



汇编语言与微机原理

熊迎军

xyj@njau.edu.cn

13776655525

第三章 指令系统

3.1 指令与指令系统

3.2 IA-32指令系统

3.3 本章小结

3.1 指令与指令系统

- **指令**

  控制计算机完成某种操作的命令

- **指令系统**

  处理器所能识别的所有指令的集合

- **指令的兼容性：**

  同一系列机的指令都是兼容的。

- 16位8086指令系统是Intel 80x86系列微处理器指令系统的基础

指令长度

- 零操作数指令：操作码
- 单操作数指令：操作码 操作数
- 双操作数指令：操作码 操作数，操作数
- 多操作数指令：三操作数及以上

指令的执行速度

- 指令的字长影响指令的执行速度
- 对不同的操作数，指令执行的时间不同：

存储器 → 立即数 → 寄存器

→

快！

3.2 IA-32指令系统

按功能划分为六大类：

- 数据传送
- 算术运算
- 逻辑运算和移位
- 串操作
- 程序控制
- 处理器控制

Intel 8086指令系统共有117条基本指令

3.2.1 数据传送类指令

- 通用数据传送
- 输入输出
- 地址传送
- 标志位操作

3.2.1.1 通用数据传送类指令

- 一般数据传送指令
- 堆栈操作指令
- 交换指令
- 查表转换指令
- 字位扩展指令

- 特点：
  - **该类指令的执行对标志位不产生影响**

（1）一般数据传送指令

- 一般数据传送指令： **MOV**
- 格式：

  MOV dest，src
- 操作：

  src → dest

  例：

  MOV AL，BL

注意点：

- 两操作数字长必须相同；
- 两操作数不允许同时为存储器操作数；两操作数不允许同时为段寄存器；在源操作数是立即数时，目标操作数不能是段寄存器；
- IP和CS不作为目标操作数，FLAGS一般也不作为操作数在指令中出现。

如果要将数据送入段寄存器(DS、ES、SS)，必须先将这个数送入一个非段寄存器，然后在送到段寄存器。

```
MOV   CS,3000H     ; 不正确
MOV   AX,3000H     ;
MOV   CS,AX        ; 不正确

MOV   SS,1000H     ; 不正确
MOV   DI,1000H
MOV   SS,DI        ; 正确
```

➢ 例1

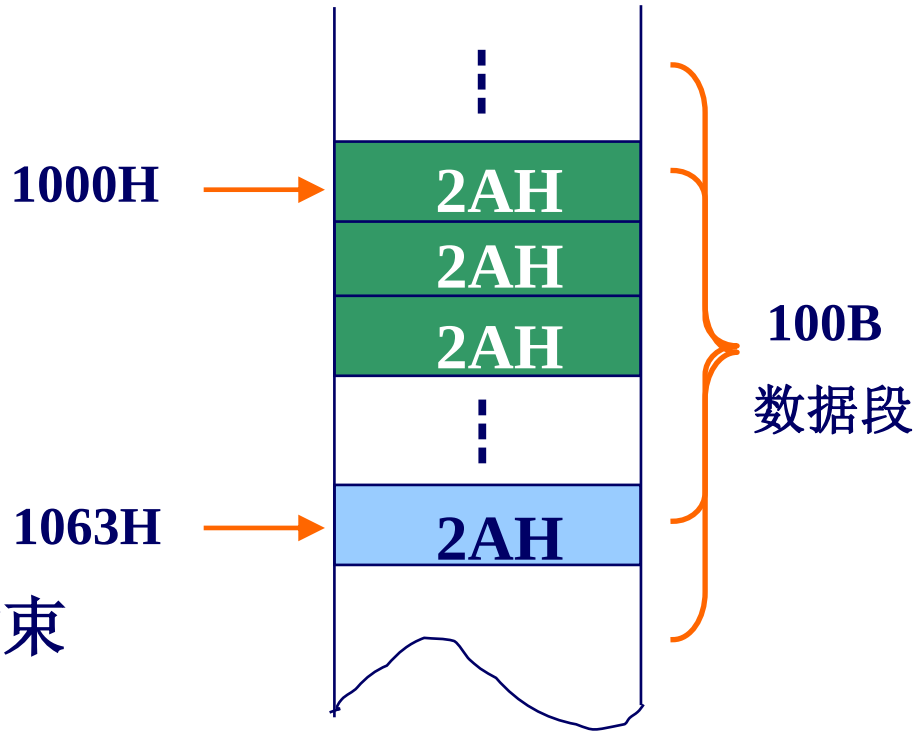
判断下列指令的正确性：

- MOV　AL，BX；错误，操作数类型不同
- MOV　AX，[SI]05H；正确
- MOV　[BX][BP]，BX；错误，不能将两个基址寄存器组合一起寻址
- MOV　DS，1000H；错误，立即数不能给段寄存器赋值
- MOV　[1200]，[SI]；错误，不能用1条指令完成两个存储单元之间的数据传送

- 例2

将立即数2AH送入内存数据段1000H开始的100个单元中。

- 题目分析：
  - 确定首地址
  - 确定数据长度
  - 写一次数据
  - 修改单元地址
  - 修改长度值
  - 判断写完否？
  - 未完继续写入，否则结束



程序段：

```
        MOV   DI，1000H
        MOV   CX，64H
        MOV   AL，2AH
AGAIN：MOV   [DI]，AL
        INC   DI                ；DI+1
        DEC   CX                ；CX-1
        JNZ   AGAIN             ；CX≠0则继续
HLT
```

- 设CS=109EH，IP=0100H，则各条指令在代码段中的存放地址如下：

CS ： IP	机器指令	汇编指令
109E：0100	B80010	MOV DI，1000H
109E：0103	.	MOV CX，64H
109E：0105	.	MOV AL，2AH
109E：0107	.	MOV [DI]，AL
109E：0109		INC DI
109E：010A		DEC CX
109E：010B		JNZ 0107H
109E：010D		HLT

➢ 数据段中的分布

送上2AH后数据段中相应存储单元的内容改变如下：

DS：1000　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A
DS：1010　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A
DS：1020　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A
DS：1030　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A
DS：1040　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A
DS：1050　2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A
DS：1060　2A 2A 2A 2A 00 00 00 00 00 00 00 00 00 00 00 00

偏移地址[DI]

