↑	↑	↑	↑	↑	↑
乘法的ASCII码调整	AAM	x	↑	↑	x	↑	x
除法的ASCII码调整	AAD	x	↑	↑	•	↑	x

x — 表示标志位为任意值

• — 表示运算结果不影响标志位

1 — 表示标志位置1

↑ — 表示运算结果影响标志位

2. 算术运算类：十进制调整

- 对二进制运算的结果进行十进制调整，以得到十进制的运算结果，以此实现十进制BCD码运算

- **8086指令系统支持两种BCD码调整运算**
  - **压缩BCD码：**8421码，用4个二进制位表示一个十进制位，一个字节可以表示两个十进制位，即00～99
  - **非压缩BCD码：**用8个二进制位表示一个十进制位，只用二进制的低4位表示一个十进制位0～9；高4位任意，通常默认为0

2. 算术运算类：十进制调整

前述的指令均为二进制数运算结果，即使参加运算的数是BCD数，其结果仍是二进制数运算所得的结果，而不是十进制数的应得结果。

例如：8的BCD码加5的BCD码，结果为0DH，它不在0～9之间，也不是期望的结果13。

因此，需要对结果进行BCD调整。显然，未压缩组合的十进制数中，每个字节存放一位BCD数，调整后，若有高位的进位(通常让AF=1)，不应在同一字节的高4位加1，而应该在高位字节(例如在AH中)加1。

具体的调整操作由指令功能给出。

2. 算术运算类：十进制调整

真值 (十进制)	8	64
二进制编码	08H	40H
压缩BCD码	08H	64H
非压缩BCD码	08H	0604H
ASCII码	38H	3634H

- 压缩BCD码加、减法调整指令

  DAA　DAS

- 非压缩BCD码加、减、乘、除法调整指令

  AAA　AAS　AAM　AAD

2. 算术运算类：十进制调整

✪ **DAA （Decimal Adjust for Addition）**

；加法的十进制调整，它必须跟在ADD或ADC
；指令之后使用

功能： 将存于AL寄存器中的2位BCD码加法运算的结果，调整为2位压缩型十进制数，仍保留在AL中。

2. 算术运算类：十进制调整

例1：(AL)＝18H，(BL)＝06H

```
ADD AL, BL      ；(AL)←(AL)＋(BL)
                ；(AL)＝1EH
DAA             ；(AL)＝24H, AF＝1
```

本指令影响标志位AF，CF，PF，SF，ZF。

2. 算术运算类：十进制调整

例2 设AX=6698H，BX=2877H，如果要将这两个十进制数相加，结果保留在AX中。

程序为：

```
ADD AL,BL     ；低字节相加
DAA           ；低字节调整
MOV CL,AL     ；保存AL内容
MOV AL,AH
ADC AL,BH     ；高字节相加
DAA           ；高字节调整
MOV AH,AL
MOV AL,CL
```

2. 算术运算类：十进制调整

✪ **DAS**（Decimal Adjustfor Subtraction）

；减法的十进制调整，它必须跟在SUB或SBB

；指令之后使用

功能：将存于AL寄存器中的减法运算的结果，
调整为2位压缩型十进制数，仍保留在AL中。

2. 算术运算类：十进制调整

✪ **AAA （ASCII Adjust for Addition）**

；加法的ASCII码调整指令，它必须跟在ADD和ADC
；指令之后使用

功能：将存于AL寄存器中的1位ASCII码数加法运算的结果调整为1位非压缩型十进制数，仍保留在AL中。

只影响标志AF和CF

2. 算术运算类：十进制调整

例1 若有两个用ASCII码表示的2位十进制数分别存放在AX和BX寄存器中，即AX=0011100000110111，BX=0011100100110101。
要求将两数相加，把结果保留在AX中，如有进位，将进位置入DX中。

程序为：

```
MOV DX,0
MOV CX,AX    ; CX='87'
MOV AH,0
ADD AL,BL    ; AL ← '7'+'5'
AAA          ; AH=01H, AL=02H
MOV CL,AL    ; CL=02H
MOV AL,CH    ;  AL='8'
ADD AL, AH
AAA          ; AL=09H
MOV AH,0
ADD AL,BH
AAA          ; AH=01H, AL=08H
MOV CH,AL    ; CH=08H
ADD DL, AH   ; DL=01H
MOV AX,CX    ; AX=0802H
```

DX	00000000	00000001
AX	00001000	00000010

2. 算术运算类：十进制调整

✪ **AAS （ASCII Adjust for Subtraction）**

；减法的ASCII码调整指令，它必须跟在SUB和SBB
；指令之后使用

功能：将存于AL寄存器中的减法运算的结果调整为1位非压缩型十进制数，仍保留在AL中。

如果有借位，则保留在借位标志CF中。
只影响标志AF和CF

2. 算术运算类：十进制调整

✪ **AAM （ASCII Adjust for Multiply）**

；乘法的ASCII码调整指令，跟在MUL指令之后。

功能：将AL寄存器中的乘法运算的结果调整为2位非压缩型十进制数，其高位AH中，低位在AL中。

参加乘法运算的十进制数必须是非压缩型，故通常在MUL指令之前安排两条AND指令。

程序为：

```
AND AL,0FH
AND BL,0FH
MUL BL
AAM
```

2. 算术运算类：十进制调整

✪ **AAD （ASCII Adjust for Divison）**

；除法的ASCII码调整指令，在除法之前进行。

功能：将AX寄存器中的2位非压缩型十进制数的被除数，调整为二进制数，保留在AL中。

例1 实现 0103÷06=02 余01。

```
MOV AX,0103   ；取被除数
MOV BL,06     ；取除数
AAD           ；调整为 (AX)=000DH
DIV BL        ；相除，商(AL)=02，余数(AH)=01
```

3. 逻辑运算和移位循环类

共13条，分为3小类：

- 逻辑运算（5条）
- 移位（4条）
- 循环（4条）

3. 逻辑运算和移位循环类：逻辑运算

指令名称	指令书写格式(助记符)	状态标志位					
		O	S	Z	A	P	C
“与”(字节/字)	AND d,s	0	↑	↑	x	↑	0
“或”(字节/字)	OR d,s	0	↑	↑	x	↑	0
“异或”(字节/字)	XOR d,s	0	↑	↑	x	↑	0
“非”(字节/字)	NOT d	•	•	•	•	•	•
测试(字节/字)	TEST d,s	0	↑	↑	x	↑	x

x — 表示标志位为任意值

• — 表示运算结果不影响标志位

1 — 表示标志位置1

↑ — 表示运算结果影响标志位

3. 逻辑运算和移位循环类：逻辑运算

✪ **AND d, s** ；d←d ∧ s 按位“与”操作

源操作数：8/16位通用寄存器，存储器，立即数

目的操作数：8/16位通用寄存器，存储器

提示：

- 目的操作数不能为立即数。
- 源操作数和目的操作数不能同时为存储器。
- 影响SF,ZF,PF；OF,CF置0；AF无意义。
- 二者均为1，结果为1；否则为0。
- 用来对一个数据的指定位清零。

3. 逻辑运算和移位循环类：逻辑运算

例1： AND AX, ALPHA

设： CS=2000H, IP=0400H, DS=1000H, AX=F0F0H, ALPHA是数据段中偏移地址为0500H和0501H地址中的字变量7788H的名字。

EA=0500H；PA=DSx16+EA=10500H

执行结果：

AX中的F0F0H，与物理地址10500H和105001H单元中数据字7788H，进行逻辑“与”，结果7080H送到AX寄存器中。

```
      1111 0000 1111 0000
AND)  0111 0111 1000 1000
-------------------------
      0111 0000 1000 0000
```

3. 逻辑运算和移位循环类：逻辑运算

例2：对指定位清零。

如将AL高4位清零，(AL)=3AH。

```
AND AL,0FH
```

结果：(AL)=0AH

```
      0011 1010
AND） 0000 1111
-------------
      0000 1010
```

3. 逻辑运算和移位循环类：逻辑运算

✪ **OR d, s** ；d←d ∨ s 按位“或”操作

源操作数：8/16位通用寄存器，存储器，立即数

目的操作数：8/16位通用寄存器，存储器

提示：

- 目的操作数不能为立即数
- 源操作数和目的操作数不能同时为存储器
- 影响SF,ZF,PF；OF,CF置0；AF无意义
- 二者均为0，结果为0；否则为1
- 用来对一个数据的指定位置1