（2）堆栈操作指令

- 压栈指令 PUSH

  格式： PUSH OPRD

- 出栈指令 POP

  格式： POP OPRD

OPRD只能是16位寄存器或存储器操作数（两单元）

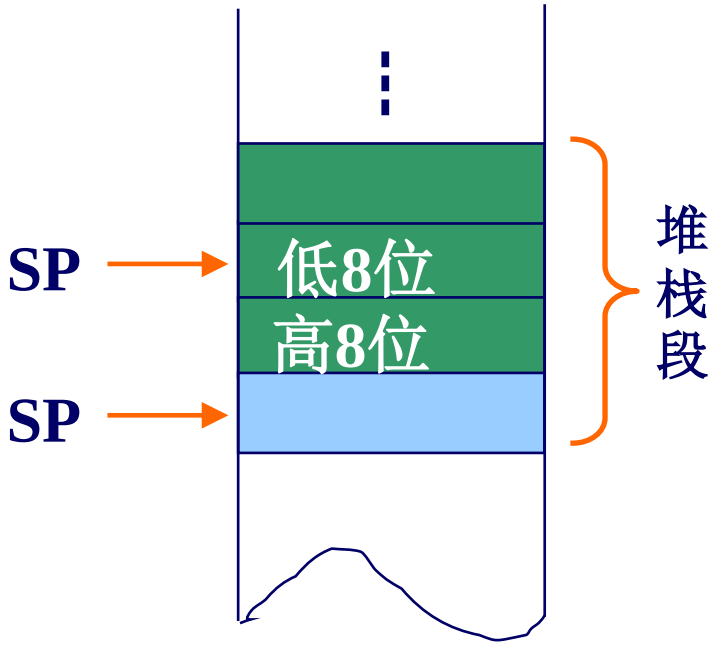
➢ 压栈指令 PUSH

- 指令执行过程：

SP － 2 → SP

操作数高字节 → SP+1

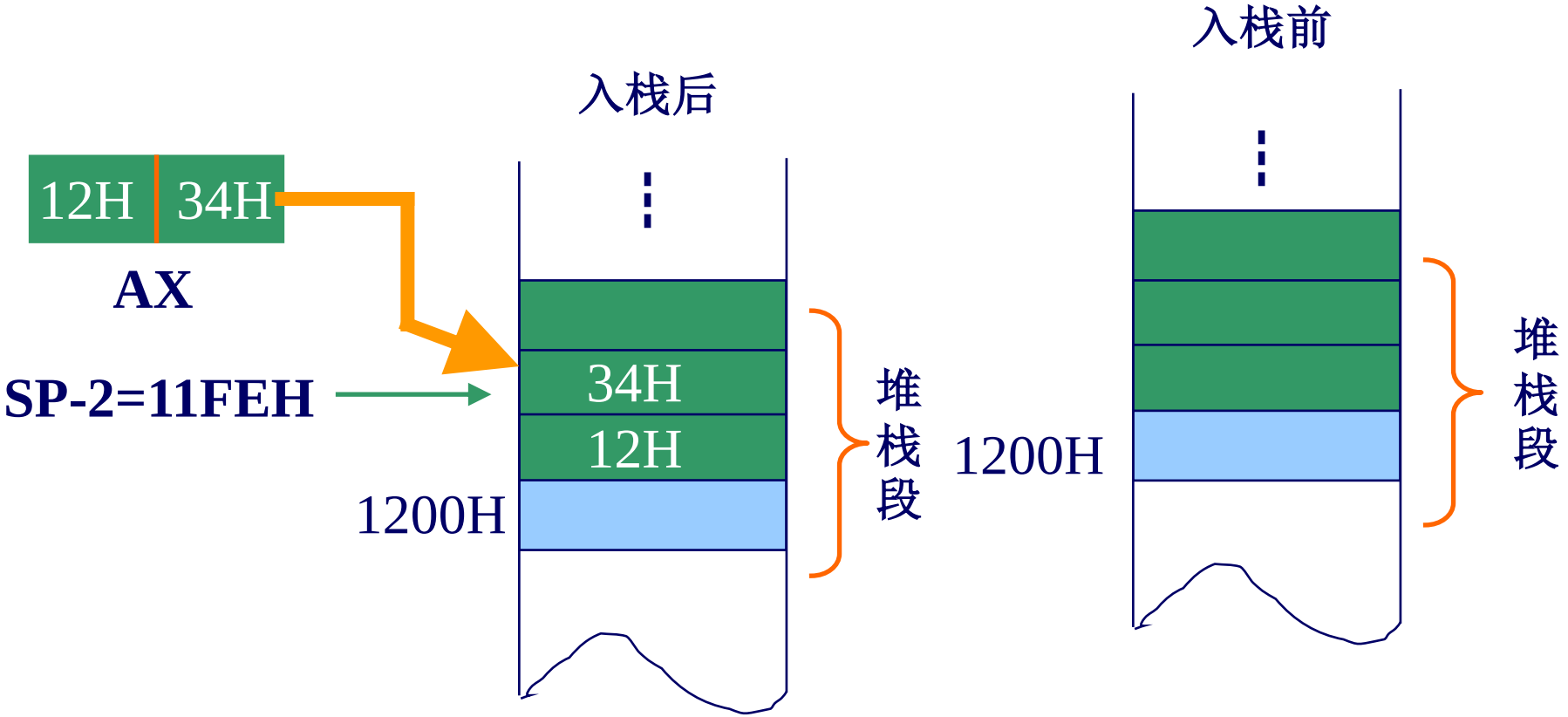
操作数低字节 → SP



➢ 压栈指令的操作

设AX=1234H，SP=1200H

执行 PUSH AX 指令后堆栈区的状态：



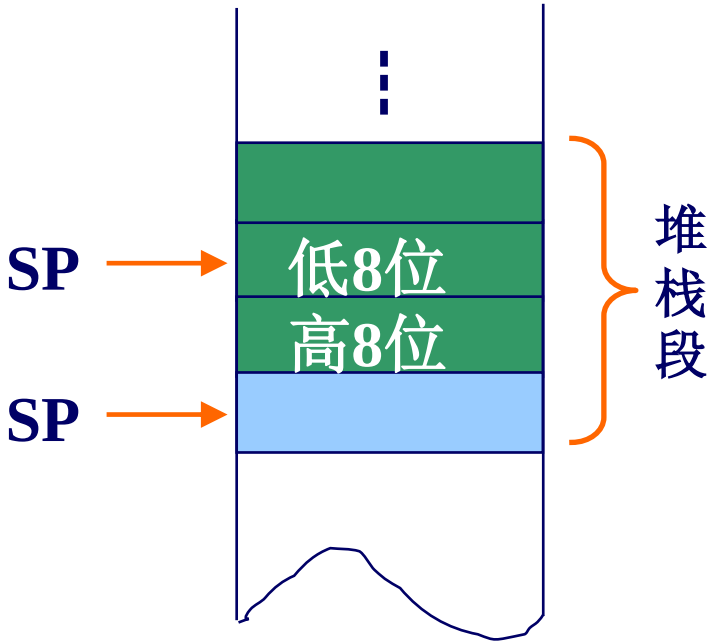
➢ 出栈指令POP

- 指令执行过程：

SP → 操作数低字节

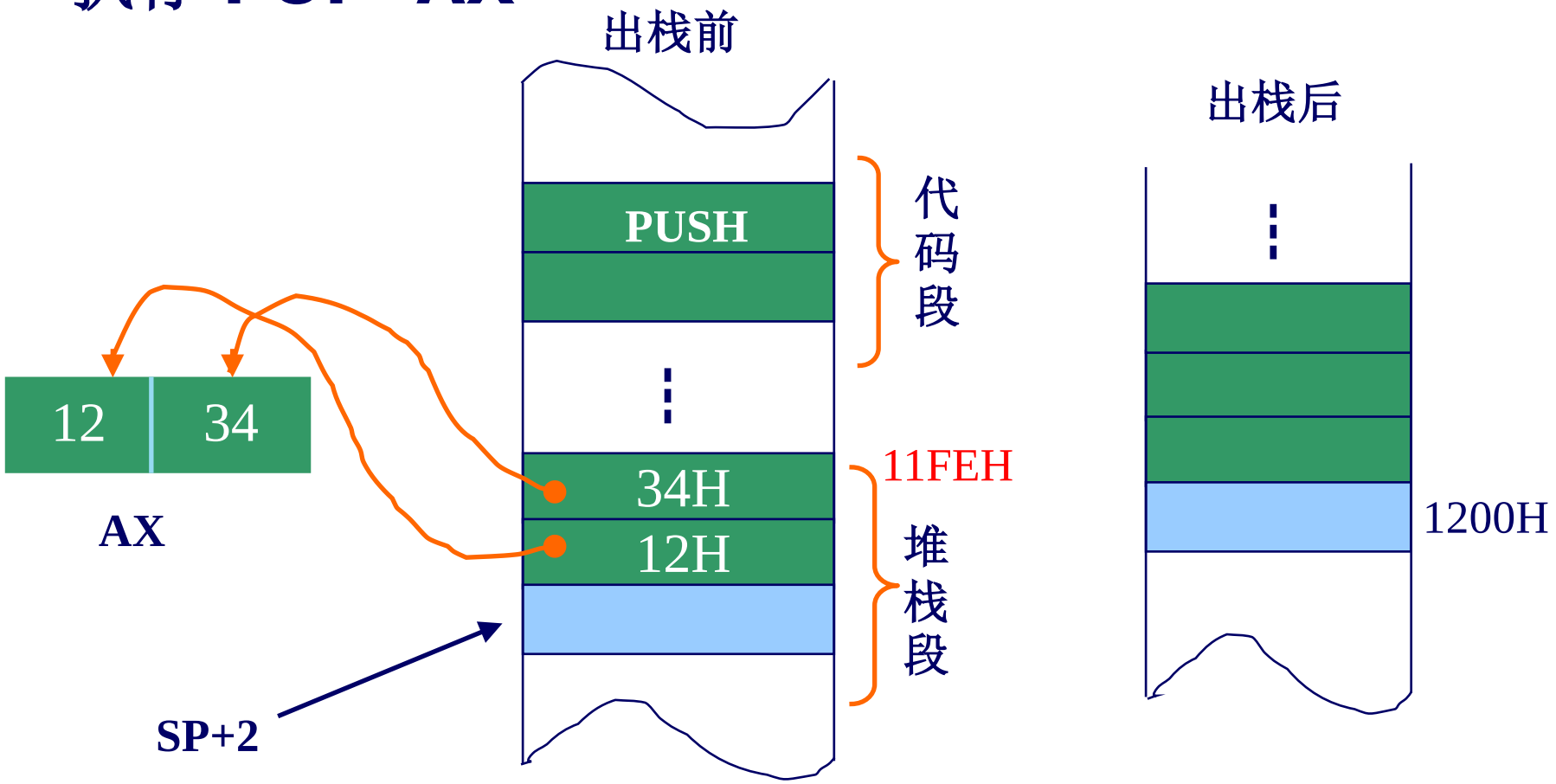
SP+1 → 操作数高字节

SP ← SP+2



➢ 出栈指令的操作

执行 POP AX



➢ 堆栈操作指令说明

- 指令的操作数必须是16位的；
- 操作数可以是寄存器或存储器两单元，但不能是立即数；
- 不能从栈顶弹出一个字给CS（允许PUSH CS）；
- PUSH和POP指令在程序中一般成对出现；
- PUSH指令的操作方向是从高地址向低地址，而POP指令的操作正好相反。

➢ 例：

```
MOV   AX，9000H
MOV   SS，AX
MOV   SP，E200H
MOV   DX，38FFH
PUSH   DX
PUSH   AX
        ┆
        ┆
        ┆
POP    DX
POP    AX
```

如此，会使**DX**和**AX**的内容互换

（3）交换指令

- 格式：

  XCHG　MEM/REG，MEM/REG

- 注：
  - 两操作数必须有一个是寄存器操作数
  - 不允许使用段寄存器
  - 两个操作数字长要一致。

- 例：

```
XCHG      AX，BX
XCHG      [2000H]，CL
```

（4）查表指令

- **格式：**

  XLAT；（AL）←（（BX）+（AL））

  或：XLAT src_table ；

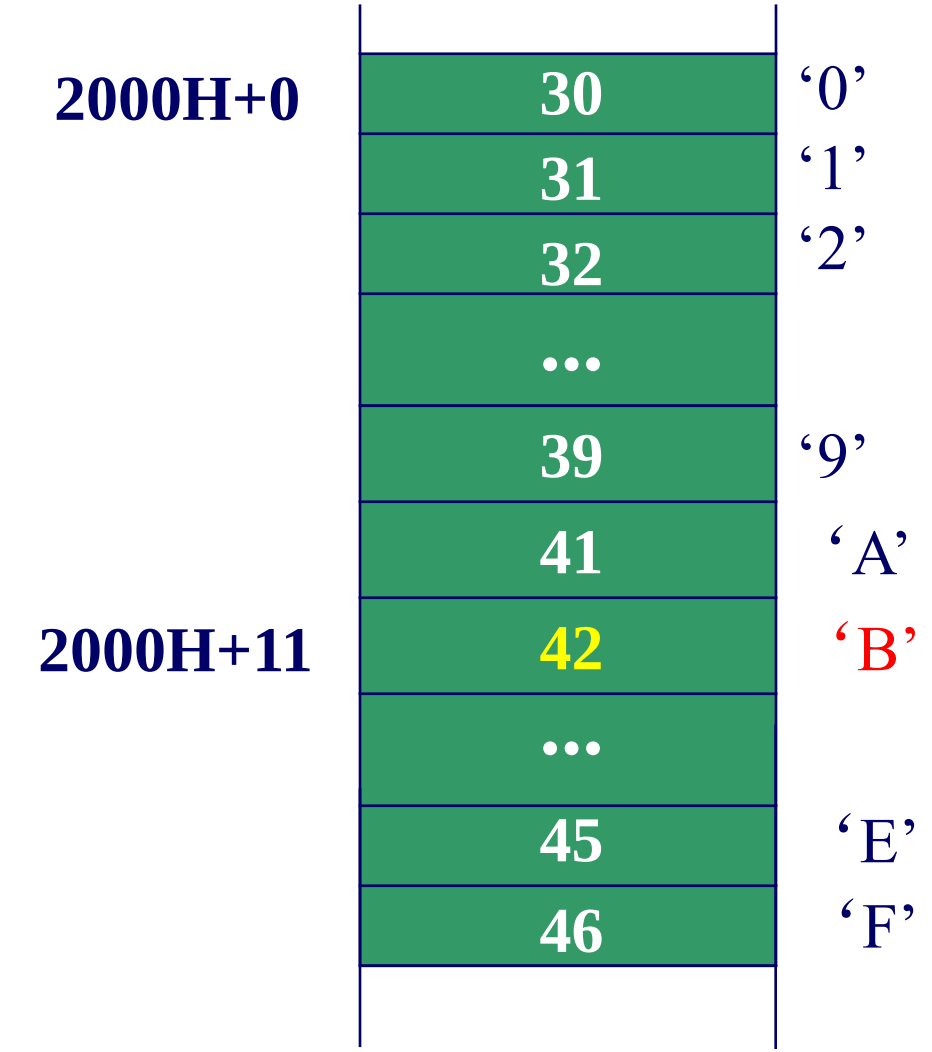
  （src_table表示要查找的表的首地址）

- **说明：**
  - 用BX的内容代表表格首地址，AL内容为表内位移量，BX+AL得到要查找元素的偏移地址
  - 表格长度最大不能超过256个字节
- **操作：**

  将BX+AL所指单元的内容送AL

➢ 例：

数据段中存放有一张ASCII码转换表，设首地址为2000H，现欲查出表中第11个代码的ASCII码



- 可用如下指令实现：

```
MOV   BX，2000H          ；BX←表首地址
MOV   AL，0BH            ；AL←序号
XLAT                     ；查表转换
```

- 执行后：AL = 42H
- 还可用其他方法实现，如：

```
MOV   BX，2000H
MOV   AL，[BX+0BH]
```

（5）字位扩展指令

- 将有符号数的符号位扩展到高位；
- 指令为零操作数指令，采用隐含寻址，隐含的操作数为AX及AX，DX
- 无符号数的扩展规则为在高位补0

➢ 字节到字的扩展指令

- 格式：
  - ⑩ CBW
- 操作：
  - ⑩ 将AL内容扩展到AX
- 规则：
  - 若最高位=1，则执行后AH=FFH
  - 若最高位=0，则执行后AH=00H

➢ 字到双字的扩展指令

- 格式：
  - CWD
- 操作：
  - 将AX内容扩展到DX AX，高位存放在DX中，隐含了操作数AX。
- 规则：
  - 若最高位=1，则执行后DX=FFFFH
  - 若最高位=0，则执行后DX=0000H

- 例：

判断以下指令执行结果：

```
MOV   AL，44H
CBW

MOV   AX，0AFDEH
CWD

MOV   AL，86H
CBW
```

3.4.1.2 输入输出指令

掌握：

- 指令的格式及操作
- 指令的两种寻址方式
- 指令对操作数的要求
- 专门面向I/O端口操作的指令

- 指令格式：
  - 输入指令： **IN acc，PORT**
  - 输出指令 ：**OUT PORT，acc**

端口地址

- 根据端口地址码的长度，指令具有两种不同的端口地址表现形式。

- 直接地址
  - 端口地址为8位时，指令中直接给出8位端口地址（立即数方式）；
  - 寻址256个端口。
- 间接地址
  - 端口地址为16位时，指令中的端口地址必须由DX指定；
  - 可寻址64K个端口。

- 例：

```
IN   AX，80H
MOV DX，2400H
IN   AL，DX
OUT   35H ，AX
OUT   AX，35H
```

3.4.1.3 地址传送指令

- 取偏移地址指令LEA
- *LDS指令
- *LES指令

（1）取偏移地址指令LEA

- 操作：
  将变量的16位偏移地址取出送目标寄存器
- 当程序中用符号地址表示内存偏移地址时，须使用该指令。
- 格式：
  LEA REG，MEM
- 指令要求：
  - 源操作数MEM必须是一个存储器操作数，目标操作数必须是16位通用寄存器，一般是间址寄存器（BX，BP，SI，DI）。

- 例1：

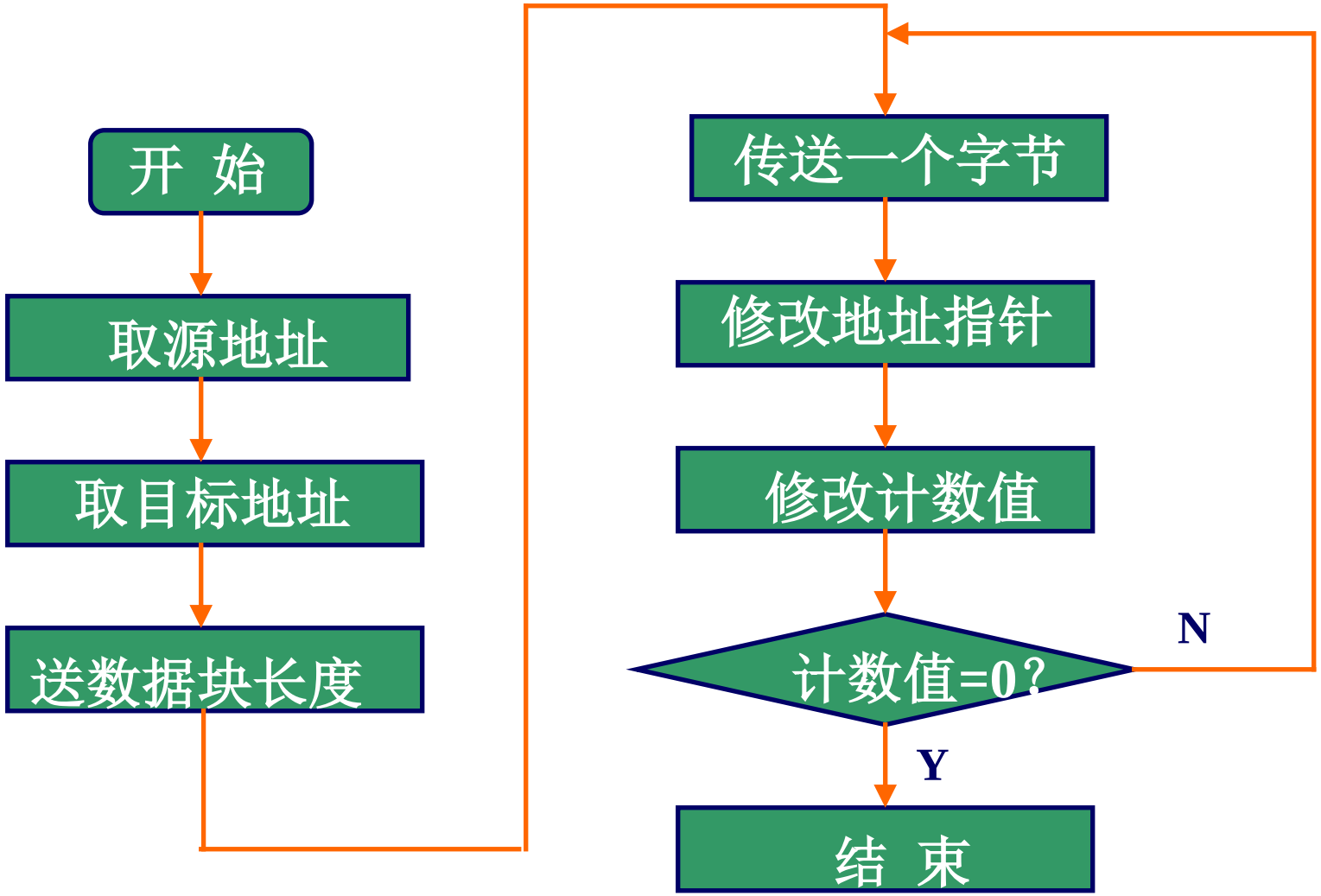
设（BX）=1000H，（DS）=6000H，（61050H）=33H，（61051H）=44H。比较以下两条指令的执行结果。

```
LEA BX，[BX+50H]
MOV BX，[BX+1]
```