3. 逻辑运算和移位循环类：逻辑运算

例1：将AL最高位置1，(AL)=14H

OR AL, 80H

结果：(AL)=94H

```
    0001 0100
OR）1000 0000
-------------
    1001 0100
```

3. 逻辑运算和移位循环类：逻辑运算

例2：将数字5转换成ASCII形式；

```
MOV AH, 05H
OR AH, 30H
```

结果：(AH)=35H

```
      0000 0101
OR）  0011 0000
-----------------
      0011 0101
```

3. 逻辑运算和移位循环类：逻辑运算

✪ **XOR d, s** ；d←d ⊕ s 按位“异或”操作

源操作数：8/16位通用寄存器，存储器，立即数

目的操作数：8/16位通用寄存器，存储器

提示：

- **目的操作数不能为立即数**
- **源操作数和目的操作数不能同时为存储器**
- **影响SF,ZF,PF；OF,CF置0；AF无意义**
- **二者相反，结果为1；否则为0**
- 用来使某个寄存器清零，如 XOR AX, AX

3. 逻辑运算和移位循环类：逻辑运算

例1：比较两个操作数是否相同。

如判断AL中数据是否为3CH。

XOR AL,3CH

结果：ZF=1，则(AL)=3CH

ZF=0，不等

3. 逻辑运算和移位循环类：逻辑运算

例2：将指定的数据变反，(AL)=3AH。

```
XOR AL,0FFH;
```

结果：(AL)=C5H

```
        0011 1010
XOR）   1111 1111
-------------------
        1100 0101
```

3. 逻辑运算和移位循环类：逻辑运算

✪ NOT d ；d ← 0FFH/0FFFF - d

；求得操作数反码后，再送回目的操作数

目的操作数：8/16位通用寄存器，存储器

提示：

– 目的操作数不能为立即数；

– 不标志影响

– NOT AX 相当于 INC AX NEG AX（求补码）

3. 逻辑运算和移位循环类：逻辑运算

例1： NOT AH；

其中(AH)=13H

结果：(AH)=ECH

```
      1111 1111
-)    0001 0011
-------------------
      1110 1100
```

3. 逻辑运算和移位循环类：逻辑运算

例2： NOT WORD PTR[1000H]；

其中1000H和1001H单元中的16位数为2FC3H。

结果：1000H和1001H单元中为D03CH

```
    1111 1111 1111 1111
-）  0010 1111 1100 0011
-------------------------
    1101 0000 0011 1100
```

3. 逻辑运算和移位循环类：逻辑运算

✪ **TEST d, s** ；d ∧ s 按位“与”操作
；不送回操作数，操作数不变

源操作数：8/16位通用寄存器，存储器，立即数

目的操作数：8/16位通用寄存器，存储器

提示：

- **目的操作数不能为立即数；**
- **源操作数和目的操作数不能同时为存储器；**
- **影响SF,ZF,PF；OF,CF置0；AF无意义**
- 二者均为1，结果为1；否则为0
- 用来检测指定位是1还是0

3. 逻辑运算和移位循环类：逻辑运算

例1：测试AL的最高位D_7是否为1（即正数/负数）

```
TEST AL,80H;
```

结果：ZF=0，则AL最高位为1
ZF=1，则AL最高位为0

例2：测试(BX)所指存储单元的最低位D_0是否为1（即奇数/偶数）；

```
TEST [BX],01H;
```

结果：ZF=0，则AL最高位为1
ZF=1，则AL最高位为0

3. 逻辑运算和移位循环类：逻辑运算

```
AND BL,11110110B
                ;BL中D0和D3清0，其余位不变
OR BL,00001001B
                ;BL中D0和D3置1，其余位不变
XOR BL,00001001B
                ;BL中D0和D3求反，其余位不变
```

- AND指令可用于复位某些位（同0相与），不影响其他位
- OR指令可用于置位某些位（同1相或），不影响其他位
- XOR指令可用于求反某些位（同1相异或），不影响其他位

3. 逻辑运算和移位循环类：移位

指令名称	指令书写格式 (助记符)	状态标志位					
		O	S	Z	A	P	C
算术左移(字节/字)	SAL d, count	↑	↑	↑	x	↑	↑
算术右移(字节/字)	SAR d, count	↑	↑	↑	x	↑	↑
逻辑左移(字节/字)	SHL d, count	↑	↑	↑	x	↑	↑
逻辑右移(字节/字)	SHR d, count	↑	↑	↑	x	↑	↑

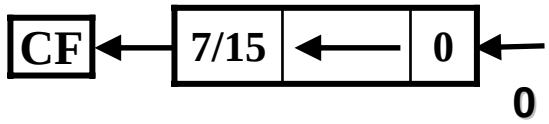
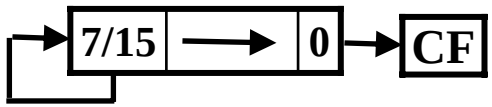
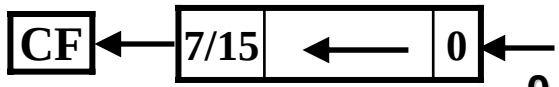
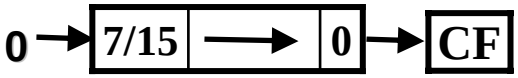
x — 表示标志位为任意值

↑ — 表示运算结果影响标志位

3. 逻辑运算和移位循环类：移位

- 将操作数移动1位或多位，分逻辑移位和算术移 位，分别具有左移或右移操作。
- 移位指令的目的操作数是指定的被移位的操作数，可以是8/16位通用寄存器或存储单元；计数值COUNT表示移位位数：
  - COUNT为1：表示移动1位，指令的COUNT字段可直接写1
  - 若移动n位(n≤255)：则n事先装入CL，指令的COUNT字段只能写CL
- 按照移入的位设置进位标志CF，根据移位后的结果影响SF、ZF、PF

3. 逻辑运算和移位循环类：移位

	指令	简图	操作说明	功能
算术移位	SAL 左移		有符号数左移，最高位进入CF，最低位补0。 左移1位后，最高位和CF不同，OF置1，否则OF置0。 移位次数不为1时，OF不确定。	左移1位， 操作数乘2
	SAR 右移		有符号数右移，最高符号位不变，所有位右移1位。	右移1位， 操作数除2
逻辑移位	SHL 左移		无符号数左移，最低位补0，最高位移入CF。 若CF=0，无溢出；若CF＝1，有溢出，倍增(x2)结果是错误的。	左移1位， 无符号数乘2
	SHR 右移		无符号数右移，最高位补0，最右边1位移入CF。 CF=1, 移位前是一个奇数。CF=0, 移位前是一个偶数。	右移1位， 无符号数除2

3. 逻辑运算和移位循环类：移位

✪ **SAL reg/mem, 1/CL**

；算术左移指令

；reg/mem左移1或CL位

；最低位补0，最高位进入CF

演示

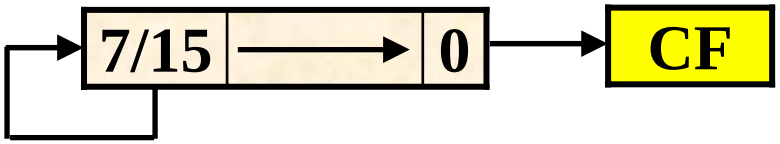
3. 逻辑运算和移位循环类：移位

✪ SAR reg/mem,1/CL

；算术右移指令

；reg/mem右移1/CL位

；最高位不变，最低位进入CF



演示

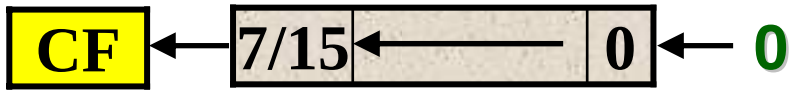
3. 逻辑运算和移位循环类：移位

✪ SHL reg/mem,1/CL

；逻辑左移指令

；reg/mem左移1或CL位

；最低位补0，最高位进入CF



演示

3. 逻辑运算和移位循环类：移位

✪ SHR reg/mem,1/CL

；逻辑右移指令

；reg/mem右移1/CL位

；最高位补0，最低位进入CF

演示

3. 逻辑运算和移位循环类：移位

例：将AL寄存器中的无符号数乘以10

```
XOR AH,AH        ；实现AH=0，同时使CF=0
SHL AX,1         ；AX←2xAL
MOV BX,AX        ；BX←AX=2xAL
SHL AX,1         ；AX←4xAL
SHL AX,1         ；AX←8xAL
ADD AX,BX        ；AX←8xAL+2xAL=10xAL
```