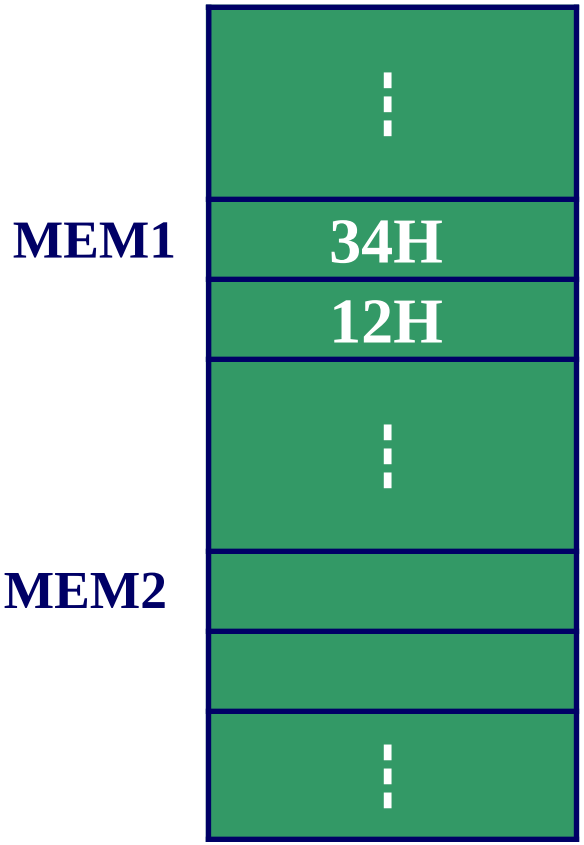
则，第1条指令执行后，（BX）=1050，第2条指令执行后，（BX）=44。

- 例2：

将数据段中首地址为MEM1 的50个字节的数据传送到同一逻辑段首地址为MEM2的区域存放。编写相应的程序段 。


开 始
取源地址
取目标地址
送数据块长度
传送一个字节
修改地址指针
修改计数值
计数值=0?
N
Y
结 束

```
        LEA   SI, MEM1
        LEA   DI, MEM2
        MOV   CL, 50
NEXT:   MOV   AL, [SI]
        MOV   [DI], AL
        INC    SI
        INC    DI
        DEC    CL
        JNZ    NEXT
        HLT
```



（2）LDS指令

- LDS reg16，mem32；

  （reg16）←((mem32)+1:(mem32))

  (DS) ←——((mem32)+3:(mem32)+2)

其中，源操作数mem32位存储器操作数，给出的是内存中4个连续的存储单元的逻辑地址。目标操作数reg16必须是BX、BP、SI、DI等4个间址寄存器之一。

该指令用于把存储器mem32中存放的一个32位远地址指针（包括偏移地址）送到reg16和DS。4个存储单元的前两个单元的内容作为偏移地址送reg16，后两个单元的内容作为段地址装入段寄存器DS

- 例1：

设（DS）=6000H，内存地址为60348H开始的4个单元中存放了一个32位的远指针98011H（该地址存放的内容是3412H），以下指令将该指针装入DS：SI中。

- LDS SI，[0348H]
- MOV AX，[SI]

指令执行后：（SI）=8011H，（DS）=9000H，（AX）=3412H

（3）LES指令

这条指令的格式及功能与LDS指令非常类似，不同的是，两个高地址单元中给出的段地址不是送往DS，而是送到ES。

例如：将上例中的LDS指令改为LES指令，则指令执行后：

（SI）=8011H，（ES）=9000H，而DS内容没有改变

3.4.1.4 标志位操作指令

- LAHF
- SAHF

} 隐含操作数AH

- PUSHF
- POPF

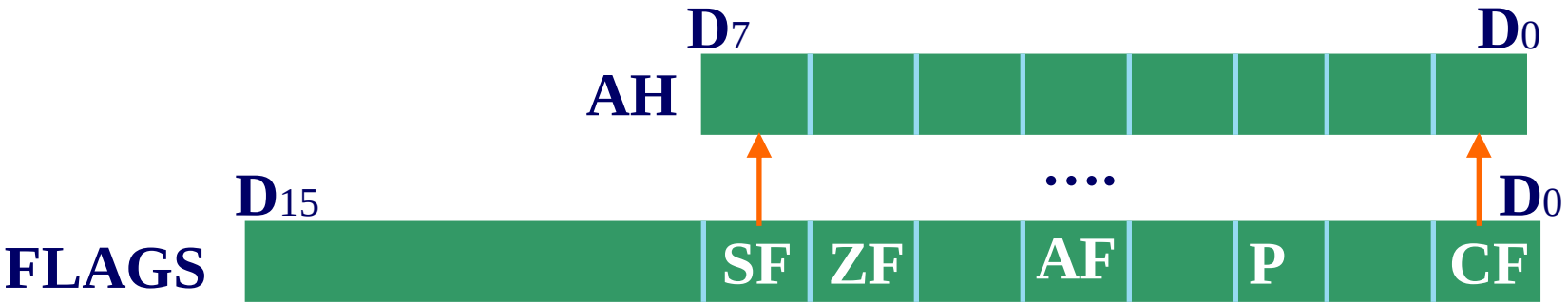
} 隐含操作数FLAGS

（1）LAHF，SAHF

- **LAHF**

  操作：

  - 将FLAGS的低8位的SF，ZF，AF，PF，和CF装入AH对应位置，8位中其余3位为无效位



- **SAHF**　　执行与LAHF相反的操作

（2）PUSHF，POPF

- 针对FLAGS的堆栈操作指令

将标志寄存器压栈或从堆栈弹出

- PUSHF:(指令本身不影响标志位)

[SP-1] ←—— （$FLAGS_H$）

[SP-2] ←—— （$FLAGS_L$）

（SP） ←—— （SP）-2

- POPF:(指令影响标志位)

（$FLAGS_L$）←—— [SP]

（$FLAGS_H$）←—— [SP+1]

（SP） ←—— （SP）+2

3.4.2 算术运算类指令

- 加法运算指令
- 减法运算指令
- 乘法指令
- 除法指令

算术运算指令的执行大多对状态标志位会产生影响

3.4.2.1 加法指令

普通加法指令**ADD**

带进位的加法指令**ADC**

加**1**指令**INC**

加法指令对操作数的要求与MOV指令相同

（1）ADD指令

- 格式：
  - ADD OPRD1，OPRD2
- 操作：
  - OPRD1+OPRD2 ⟶ OPRD1

ADD指令的执行对全部6个状态标志位都产生影响

注意：源操作数和目标操作数不能同时为存储器操作数；

不能对段寄存器进行运算。

➢ 例： MOV AL，78H

ADD AL，99H

指令执行后6个状态标志位的状态

```
  01111000
+ 10011001
----------
1 00010001
```

标志位状态： CF=1 SF= 0

AF= 1 ZF= 0

PF=1 OF= 0

```
AX=0011  BX=0000  CX=0000  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0108   NV UP EI PL NZ AC PE CY
```

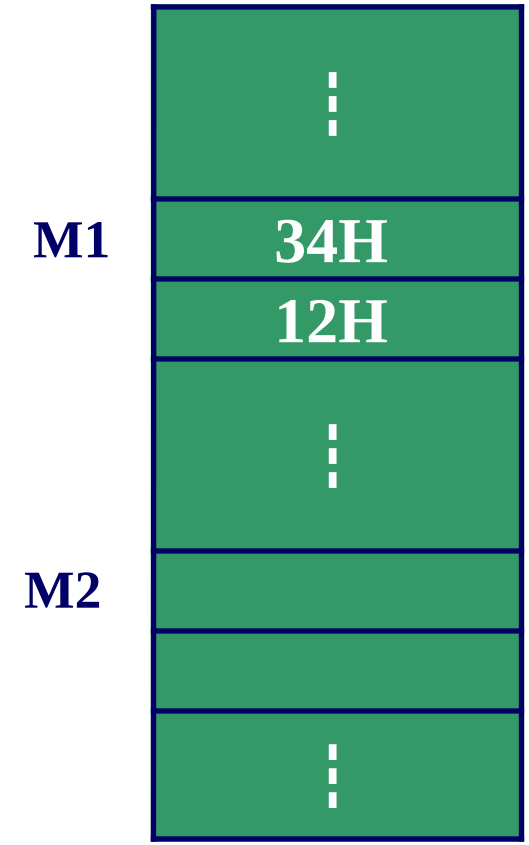
溢出标志OF(Over flow flag)	OV(1)	NV(0)
方向标志DF(Direction flag)	DN(1)	UP(0)
中断标志IF(Interrupt flag)	EI(1)	DI(0)
符号标志SF(Sign flag)	NG(1)	PL(0)
零标志ZF(Zero flag)	ZR(1)	NZ(0)
辅助标志AF(Auxiliary carry flag)	AC(1)	NA(0)
奇偶标志PF(Parity flag)	PE(1)	PO(0)
进位标志CF(Carry flag)	CY(1)	NC(0)

（2）ADC指令

- 指令格式、对操作数的要求、对标志位的影响与**ADD**指令完全一样
- 指令的操作：
  - ⑩ OPRD1+OPRD2+CF ⟶ OPRD1
- **ADC**指令多用于多字节数相加，使用前要先将**CF**清零。

例：求两个20B数的和

```
        LEA  SI，M1
        LEA  DI，M2
        MOV CX，20
        CLC              ；使CF=0
NEXT ：  MOV AL，[SI]
        ADC [DI]，AL
        INC SI
        INC DI
        DEC CX
        JNZ NEXT
HLT
```



（3）INC指令

格式：

不能是段寄存器
或立即数

INC OPRD

操作：

OPRD+1 ⟶ OPRD

常用于在程序中修改地址指针

INC指令不影响CF标志位，但对其它5个状态标志AF、OF、PF、SF及ZF会产生影响。

3.4.2.2 减法指令

- 普通减法指令SUB
- 考虑借位的减法指令SBB
- 减1指令DEC
- 比较指令CMP
- 求补指令NEG

减法指令对操作数的要求与对应的加法指令相同

（1）SUB指令

- 格式：
  - SUB　OPRD1，OPRD2
- 操作：
  - OPRD1- OPRD2 ⟶ OPRD1
- 对操作数的要求以及对标志位的影响与ADD指令相同

（2）SBB指令

- 指令格式、对操作数的要求、对标志位的影响与SUB指令完全一样。
- 主要用于多字节减法运算。
- 指令的操作：
  - OPRD1- OPRD2- CF ⟶ OPRD1

（3）DEC指令

- 格式：
  - ⑩DEC　OPRD
- 操作：
  - OPRD - 1 ⟶ OPRD

- 指令对操作数的要求与INC相同（不能是段寄存器或立即数）
- 指令常用于在程序中修改计数值

- 例：

```
        MOV   BL，2
NEXT1：  MOV   CX，0FFFFH
NEXT2：  DEC   CX
         JNZ   NEXT2        ；ZF=0转NEXT2
         DEC   BL
         JNZ   NEXT1        ；ZF=0转NEXT1
         HLT                ；暂停执行
```

（4）NEG指令（求补指令）

8/16位寄存器或存储器操作数

- 格式：
  - NEG OPRD
- 操作：
  - 0 - OPRD ⟶ OPRD

用0减去操作数，相当于对该操作数求补码

NEG指令对6个状态标志位均有影响，但要注意以下两点：

①执行NEG指令后，一般情况都会使CF为1，除非给定操作数为0，CF才会为0。

②当指定的操作数为80H（-128）或为8000H（-32768），则执行NEG指令后，结果不变，但OF置1，其它情况下OF均值0

（5）CMP指令

- 格式：
  - ⑩CMP　OPRD1，OPRD2
- 操作：
  - OPRD1- OPRD2
- 用途：用于比较两个数的大小，可作为条件转移指令转移的条件
- 指令执行的结果不影响目标操作数，仅影响标志位！
- 指令对操作数的要求及对标志位的影响与SUB指令相同

- 两个无符号数的比较

不等关系

CMP AX，BX

若 AX ＞ BX CF=0

若 AX ＜ BX CF=1

相等关系

根据ZF状态判断。如果ZF=1，则两个操作数相等，否则不相等。

- 两个带符号数的比较

CMP AX，BX

两个数的大小由OF和SF共同决定

OF和SF状态相同 AX ＞ BX

OF和SF状态不同 AX ＜ BX

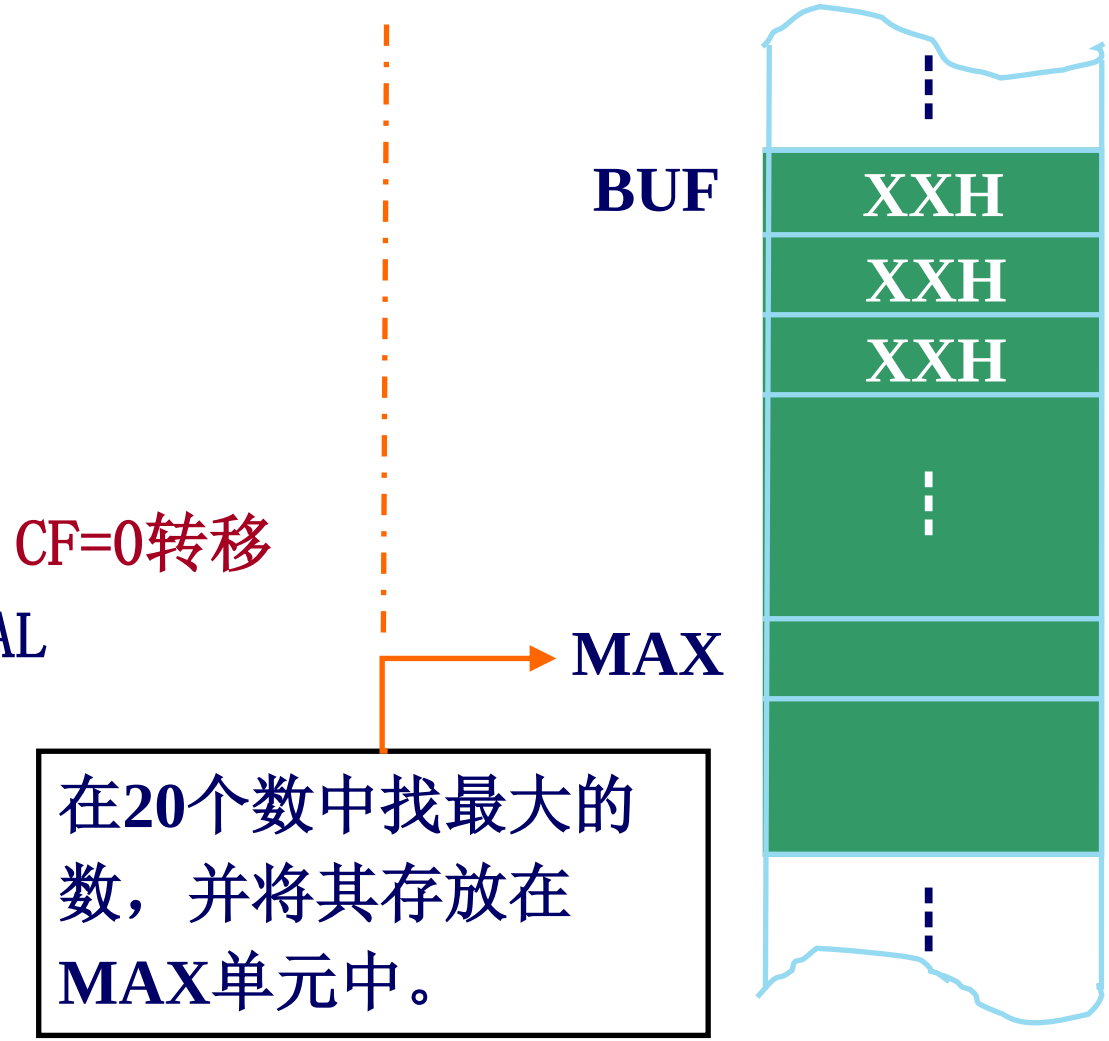
若SF=0，OF=0 则说明结果为正数，没有溢出，AX>BX

若SF=1，OF=0 则说明结果为负数，没有溢出，AX<BX

若SF=0，OF=1 则说明结果为正数，发生溢出，AX<BX

若SF=1，OF=1 则说明结果为负数，发生溢出，AX>BX

例：
```
        LEA  BX，MAX
        LEA  SI，BUF
        MOV  CL，20
        MOV  AL，[SI]
NEXT：  INC    SI
        CMP  AL，[SI]
        JNC   GOON   ；CF=0转移
        XCHG   [SI]，AL
GOON：  DEC   CL
        JNZ   NEXT
        MOV  [BX]，AL
 HLT
```