- 逻辑左移一位相当于无符号数乘以2
- 逻辑右移一位相当于无符号数除以2

```
SUB AH,AH
AND AH,0
```

3. 逻辑运算和移位循环类：循环

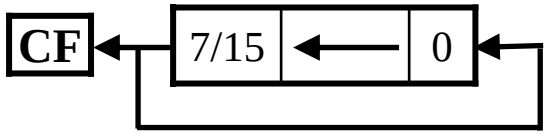
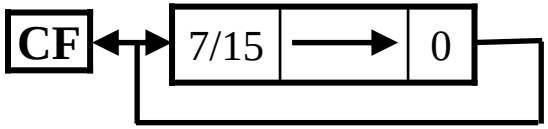
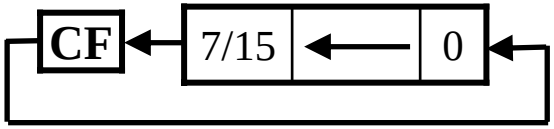
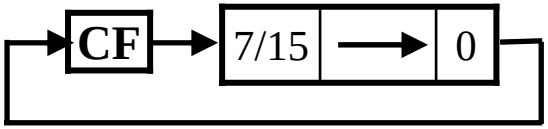
指令名称	指令书写格式(助记符)	状态标志位					
		O	S	Z	A	P	C
循环左移(字节/字)	ROL d, count	↑	.	.	x	.	↑
循环右移(字节/字)	ROR d, count	↑	.	.	x	.	↑
带进位循环左移(字节/字)	RCL d, count	↑	.	.	x	.	↑
带进位循环右移(字节/字)	RCR d, count	↑	.	.	x	.	↑

x — 表示标志位为任意值

• — 表示运算结果不影响标志位

↑ — 表示运算结果影响标志位

3. 逻辑运算和移位循环类：循环

	指令	简图	操作说明	功能
不带进位位	ROL 左移		CF不包括在循环中，CF由最高位移入。 1次移位后，最高位和CF不同，OF置1，否则OF置0。 移位次数不为1时，OF不确定。	CF的每1位可移入CF中，根据CF状态作条件转移。
	ROR 右移		CF不包括在循环中， CF由最低位移入。 1次移位后，新的最高位和原最高位不等，OF置1，否则OF置0。 移位次数不为1时，OF不确定。	
带进位位	RCL 左移		连同CF一起循环，旧的CF移入最低位。 1次移位后，最高位改变了，OF置1，否则OF置0。 移位次数不为1时，OF不确定。	移位1次，还可根据OF判断数据的符号是否改变。
	RCR 右移		连同CF一起循环，旧的CF移入最高位。 1次移位后，最高位改变了，OF置1，否则OF置0。 移位次数不为1时，OF不确定。	

3. 逻辑运算和移位循环类：循环

◆ 循环移位指令类似移位指令，但要将从一端移出的位返回到另一端形成循环。分为：

- ROL reg/mem,1/CL ;不带进位循环左移 演示
- ROR reg/mem,1/CL ;不带进位循环右移 演示
- RCL reg/mem,1/CL ;带进位循环左移 演示
- RCR reg/mem,1/CL ;带进位循环右移 演示

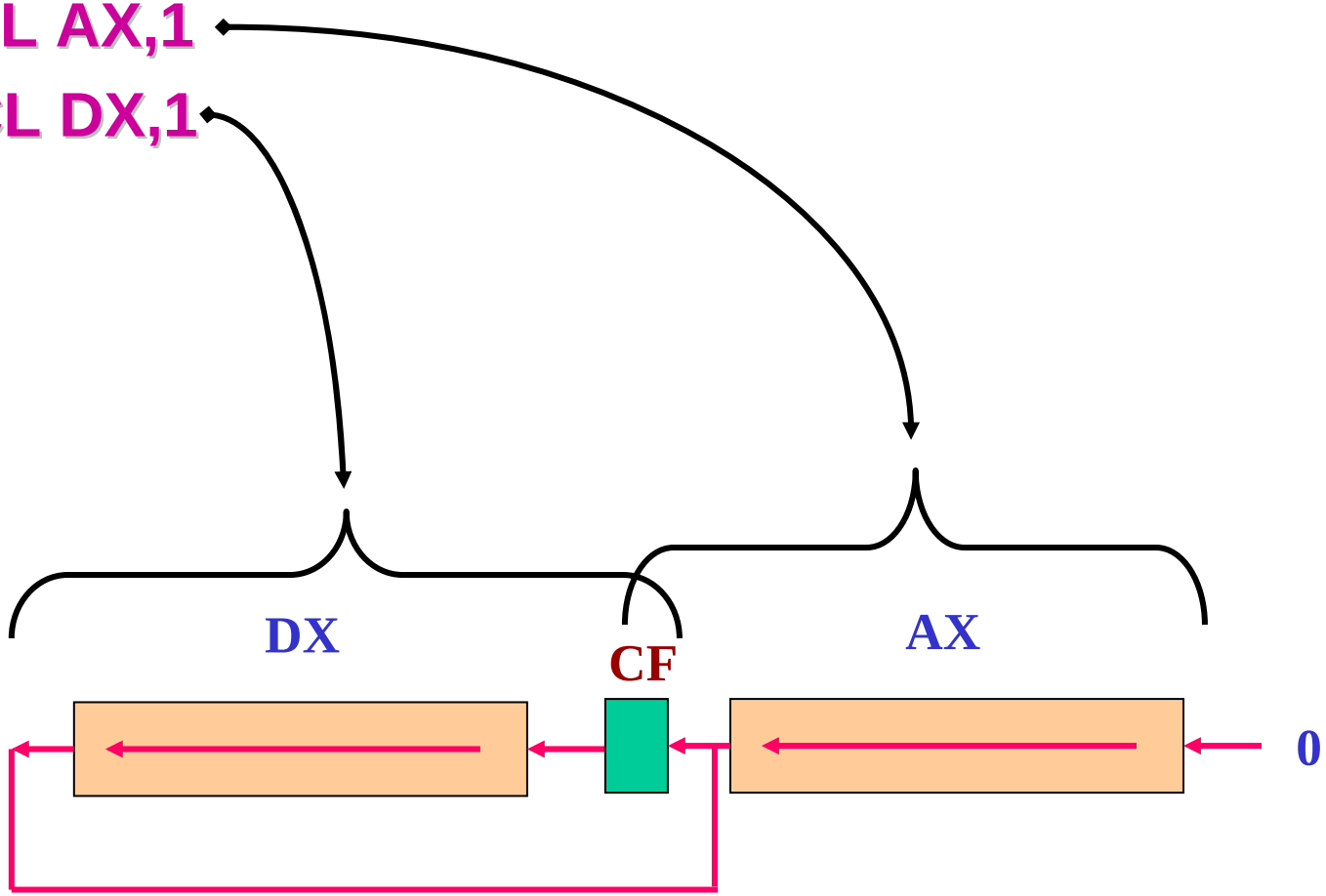
◆ 循环移位指令的操作数形式与移位指令相同，按指令功能设置进位标志CF，但不影响SF、ZF、PF、AF标志

3. 逻辑运算和移位循环类：循环

例：将DX.AX中32位数值左移一位

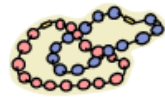
SHL AX,1

RCL DX,1



考虑32位数据的右移等操作

4. 串操作类指令



共13条，分为2小类：

- 基本字符串指令（10条）
- 重复前缀（3条）

4. 串操作类指令

数据串：位于存储器中由若干个字节或字组成的一组数据(或字符)。

每个字节或字称为数据串的元素。

字节串：元素为字节的数据串。

字串：元素为字的数据串。

4. 串操作类指令

- 是唯一的源操作数和目的操作数均在存储单元的指令。
- 可以对字节串或字串进行操作。
- 所有串操作指令都用SI对DS段中的源操作数进行间接寻址，用DI对ES段中的目的操作数进行间接寻址。
- 执行时：地址指针的修改与方向标志DF有关。

  DF=1，SI和DI作自动减量修改，

  DF=0，SI和DI作自动增量修改。

  执行前：需对SI,DI,DF进行设置，且把串长度设置在CX中。
- 在串操作指令前加前缀，可使串操作重复进行，其执行过程相当于一个循环程序的运行。

4. 串操作类指令：字符串

指令名称	指令书写格式 (助记符)	状态标志位					
		O	S	Z	A	P	C
字节串/字串传送	MOVS d, s MOVSB/MOVSW	•	•	•	•	•	•
字节串/字串比较	CMPS d, s CMPSB/CMPSW	↑	↑	↑	↑	↑	↑
字节串/字串搜索	SCAS d SCASB/SCASW	↑	↑	↑	↑	↑	↑
读字节串/字串	LODS s LODSB/LOADSW	•	•	•	•	•	•
写字节串/字串	STOS d STOSB/STOSW	•	•	•	•	•	•

↑ — 表示运算结果影响标志位

• — 表示运算结果不影响标志位

4. 串操作类指令：重复前缀

指令名称	指令书写格式（助记符）	状态标志位					
		O	S	Z	A	P	C
无条件重复	REP	•	•	•	•	•	•
当相等/为0时重复	REPE/REPZ	•	•	•	•	•	•
当不等/不为0时重复	REPNE/REPNZ	•	•	•	•	•	•

- • — 表示运算结果不影响标志位

4. 串操作类指令

✪ MOVS 目标串，源串

功能：将由SI作为指针的源串中的1个字节或字，传送到由DI作为指针的目的串中，且相应地自动修改SI/DI，使之指向下一个元素。

如果加上REP前缀，则每传送1个元素，CX自动减1，直到CX=0为止。

4. 串操作类指令

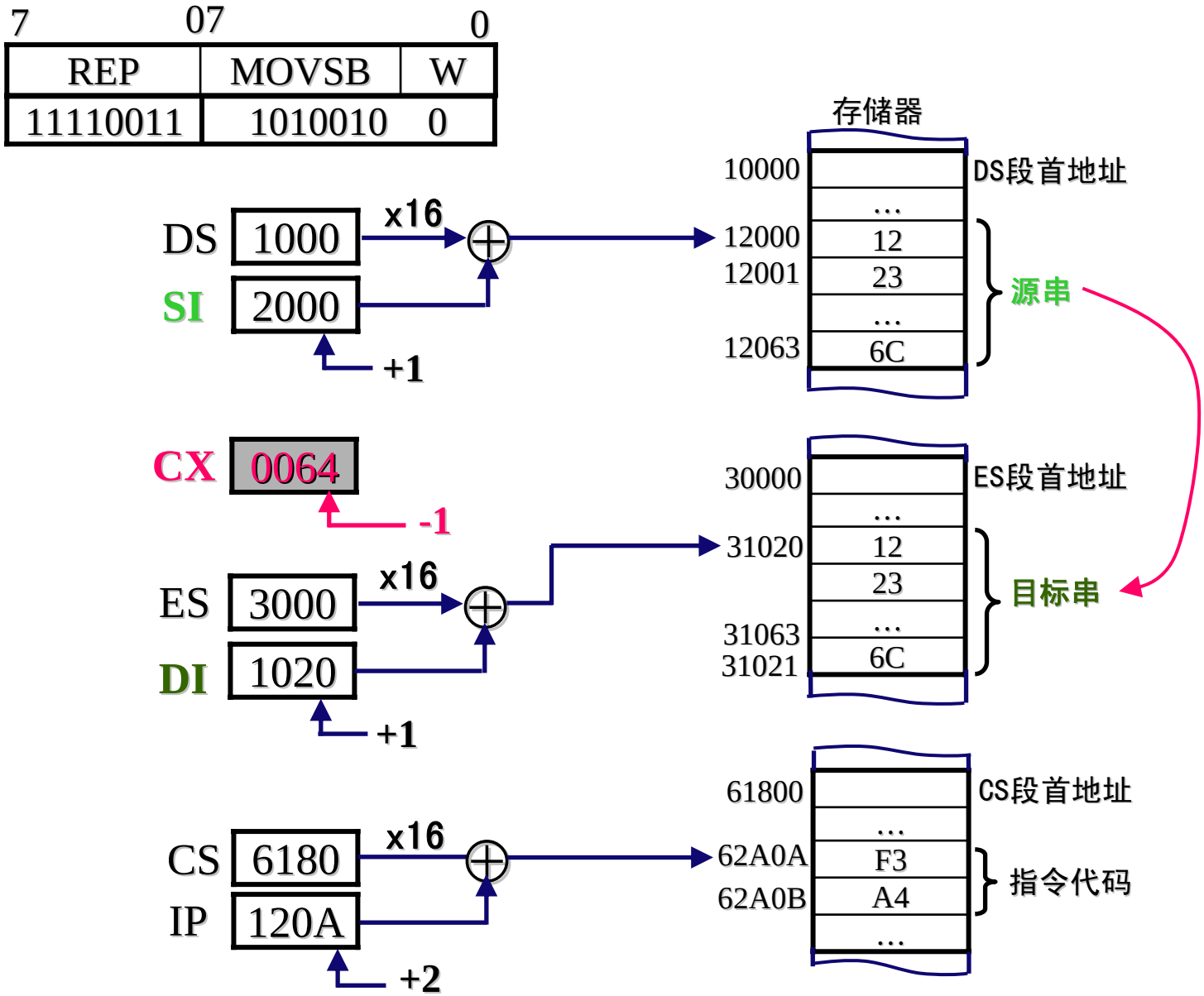
例1：REP MOVSB

设：CS=6180H, IP=120AH, DS=1000H, DI=1020H, CX=0064H, DF=0。

执行结果：

源串的100个字节传送到目标串，每传送1个字节，SI+1，DI+1，CX-1，直到CX=0为止。

4. 串操作类指令


7 07 0
REP MOVSB W
11110011 1010010 0
存储器
10000
DS段首地址
12000
12001
12063
12
23
6C
源串
DS 1000
SI 2000
x16
+1
CX 0064
-1
30000
ES段首地址
31020
31063
31021
目标串
ES 3000
DI 1020
x16
+1
61800
CS段首地址
62A0A
62A0B
F3
A4
指令代码
CS 6180
IP 120A
x16
+2

4. 串操作类指令

例2：若将源串的100个字节数据传送到目标单元中去，

设：源串首元素的偏移地址为2500H，
目标串首元素的偏移地址为1400H。

程序为：

```
CLD                ；DF置0，地址自动递增
MOV CX, 100        ；设置串长度
MOV SI,2500H       ；源串首元素的偏移地址
MOV DI,1400H       ；目标串首元素的偏移地址
REP MOVSB          ；重复传送操作，直到CX=0为止
```

4. 串操作类指令

✪ CMPS 目的串，源串

功能： 将由SI作为指针的源串中的1个元素，减去由DI作为指针的目的串中相对应的1个元素，不送回结果，只根据结果特征置标志位。

相应地自动修改SI/DI，使之指向下一个元素。

通常在CMPS指令前加重复前缀REPE/REPZ，用来确定两个串中的第1个不相同的数据。

4. 串操作类指令

例1： 比较上例中两串是否完全相同。

若两串相同，则BX内容为0；

若两串不同，则BX指向源串中第1个不相同字节的地址，且该字节的内容保留在AL中。

程序为：

```
        CLD                 ; DF置0，地址自动递增
        MOV CX, 100         ; 设置串长度
        MOV SI,2500H        ; 源串首元素的偏移地址
        MOV DI,1400H        ; 目标串首元素的偏移地址
        REP CMPSB           ; 串比较，直到ZF=0或CX=0
        JZ EQQ
        DEC SI
        MOV BX,SI           ; 第1个不相同字节的
                            ; 偏移地址送入BX
        MOV AL,[SI]         ; 第1个不相同字节的
                            ; 内容送入AL
EQQ:    JMP STOP
STOP:   MOV BX,0            ; 两串完全相同，BX=0
        HLT
```