在20个数中找最大的数，并将其存放在MAX单元中。

3.4.2.3 乘法指令

- 无符号的乘法指令MUL
- 带符号的乘法指令IMUL

- 注意点：
  - **乘法指令采用隐含寻址，隐含的是存放被乘数的累加器AL或AX及存放结果的AX，DX.**

（1）无符号数乘法指令

- 格式：

  MUL　OPRD

- 操作：

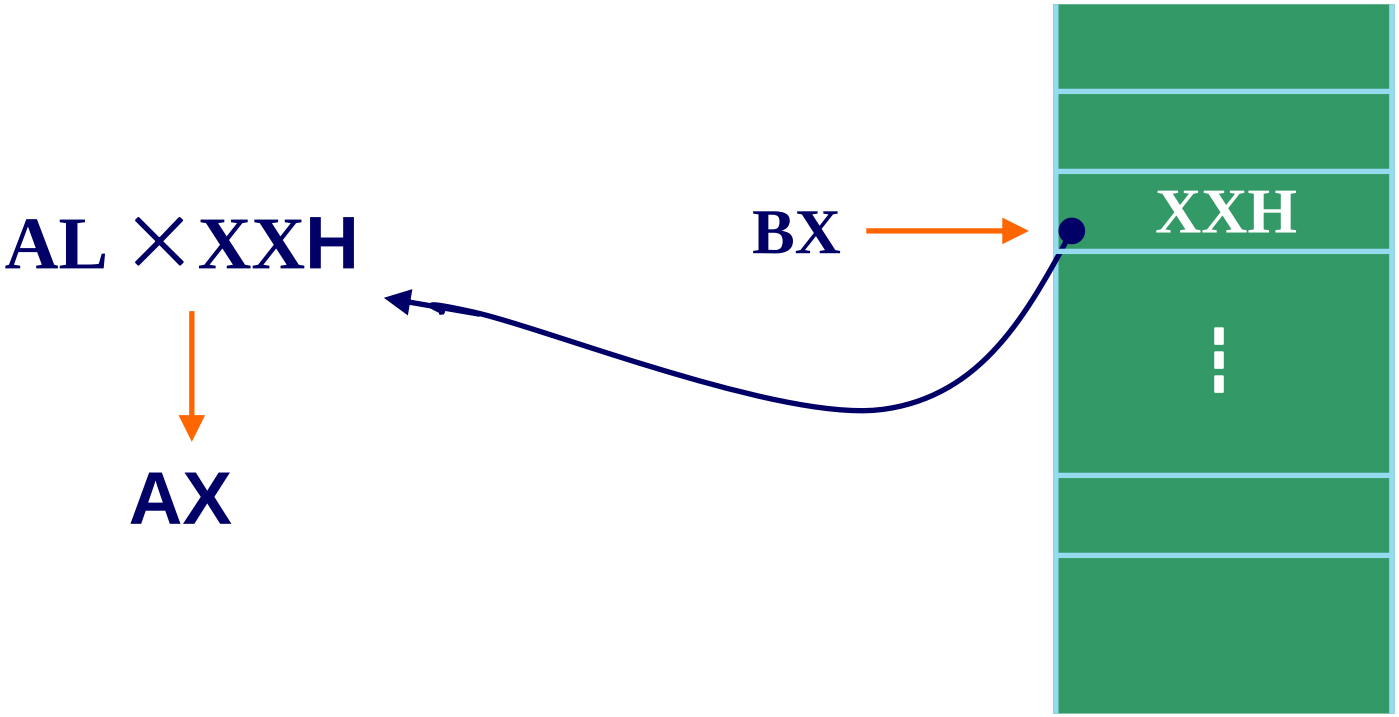
  OPRD为字节数 ⟹ AL×OPRD ⟶ AX

  OPRD为16位数 ⟹ AX×OPRD ⟶ DX：AX

- 注意：两操作数字长必须相等，且不能为立即数

【例】

MUL　BYTE　PTR[BX]



（2）带符号数乘法指令

- 格式：

  IMUL　OPRD

- 操作：

  OPRD为字节数 → AL×OPRD → AX

  OPRD为16位数 → AX×OPRD → DX：AX

DX:高16位，AX:低16位

说明：

IMUL指令在格式上和功能上都与MUL指令类似，有如下区别：

- 要求两乘数都须为有符号数
- 若乘积的高半部分是低半部分的符号位的扩展，则CF=OF=0，否则CF=OF=1
- 指令中给出的源操作数应满足带符号数的表示范围

【例】

设（AL）=FEH，（CL）=11H，求AL与CL的乘积。

- 若将两个寄存器中内容看做无符号数，则应使用指令：

MUL CL

指令执行后（AX）=10DEH，因AH中的结果不为零，故CF=OF=1

- 若将两操作数看做有符号数，则二者相乘应采用有符号数的乘法指令，即：

IMUL CL

指令执行后（AX）=FFDEH=-34，则AH中内容为AL中的符号扩展，故CF=OF=0.

3.4.2.4 除法指令

- 无符号除法指令：DIV
- 有符号除法指令：IDIV

- 进行除法时：16位/8位 → 8位商；32位/16位→16位商
- 对被除数、商及余数存放有如下规定：

	被除数	商	余数
字节除法	AX	AL	AH
字除法	DX:AX	AX	DX

除法指令规定被除数的字长必须为除数字长的2倍。如果被除数字长不够，就要使用字位扩展指令来扩展其位数。

无符号除法指令和有符号除法指令对6个标志位均无影响。

除数为16位或8位寄存器或者内存单元

（1）无符号数除法指令

- 格式

DIV　OPRD；OPRD的长度确定除法类型

操作：字节操作　(AL) ← (AX)/(OPRD)的商
　　(AH) ← (AX)/(OPRD)的余数
　　字操作　(AX) ← (DX, AX)/(OPRD)的商
　　(DX) ← (DX, AX)/(OPRD)的余数

注：

- 若除数为零或AL中商大于FFH,(或AX中商大于FFFFH)，则CPU产生一个类型0的内部中断。

【1】

```
DIV BL ；（AX）除以（BL），商=（AL），余数=（AH）
DIV WORD PTR[SI];
        ；（DX）:(AX)除以SI和SI+1所指向单元的内容，
        ；商=（AX），余数=（DX）
```

【2】

```
MOV AX，7FA2H ；（AX）=7FA2H
MOV BX，03DDH ；（BX）=03DDH
CWD           ；（DX）:(AX)=00007FA2H
DIV BX        ；商=（AX）=0021H，
              ；余数=（DX）=0025H
```

（2）有符号数除法指令

这条指令除要求操作数为有符号数外，在格式和功能上都和DIV指令类似。

例：

```
IDIV CX ；DX和AX中的32位数除以（CX），
        ；商=（AX），余数=（DX）
IDIV BYTE PTR[BX]；（AX）除以BX所指单元中的内容，
        ；商=（AL），余数=（AH）
```

IDIV指令的结果，商和余数均为带符号数，且余数符号与被除数符号相同。如-26除以+4，可得到两种结果，一是商=-6，余数=-2；另一种是商=-7，余数=+2。两种结果都正确，但是，按照8086指令系统规定，会取第一种结果。

3.4.2.5 BCD码调整指令

BCD码

BCD码：用四个二进制位表示一个十进制数字；最常用的是8421 BCD码。

- 压缩BCD码：每一个10进制位用4位二进制表示，一个字节可以表示2个10进制数。

  如：56的压缩型8421 BCD码是01010110；

- 非压缩BCD码：每一个10进制位用1个字节二进制表示。其中，高4位为0，低4位存放对应的二进制数。

  如：5的非压缩型BCD码是0000 0101，56的非压缩型BCD码是00000101 00000110。

真值	8	64
二进制编码	08H	40H
压缩BCD码	08H	64H
非压缩BCD码	08H	0604H

BCD码运算调整指令

- 加法10进制调整指令：DAA, AAA
- 减法10进制调整指令：DAS, AAS
- 乘法10进制调整指令：AAM
- 除法10进制调整指令：AAD

BCD码本质上是十进制数，即应遵循逢十进一的规则。而计算机是按二进制（十六进制）进行运算，并未按十进制规则进行运算。

注：

- BCD调整指令不能单独使用，必须跟在算术运算指令之后（除法例外）。
- 均为隐含寻址方式，隐含的操作数是AL或AL、AH。

（1）压缩BCD码加法十进制调整指令DAA

DAA用于对两个压缩BCD码相加之后的和（结果必须存入AL）进行调整，产生正确的压缩BCD码。

调整方法：

- 若（AL）中低4位>9或AF=1，则（AL）+06H →（AL），并使AF=1
- 若（AL）中高4位>9或CF=1，则（AL）+60H →（AL），并使CF=1

- 例1：编程用BCD数计算48+27=?

MOV AL，48H

ADD AL，27H

DAA

则ADD结果为：01101111。而BCD数48H+27H应该等于75H，但ADD运算结果为6FH，结果不正确，应该采用DAA指令进行调整。

因为低4位（1111）>9,故进行加6调整：

01101111+00000110=01110101

可以看出，调整后：（AL）=75H，AF=1，CF=0，结果正确。

（2）非压缩BCD码加法十进制调整指令AAA

AAA指令用于对两个非压缩BCD数相加之和（结果必须存入AL）进行调整，形成一个正确的非压缩BCD码，调整后的结果低位存入AL，高位存入AH。

具体步骤为：

- 若（AL）中低4位>9或AF=1，则（AL）+6，（AH）+1，AF=1；
- 屏蔽掉（AL）高4位，即（AL）←（AL）∩0FH；
- CF←AF

- 例1：

用非压缩BCD码计算9+4=?

```
MOV AL，09H    ；BCD数9
MOV BL，04H    ；BCD数4
ADD AL，BL     ；（AL）=09H+04H=0DH
AAA            ；（AL）=0DH+06H=03H（高4位清零）
               ；（AF）=1；（CF）=1
```

- 例2：

计算8＋9　　0000 1000

+0000 1001

0001 0001 ＝ 11

这个1代表了16，而实际上仅应为10。

结果应为17，而计算机相加为11，原因在于运算过程中，如遇到低4位往高4位产生进位时（此时AF=1）是按逢十六进一的规则，但BCD码要求逢十进一，因此只要产生进位，个位就会少6，这就要进行加6调正。

实际上当低4位的结果＞9(即A～F之间)时，也应进行加6调整(原因是逢十没有进位，故用加6的方法强行产生进位）。

如对上例的结果进行加6：

	0001 0001	11
+	0000 0110	6
	0001 0111	17

（3）压缩BCD码减法十进制调整指令DAS

DAS用于对两个压缩BCD码相减后的结果（存入AL）进行调整，产生正确的压缩BCD码。对标志位的影响与DAA指令相同。调整方法如下：

- 若（AL）中低4位>9或AF=1，则（AL）-06H，AF=1
- 若（AL）中高4位>9或CF=1，则（AL）-60H，CF=1

（3）非压缩BCD码减法十进制调整AAS

AAS用于对两个非压缩BCD码相减后的结果（存入AL）进行调整，产生正确的非压缩BCD码，其低位存入AL，高位存入AH。对标志位的影响与AAA指令相同。调整方法如下：

- 若（AL）中低4位>9或AF=1，则（AL）-06H，（AH）-1，AF=1
- 屏蔽掉（AL）高4位，即（AL）←（AL）∩0FH；
- CF←AF

（4）乘法的十进制调整指令AAM

AAM是非压缩BCD码乘法的十进制调整指令。对两个非压缩BCD码数相乘的结果（存入AX）进行调整，以得到正确的结果。具体步骤：

（AH）←（AL）/0AH

（AL）←（AL）%0AH

即把AL寄存器的内容除以0AH，商存入AH，余数存入AL。

说明：

- AAM的实质是把AL中的二进制数转换为十进制数，所以对于不超过99的二进制数，只用一条AAM指令即可实现二-十进制转换
- AAM指令影响PF、SF和ZF标志位
- 执行AAM指令前须有一条MUL指令（BCD码总视为无符号数）将两个非压缩BCD码相乘，结果放入AL，然后用AAM指令进行调整。

- 例：

用非压缩BCD码计算7*9=?

```
MOV AL，07H   ；（AL）=07H，即非压缩BCD数7
MOV BL，09H   ；（AL）=09H，即非压缩BCD数9
MUL BL        ；（AX）=07H*09H=0603H
AAM           ；（AX）=0603H，即非压缩BCD数63，SF=0，
              ；ZF=0，PF=1
```

（5）除法的十进制调整指令AAD

在两个非压缩BCD码相除之前，先用一条AAD指令进行调整，然后再用DIV指令。具体步骤：

（AL）←（AH）*A+（AL）

（AH）←0

即把AX中的非压缩BCD码（十位数放AH，个位数放AL）调整为二进制数，并将结果放入AL中。AAD的操作实质上是把AX中的两位十进制数转换为二进制数，所以对于不超过99的十进制数，只用一条AAD指令即可实现十-二进制转换。

AAD指令影响PF、SF和ZF标志位。

例：

计算23/4=?

```
MOV AX，0203H ；（AX）=0203H，即非压缩BCD数23
MOV BL，4     ；（BL）=04H，即非压缩BCD数4
AAD           ；（AX）=02H*0AH+03H=0017H
DIV BL        ；（AH）=03H，（AL）=05H，即商5余3
```

执行完AAD后，（AH）=0，（AL）=17；再执行DIV指令后，（AH）=03H，（AL）=05H

3.4.3 逻辑运算和移位指令

- 逻辑运算

  and，or，not，xor，test

- 移位操作

  非循环移位，循环移位

- 说明

- 逻辑运算指令对操作数的 要求大多与MOV指令相同。
- 除“非”运算指令 外，其余逻辑运算指令的执行都会使标志位OF=CF=0,AF值不定，并对SF，PF和ZF有影响。

3.4.3.1 逻辑运算指令

（1）“与”指令AND

- **格式：**
  - ⑩ AND　OPRD1，OPRD2
- **操作：**
  - **两操作数相“与”，结果送目标地址。**

➢ 说明：

源操作数OPRD2可以是寄存器、存储器或立即数，但目标操作数OPRD1只能是寄存器或存储器

➢ “与”指令的应用

- 实现两操作数按位相与的运算

  AND　BL，[SI]