4. 串操作类指令

✪ SCAS 目的串，源串

功能： 用来从目标串中搜索(或查找)某个关键字，

要求将待查找的关键字在执行该指令之前，
事先置入AX或AL中，取决于W=1或0。

实质： 将AX或AL中的关键字减去由DI所指向的目标串中的一个元素，不传送结果，只根据结果置标志位。
修改DI的内容指向下一个元素。

通常在SCAS指令前加重复前缀REPNE/REPNZ，用来从目标串中寻找关键字，一直进行到ZF=1(查到了某关键字)或CX=0为止。

4. 串操作类指令

例1： 在长度为N的某字符串中查找是否存在“$”字符。

若存在，则将“$”字符所在地址送入BX;否则将BX清零。

假定字符串首元素的偏移地址为DSTO。

程序为：

```
        CLD             ; DF置0，地址自动递增
        MOV CX,N        ; 串长度赋CX
        LEA DI,DSTO     ; 目标串首元素的偏移地址
                        ; 送DI
        MOV AL,'$'      ; 关键字$的ASCII码送AL
        REPNE SCASB     ; 查关键字，直至ZF=1或CX=0
        JNZ ZERQ        ; ZF=0，表示未查到
        DEC DI          ; 若已查到，则恢复关键字
                        ; 所在地址指针
        MOV BX,DI       ; 关键字所在的地址送BX
ZER:    JMP STO
STOP:   MOV BX,0        ; 未找到，BX=0
        HLT
```

4. 串操作类指令

✪ LODS 源串

功能：将源串中的SI所指向的元素，取到AX/AL中，修改SI内容，使之指向下一个元素。

LODS指令一般不加重复前缀，常用来和其它指令结合，完成复杂的串操作功能。

4. 串操作类指令

例1： 在数据段中有100个字组成的串。

现要求将其中的负数相加，其和数存放到紧接着该串的下一个顺序地址中。

已知串首元素的偏移地址为1680H。

程序为：

```
      CLD                ┐
      MOV CX,202         │
      MOV SI,1680H       ├ 初始化
      MOV BX,0           │
      MOV DX,0           ┘
LOO:  DEC CX             ;
      DEC CX             ;
      JZ STO             ; 计数是否已完？
      LODSW              ; 从源串中取一个字送AX
      MOV BX,AX          ;
      AND AX,8000H       ; 该元素是否为负数？
      JZ LOO             ; 若为正数，则重取一个字
      ADD X,BX           ; 求负数元素之和，送DX
      JMP LOO
STO:  MOV [SI],DX        ; 负数之和写入顺序地址中
      HLT
```

4. 串操作类指令

✪ STOS 目标串

功能：将AX/AL中的1个字或字节写入由DI作为指针的目标串中，同时修改DI内容，使之指向下一个元素。

STOS指令一般不加重复前缀，常用来和其它指令结合，完成复杂的串操作功能。

利用重复操作，可建立一串相同的值。

4. 串操作类指令

例1：将两串中各对应元素相加，所得到的新串写入目标串中。

若已知目标串和源串的偏移地址分别为0300H和0500H，

串长度为100字节

程序为：

```
    CLD                ┐
    MOV CX,100         │
LL: MOV SI,1680H       ├ 初始化
    MOV BX,0300H       ┘
    MOV SI,BX      ;
    LODSB          ; 从SI中读1个元素送AL
    MOV DL,AL      ; AL中元素送DL中保存
    ADD BX,0200H   ;
    MOV SI,BX      ; 确定源串的地址指针
    LODSB          ; 从源串中取1个元素送AL
    ADD AL,DL      ; 两串对应元素相加,结果送AL
    SUB BX,0200H   ;
    MOV DI,BX      ; 恢复当前目标串地址指针
    STOSB          ; AL中和数写入目标串单元
    INC BX         ; 确定下一个元素地址
    DEC CX         ;
    JNZ LL         ; CX不为零，则继续操作
    HLT
```

5. 程序控制类指令

控制转移类指令用于实现分支、循环、过程等程序结构，是仅次于传送指令的常用指令。

- 控制转移类指令通过改变IP（和CS）值，实现程序执行顺序的改变

5. 程序控制类指令

◆ 相对寻址方式 —— 用标号表达

– 指令代码中提供目的地址相对于当前IP的位移量，转移到的目的地址(转移后的IP值)就是当前IP值加上位移量。

◆ 直接寻址方式 —— 用标号表达

– 指令代码中提供目的逻辑地址，转移后的CS和IP值直接来自指令操作码后的目的地址操作数。

◆ 间接寻址方式 —— 用寄存器或存储器操作数表达

– 指令代码中指示寄存器或存储单元，目的地址从寄存器或存储单元中间接获得

5. 程序控制类指令

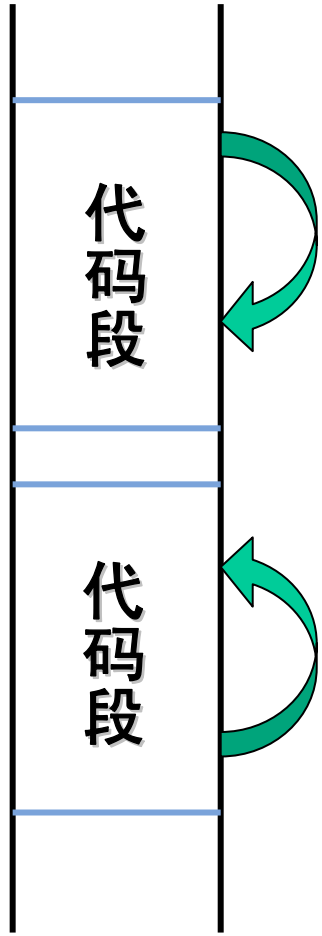
◆ **段内转移 — 近转移（near）**

– 在当前代码段64KB范围内转移（ ±32KB范围）

– 不需要更改CS段地址，只要改变IP偏移地址

◆ **段内转移 — 短转移（short）**

– 转移范围可以用一个字节表达，在段内－128～＋127范围的转移



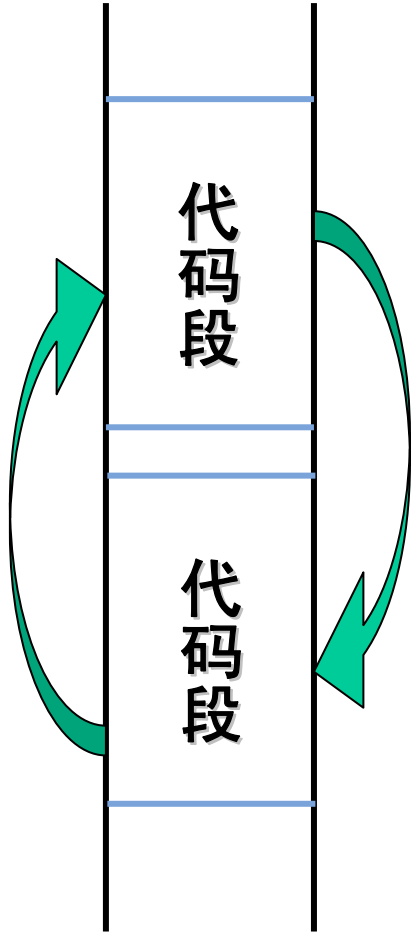
5. 程序控制类指令

◆ 段间转移 — 远转移（far）

– 从当前代码段跳转到另一个代码段，可以在1MB范围

– 需要**更改CS段地址和IP偏移地址**

– 目标地址必须用一个32位数表达，称为32位远指针，它就是逻辑地址



- 实际编程时，汇编程序会根据目标地址的属性，自动处理成短转移、近转移或远转移
- 程序员可用操作符short、near ptr 或far ptr 强制成为需要的转移类型

5. 程序控制类指令

共28条，分为4小类：

- 无条件转移（3条）
- 条件转移（18条）
  - 无符号数（4条）
  - 单标志（4条）
  - 带符号数（4条）
  - 位条件转移（6条）
- 循环控制（4条）
- 中断（3条）

5. 程序控制类指令：无条件转移

指令名称	指令书写格式 (助记符)
无条件转移	JMP 目标标号
调用过程	CALL 过程名
从过程返回	RET 弹出值

5. 程序控制类指令：无条件转移

指令名称	指令书写格式 (助记符)
无条件转移	JMP 目标标号
调用过程	CALL 过程名
从过程返回	RET 弹出值

5. 程序控制类指令：无条件转移

JMP Label　　；程序转向Label标号指定的地址

- 只要执行无条件转移指令JMP，就使程序转到指定的目标地址，从目标地址处开始执行指令。
- 操作数Label是要转移到的目标地址（目的地址、转移地址）
- **JMP指令分成4种类型：**

(1) 段内转移、相对寻址
(2) 段内转移、间接寻址
(3) 段间转移、直接寻址
(4) 段间转移、间接寻址

5. 程序控制类指令：无条件转移

✪ **JMP Label** ；段内直接转移

目标地址就在当前代码段内，其偏移地址(即目标地址的偏移量)与本指令当前IP值(即JMP指令的下一条指令的地址)之间的字节距离，即位移量在指令中直接给出。

IP←IP＋位移量

段内短转移： 位移量只有1个字节，目标标号与本指令之间的距离不超过＋127～－128字节

段内近转移： 位移量占2个字节，目标标号与本指令之间的距离不超过 ±32K字节

例如：JMP 110

5. 程序控制类指令：无条件转移

✪ **JMP R16/M16**　　；段内间接寻址

将段内的目标地址(指偏移地址或按间接寻址方式计算出的有效地址)，先存在某通用机存器或存储器的某两个连续地址中。

指令中只需给出该寄存器或存储单元地址即可。

IP←R16/M16

例如：JMP AX

JMP WORD PTR[2000]

5. 程序控制类指令：无条件转移

✪ **JMP FAR PTR Label**　　；段间直接寻址

由当前代码段转移到其它代码段，其转移范围超过±32K字节，故段间转移指令也称为远转移。

远转移时，目标标号是在其它代码段中，若指令中直接给出目标标号的段地址和偏移地址，则构成段间直接转移指令。

IP←偏移地址

CS←段地址

例如：JMP IBDF:2000

5. 程序控制类指令：无条件转移

✪ **JMP FAR PTR MEM** ；段间间接寻址

由当前代码段转移到其它代码段，其转移范围超过±32K字节，故段间转移指令也称为远转移。

将目标地址的段地址和偏移地址先存放于存储器的4个连续地址中，其中前两个字节为偏移地址，后两个字节为段地址。

指令中只需给出存放目标地址的4个连续地址首字节的偏移地址。

IP←[mem]
CS←[mem＋2]

例如：JMP FAR PTR[2000]

5. 程序控制类指令：无条件转移

✪ CALL 过程名　　；无条件调用过程指令

- 子程序是完成特定功能的一段程序
- 当主程序（调用程序）需要执行这个功能时，采用CALL调用指令转移到该子程序的起始处执行
- 当运行完子程序功能后，采用RET返回指令回到主程序继续执行

- 转移指令有去无回
- 子程序调用需要返回
  其中利用堆栈保存返回地址

5. 程序控制类指令：无条件转移

子程序： 为便于模块化设计，往往把程序中某些具有独立功能的部分编写成独立的程序。

过程： 子程序结构相当于高级语言中的过程。
调用过程即调用子程序。

近过程： 处于当前代码段的过程，用NEAR表示。
调用时，只需将当前IP值入栈。

远过程： 处于其它代码段的过程，用FAR表示。
调用时，必须将当前CS和IP值一起入栈。

5. 程序控制类指令：无条件转移

- **CALL指令分成4种类型（类似JMP）**

  CALL Label ；段内调用、直接寻址

  CALL R16/M16 ；段内调用、间接寻址

  CALL FAR PTR Label ；段间调用、直接寻址

  CALL FAR PTR MEM ；段间调用、间接寻址

- **CALL指令需要保存返回地址：**
  - 段内调用——入栈偏移地址IP

    SP←SP-2，SS:[SP]←IP
  - 段间调用——入栈偏移地址IP和段地址CS

    SP←SP-2，SS:[SP]←CS

    SP←SP-2，SS:[SP]←IP

5. 程序控制类指令：无条件转移

- CALL N_PROC　　；段内直接寻址

N_PROC是一个近过程名，将给出目标(转向)地址，即过程的入口地址。

执行时

第1步： 把过程的返回地址压入堆栈中，
以便过程返回调用程序时使用。

第2步： 转移到过程的入口地址去继续执行。

5. 程序控制类指令：无条件转移

- **CALL BX** ；段内间接寻址

事先将过程入口的偏移地址置入BX。

执行时，调用程序将转向由BX寄存器的内容所指定的某内存单元。

5. 程序控制类指令：无条件转移

- **CALL 2000H: 5600H** ；段间直接寻址

过程入口的段地址为2000H，偏移地址为5600H。

执行时，调用程序将转向物理地址为25600H所指定的某内存单元。

5. 程序控制类指令：无条件转移

- **CALL DWORD PTR [DI]** ；段间间接寻址

  事先将过程入口的偏移地址和段地址置入DI, DI+1, DI+2, DI+3 所指定的4个连续内存单元中。

5. 程序控制类指令：无条件转移

✪ RET 弹出值

过程返回(RET)指令应安排在过程的出口，即过程的最后一条指令处。

功能是从堆栈顶部弹出由CALL指令压入的断点地址值，迫使CPU返回到调用程序的断点去继续执行。

RET指令与CALL指令相呼应，CALL指令安排在调用过程中，RET指令安排在被调用的过程末尾处。并且，为了能正确返回，返回指令的类型要和调用指令的类型相对应。

在8086/8088指令系统中，段内与段间返回的指令形式是一样的，都是RET，但它们的指令代码却不同(取决于伪指令)。

5. 程序控制类指令：无条件转移

- 需要弹出CALL指令压入堆栈的返回地址
  - 段内返回——出栈偏移地址IP

    IP←SS:[SP]， SP←SP＋2
  - 段间返回——出栈偏移地址IP和段地址CS

    IP←SS:[SP]，SP←SP＋2

    CS←SS:[SP]，SP←SP＋2
- RET指令可以带立即数n，n必须为偶数。

  如 RET 4，该指令在返回地址出栈后，继续修改堆栈指针，SP←SP＋4，这一特性可用来冲掉在执行CALL指令之前推入堆栈的一些数据。

5. 程序控制类指令：条件转移

条件转移指令与无条件转移指令区别：

- ✓ 无条件转移指令出现后，一定转移至目标地址执行程序。

  条件转移指令只有当条件成立时，才转移至目标地址执行程序，否则程序顺序执行。

- ✓ 条件转移指令之前，一定要有测试条件的指令。

- ✓ 为缩短指令长度，加快转移速度，所有的条件转移指令都被设计成短转移，即转移目标与本指令之间的字节距离在-128～+127范围以内。

  如果遇到超出了短转移所能转移的范围时，可通过两条转移指令来实现转移。

5. 程序控制类指令：条件转移

- 条件转移指令Jcc根据指定的条件确定程序是否发生转移。其通用格式为：

```
Jcc  label      ；条件满足，发生转移
                ；IP←IP＋8位位移量；
                ；否则，顺序执行
```

- label是一个标号、一个8位位移量，表示Jcc指令后的那条指令的偏移地址，到目标指令的偏移地址的地址位移。
- Jcc指令不影响标志，但要利用标志。

5. 程序控制类指令：条件转移

根据利用的标志位不同，分成三种情况：

- **比较无符号数高低（）**

  高于　（Above，简写A）
  不低于（Not Below，简写NB）
  低于　（Below，简写B）
  不高于（Not Above，简写NA）

- **比较有符号数大小**

  大于　（Greater，简写G）
  不小于（Not Less，简写NL）
  小于　（Less，简写L）
  不大于（Not Greater，简写NG）

- **判断单个标志位状态**

5. 程序控制类指令： 条件转移

	指令名称	助记符	测试条件
无符号数	高于/不低于也不等于 转移	JA/JNBE 目标标号	CF=0 AND ZF=0
	高于或等于/不低于 转移	JAE/JNB 目标标号	CF=0 OR ZF=1
	低于/不高于也不等于 转移	JB/JNAE 目标标号	CF=1 AND ZF=0
	低于或等于/不高于 转移	JBE/JNA 目标标号	CF=1 OR ZF=1
带符号数	大于/不小于也不等于 转移	JG/JNLE 目标标号	(SF XOR OF) AND ZF=0
	大于或等于/不小于 转移	JGE/JNL 目标标号	SF XOR OF=0 OR ZF=1
	小于/不大于也不等于 转移	JL/JNGE 目标标号	SF XOR OF=0 AND ZF=0
	小于或等于/不大于 转移	JLE/JNG 目标标号	(SF XOR OF) OR ZF=1
单标志	等于/结果为0 转移	JE/JZ 目标标号	ZF=1
	不等于/结果不为0 转移	JNE/JNZ 目标标号	ZF=0
	有进位/有借位 转移	JC 目标标号	CF=1
	无进位/无借位 转移	JNC 目标标号	CF=0
位条件转移	溢出 转移	JO 目标标号	OF=1
	不溢出 转移	JNO 目标标号	OF=0
	奇偶性为1/偶状态 转移	JP/JPE 目标标号	PF=1
	奇偶性为0/奇状态 转移	JNP/JPO 目标标号	PF=0
	符号位为1 转移	JS 目标标号	SF=1
	符号为为0 转移	JNS 目标标号	SF=0

5. 程序控制类指令： 条件转移

- Jcc指令实际虽然只有18条，但却有30个助记符
- 采用多个助记符，目的是为了方便记忆和使用

5. 程序控制类指令：条件转移

例1：将AX中的无符号数除2，如果是奇数，则加1后除以2。

问题：如何判断AX中的数据是奇数还是偶数？

方法：判断AX最低位是“0”（偶数），还是“1”（奇数）。
可以用位操作类指令

1. 用逻辑与指令，将除最低位外的其它位变成0，保留最低位不变。判断这个数据是0，AX就是偶数；否则，为奇数。

2. 将最低位用移位指令移至进位标志，判断进位标志是0，AX就是偶数；否则，为奇数。

3. 将最低位用移位指令移至最高位（符号位），判断符号标志是0，AX就是偶数；否则，为奇数。

5. 程序控制类指令：条件转移

方法1：用JZ指令实现

```
        TEST AX,01H
；测试AX的最低位D0（不用AND指令，以免改变AX）
        JZ even
；标志ZF＝1，即D0＝0：AX内是偶数，程序转移
        ADD AX,1
；标志ZF＝0，即D0＝1：AX内的奇数，加1
even:  SHR AX,1          ；AX←AX÷2
```

用右移一位的方法实现除以2。
本例中用RCR指令比SHR指令更好。

5. 程序控制类指令：条件转移

方法2：用JNC指令实现

还可用SAR、ROR和RCR指令

```
        MOV BX, AX
        SHR BX, 1
；将AX的最低位D0移进CF

        JNC even
；标志CF＝0，即D0＝0：AX内是偶数，程序转移

        ADD AX, 1
；标志CF＝1，即D0＝1：AX内的奇数，加1

even:   SHR AX, 1          ；AX←AX÷2
```

5. 程序控制类指令：条件转移

方法3：用JNS指令实现

错误！循环指令不影响SF等标志

```
        MOV BX, AX
        ROR BX, 1
；将AX的最低位D0移进最高位（符号位SF）
        JNS even
；标志SF＝0，即D0＝0：AX内是偶数，程序转移
        ADD AX, 1
；标志SF＝1，即D0＝1：AX内的奇数，加1
even:   SHR AX, 1          ；AX←AX÷2
```

ADD BX,0 ；增加一条指令

5. 程序控制类指令：条件转移

例2：寄存器AL中是字母Y（含大小写），则令AH＝0，否则令AH= -1。

```
        CMP AL, 'y'      ; 比较AL与小写字母y
        JE next          ; 相等，转移
        CMP AL, 'Y'      ; 不相等，
                         ; 继续比较AL与大写字母Y
        JE next          ; 相等，转移
        MOV AH, -1       ; 不相等，令AH＝－1
        JMP done         ; 无条件转移指令
next:   MOV AH, 0        ; 相等的处理：令AH＝0
done:   ……
```

5. 程序控制类指令：条件转移

例3：对DL寄存器中8位数据进行偶校验；校验位存入CF标志。

```
        TEST DL, 0FFH
              ；使CF＝0，同时设置PF标志
        JPE done
              ；DL中“1”的个数为偶数
              ；正好CF＝0，转向done
        STC
              ；DL中“1”的个数为奇数，设置CF＝1
done:   ……
              ；完成
```

5. 程序控制类指令：条件转移

例4：比较AX和BX中的数，并将较大者存入wmax。

方法1：

```
        CMP AX, BX      ；比较AX和BX
        JAE next        ；若AX≥BX，转移
        XCHG AX, BX     ；若AX＜BX，交换
next:   MOV wmax, AX
```

如果AX和BX存放的是有符号数，
则条件转移指令应采用JGE指令

5. 程序控制类指令：条件转移

例4：比较AX和BX中的数，并将较大者存入wmax。

方法2：

```
        CMP AX, BX        ；比较AX和BX
        JAE next
        MOV WMAX, BX      ；若AX＜BX，wmax←BX
        JMP done
next:   MOV WMAX, AX      ；若AX≥BX，WMAX←AX
done:   ……
```

5. 程序控制类指令：条件转移

例4：比较AX和BX中的数，并将较大者存入wmax。

方法3：

```
        CMP AX, BX      ；比较AX和BX
        JBE next
        MOV wmax, AX    ；若AX＞BX，wmax←AX
        JMP done
next:   MOV wmax, BX    ；若AX≤BX，wmax←BX
done:   ……
```

5. 程序控制类指令： 循环控制

指令名称	指令书写格式 (助记符)	
循环	LOOP	目标标号
相等/结果为0时循环	LOOPE/LOOPZ	目标标号
不等/结果不为0时循环	LOOPNE/LOOPNZ	目标标号
CX=0时转移	JCXZ	目标标号

5. 程序控制类指令：循环控制

✪ LOOP label

功能：先将CX寄存器内容减1后送回CX，再判断CX是否为0，若CX≠0，则转移到目标标号所给定的地址继续循环，否则，结束循环顺序执行下一条指令。

这是一条常用的循环控制指令，使用LOOP指令前，应将循环次数送入CX寄存器。其操作过程与条件转移指令类似，只是它的位移量应为负值。

```
DEC CX
JNZ label
```

5. 程序控制类指令：循环控制

✪ LOOPE/LOOPZ 目标标号

LOOPE和LOOPZ是同一条指令的两种不同的助记符。

功能： 是先将CX减1送CX，若ZF=1且CX≠0时则循环，否则顺序执行下一条指令。

5. 程序控制类指令：循环控制

✪ LOOPNE/LOOPNZ 目标标号

LOOPNE和LOOPNZ也是同一条指令的两种不同的助记符。

功能： 是先将CX减1送CX，若ZF=0且CX≠0时则循环，否则顺序执行下一条指令。

5. 程序控制类指令：循环控制

✪ JCXZ 目标标号

JCXZ指令不对CX寄存器内容进行操作，只根据CX内容控制转移。

它既是一条条件转移指令，也可用来控制循环，但循环控制条件与LOOP指令相反。

循环控制指令在使用时放在循环程序的开头或结尾处，以控制循环程序的运行。

5. 程序控制类指令：循环控制

例：若在存储器的数据段中有100个字节构成的数组，要求从该数组中找出“$”字符，然后将“$”字符前面的所有元素相加，结果保留在AL寄存器中。

```
        MOV CX, 100          ; 初始化
        MOV SI, 00FFH
LL1:    INC SI               ; 找"$"字符
        CMP BYTE PTR [SI], '$'
        LOOPE LL1
        SUB SI, 0100H        ; "$"字符之前字节数
        MOV CX, SI
        MOV SI, 0100H
        MOV AL, [SI]
        DEC CX               ; 相加次数
LL2:    INC SI               ; 累加"$"字符前的字节
        ADD AL,[SI]
        LOOP LL2
        HLT
```

5. 程序控制类指令：中断指令

指令名称	指令书写格式 (助记符)
中断	INT　　中断类型码
溢出中断	INTO
中断返回	IRET

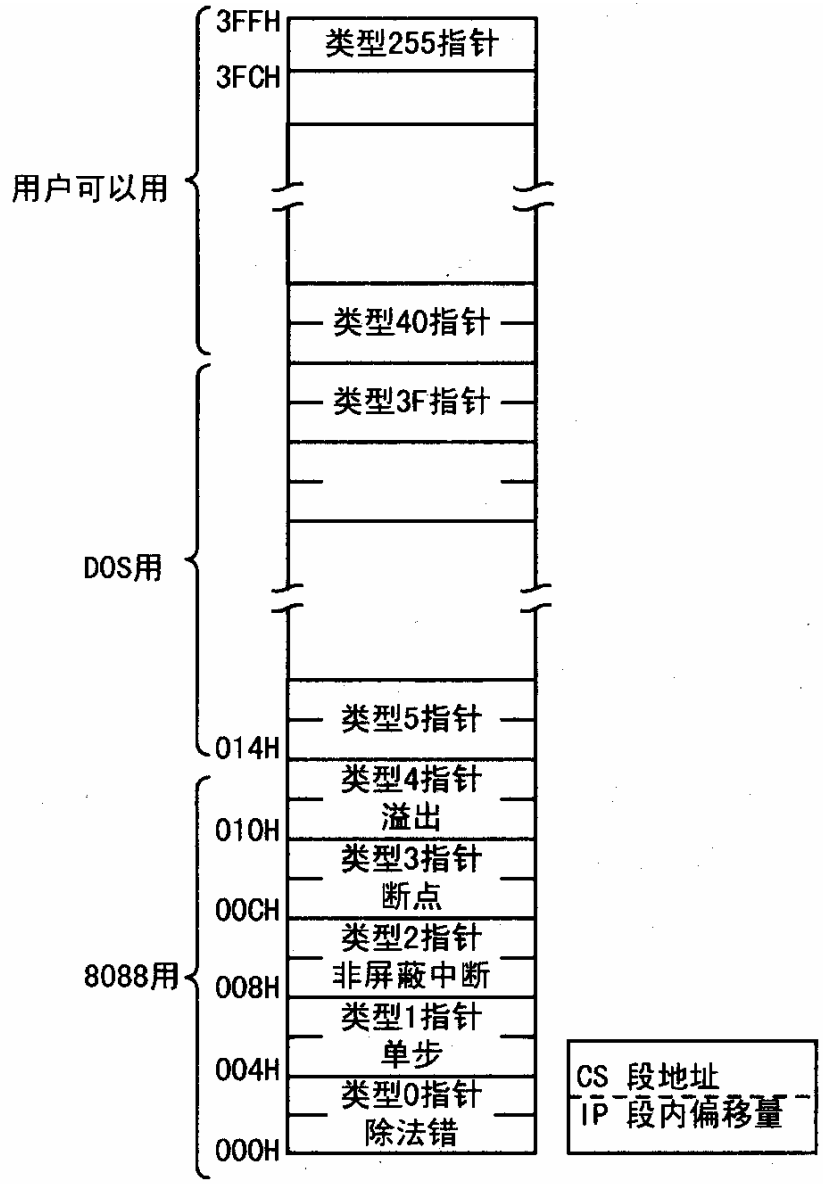
5. 程序控制类指令：中断指令

✪ INT 中断类型n

8086/8088系统中允许有256种中断类型(0~255)，CPU根据中断类型号，从内存实际地址为00000H~003FFH区中的中断向量表找到中断服务程序的入口地址。

每个类型号含4字节的中断向量，中断向量就是中断服务程序的入口地址。

中断类型n*4，就得到中断向量的存放地址，由此地址开始，前2个单元中存放中断服务程序入口地址的偏移量(即IP)，后2个单元中存放着中断服务程序入口地址的段首址(即CS)。



5. 程序控制类指令：中断指令

CPU执行INT指令过程：

① (SP)←(SP)-2，将标志寄存器F内容入栈；

② 清除中断标志IF和单步标志TF，以禁止可屏蔽中断和单步中断进入；

③ (SP)←(SP)-2，将当前程序断点的段地址入栈保护；

④ (SP)←(SP)-2，将当前程序断点的偏移地址入栈保护；

⑤ n*4，从中断入口地址表中获得中断入口的段地址和偏移地址，分别置入段寄存器CS和指令指针IP中，CPU将转向中断入口去执行相应的中断服务程序。

5. 程序控制类指令：中断指令

✪ INTO

为了判断有符号数的加减运算是否产生溢出，专门设计了1个字节的INTO指令，用于对溢出标志OF进行测试；

当OF=1，立即向CPU发出溢出中断请求，并根据系统对溢出中断类型的定义（类型4），可从中断入口地址表中得到类型4的中断服务程序入口地址。

该指令一般安排在带符号的算术运算指令之后，用于处理溢出中断。

5. 程序控制类指令：中断指令

✪ IRET

IRET指令总是安排在中断服务程序的出口处。

由它控制从堆栈中弹出程序断点送回CS和IP中，弹出标志寄存器内容送回F中，迫使CPU返回到断点继续执行后续程序。IRET也是一条1字节指令。

6. 处理器控制类指令

共12条，分为3小类：

- 对标志位操作（7条）
- 同步控制（3条）
- 其它（2条）

6. 处理器控制类指令：对标志位操作

指令名称	指令书写格式 (助记符)
清除进位标志	CLC
置“1”进位标志	STC
取反进位标志	CMC
清除方向标志	CLD
置“1”方向标志	STD
清除中断标志	CLI
置“1”中断标志	STI

6. 处理器控制类指令：对标志位操作

① CLC、STC、CMC指令用来对进位标志CF清“0”、置“1”、取反操作。

② CLD、STD指令用来将方向标志DF清“0”、置“1”。
常用于串操作指令之前。

③ CLI、STI指令用来将中断标志IF清“0”、置“1”。
当CPU需要禁止可屏蔽中断进 入时，应将IF清“0”，允许可屏蔽中断进入时，应将IF置“1”。

6. 处理器控制类指令：同步控制

指令名称	指令书写格式 (助记符)
等待 交权 封锁总线	WAIT ESC LOCK

6. 处理器控制类指令：同步控制

8086/8088 CPU工作于最大方式时，可与其它处理器一起构成多处理器系统，当CPU需要协处理器帮助它完成某个任务时，CPU可用同步指令向协处理器发出请求，待它们接受这一请求，CPU才能继续执行程序。

为此，专门设置了3条同步控制指令。

6. 处理器控制类指令：同步控制

✪ ESC 外部操作码，源操作数

ESC（Escape）指令是在最大方式系统中CPU要求协处理器完成某种任务的命令。

功能：使某个协处理器可以从CPU的程序中取得一条指令或一个存储器操作数。

ESC指令与WAIT指令、TEST引脚信号结合使用时，能够启动一个在某个协处理器中执行的子程序。

6. 处理器控制类指令：同步控制

✪ WAIT

WAIT指令通常用在CPU执行完ESC指令后，用来等待外部事件，即等待TEST线上的有效信号。

当TEST=1时，表示CPU正处于等待状态，并继续执行WAIT指令，每隔5个时钟周期就测试一次TEST状态；一旦测试到TEST=0，则CPU结束WAIT指令，继续执行后续指令。

WAIT与ESC两条指令是成对使用的，它们之间可以插入一段程序，也可以相连。

6. 处理器控制类指令：同步控制

✪ LOCK

LOCK是一字节的指令前缀，而不是一条独立的指令，常作为指令的前缀可位于任何指令的前端。

凡带有LOCK前缀的指令，在该指令执行过程中都禁止其他协处理器占用总线，故它可称为总线锁定前缀。

6. 处理器控制类指令：其它

指令名称	指令书写格式 (助记符)
暂停 空操作	HLT NOP

6. 处理器控制类指令：其它

✪ HLT

HLT是暂停指令，用于迫使CPU暂停执行程序。

执行HLT指令时，实际上是用软件方法使CPU处于暂停状态等待硬件中断。

此时，CS和IP指向HLT后面的一条指令的地址；如果有一个外部硬件中断产生，只要IF为1，CPU便可用2个总线周期响应中断。

在执行完中断处理程序而中断返回以后，CPU接着执行HLT后面的一条指令。

此外，对系统进行复位操作，也会使CPU退出暂停状态。

6. 处理器控制类指令：其它

✪ NOP

NOP是一条空操作指令，它并未使CPU完成任何有效功能，只是每执行一次该指令要占用3个时钟周期的时间。

常用来作延时，或取代其他指令作调试之用。


学有所成
这是你收获的季节
丰收去


第三章第2部分结束！