- 使目标操作数的某些位不变，某些位清零

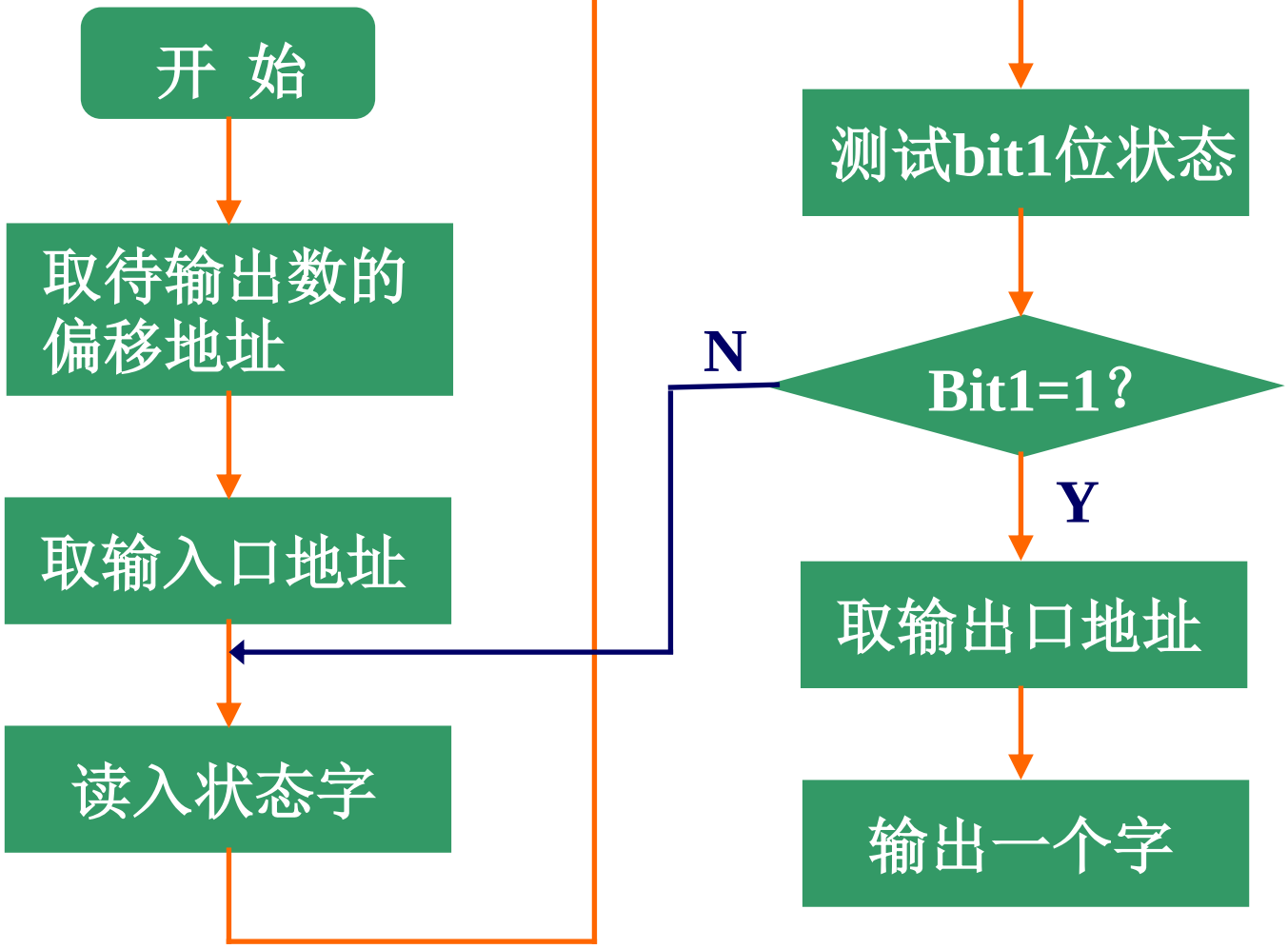
  AND　AL，0FH

- 在操作数不变的情况下使CF和OF清零

  AND　AX，AX

例

从地址为3F8H端口中读入一个字节数，如果该数bit1位为1，则可从38FH端口将DATA为首地址的一个字输出，否则就不能进行数据传送。编写相应的程序段。


开 始
取待输出数的偏移地址
取输入口地址
读入状态字
测试bit1位状态
Bit1=1?
N
Y
取输出口地址
输出一个字

```
        LEA   SI，DATA
        MOV   DX，3F8H
WATT：  IN    AL，DX
        AND   AL，02H
        JZ    WATT        ；ZF=1转移
        MOV   DX，38FH
        MOV   AX，[SI]
        OUT   DX，AX
```

（2）“或”运算指令OR

- 格式：
  - ⑩OR　OPRD1，OPRD2
- 操作：
  - 两操作数相“或”，结果送目标地址

➢ 说明：

对操作数的要求以及对标志位的影响和AND指令一样

➢ “或”指令的应用

- 实现两操作数相“或”的运算
  - OR　AX，[DI]
- 使某些位不变，某些位置“1”
  - OR　CL，0FH
- 在不改变操作数的情况下使OF=CF=0
  - OR　AX，AX

【例】奇偶校验

```
        OR   AL，AL
        JPE   GOON
        OR   AL，80H
GOON：….
```

PF=1转移

（3）“非”运算指令

- 格式：
  - NOT　OPRD
- 操作：
  - 操作数按位取反再送回原地址
  - 操作数为8位或16位的寄存器或者存储器操作数，但不能是立即数
- 注：
  - 指令中的操作数不能是立即数
  - 指令的执行对标志位无影响
- 例：NOT BYTE PTR[BX]

（4）“异或”运算指令

- 格式：
  - XOR　OPRD1，OPRD2
- 操作：
  - 两操作数相“异或”，结果送目标地址
- 例：

```
XOR　BL，80H
XOR　AX，AX
```

某一操作数和自身相“异或”，结果为0，在程序中常利用这一特性，使某寄存器清零

（5）“测试”指令TEST

- 格式：
  - TEST　OPRD1，OPRD2
- 操作：
  - 执行“与”运算，但运算的结果不送回目标地址，只是影响标志位。
- 应用：
  - 常用于测试某些位的状态
- 例：

TEST AL，02H；若AL中D1位为1，则ZF=0，否则ZF=1

TEST AX，8000H；若AX中最高位为1，则ZF=0，否则ZF=1

TEST AX，AX；该条语句，影响ZF标志位,可判断AX是否为0

- 如果AX的内容为空，那么执行该语句后ZF标志位为1
- 如果AX的内容不为空，那么执行该语句后ZF标志位为0

3.4.3.2 移位指令

- 非循环移位指令
- 循环移位指令

注：

- 移动一位时由指令直接给出；
- 移动两位及以上，则移位次数由CL指定。

（1）非循环移位

- 逻辑左移SHL
- 算术左移SAL
- 逻辑右移SHR
- 算术右移SAR

- 左移可实现乘法运算
- 右移可实现除法运算

➢ 算术左移和逻辑左移

- 算术左移指 令：

  SAL　OPRD，1
  SAL　OPRD，CL

  } 有符号数

- 逻辑左移指 令：

  SHL　OPRD，1
  SHL　OPRD，CL

  } 无符号数

- 在移动次数为1的情况下，若移位之后，操作数的最高位与CF标志位的值不相等，则溢出标志位OF=1，否则OF=0.此外，指令还影响标志位PF、SF和ZF。
- SHL指令和SAL指令的区别是：逻辑左移指令SHL将操作数视为无符号数，而算术左移指令SAL将操作数视为有符号数。OF=1对SHL指令不表示左移后溢出，而对SAL指令表示移位后超出了符号数的表示范围。

注意：SAL指令不能在Debug中测试，但可以在MASM中编译，因为汇编中将SAL自动转换为SHL执行的，这点可以通过对MASM生成的EXE文件反编译进行验证。

```
code Segment
     MOV al,2
     sal al,1
     shl bl,1
code ends
     end
```

反编译

```
C:\>debug 1.exe
-u
076A:0000 B002          MOV     AL,02
076A:0002 D0E0          SHL     AL,1
076A:0004 D0E3          SHL     BL,1
```

参考资料：浅谈8086汇编语言中SAL指令的应用研究，冯文超

【例】

MOV AL，41H

SHL AL，1

执行结果：（AL）=82H，CF=0，OF=1.若视82H位无符号数，则它没有溢出（82H<FFH）；若视为有符号数，则溢出了（82H>7FH），因为移位后正数变成了负数。

➢ 逻辑右移SHR

- 格式：

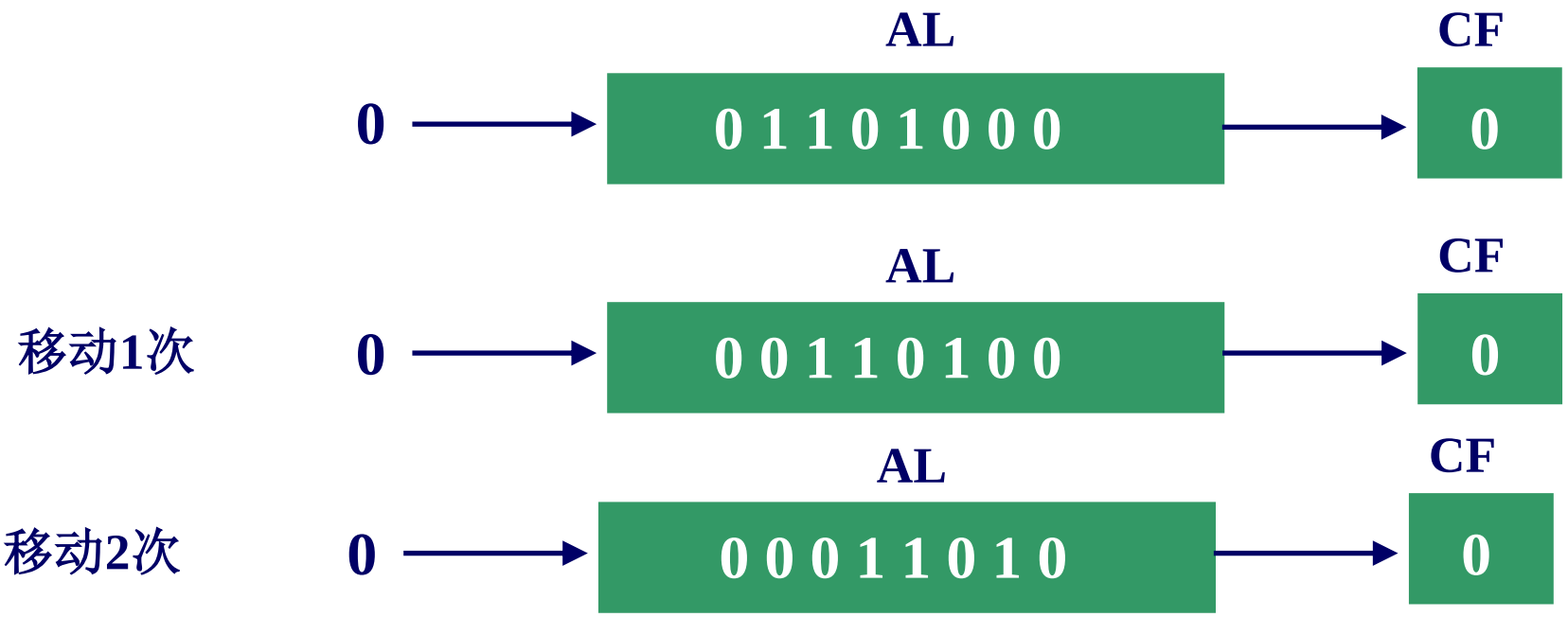
```
SHR   OPRD，1
SHR   OPRD，CL
```

无符号数的右移

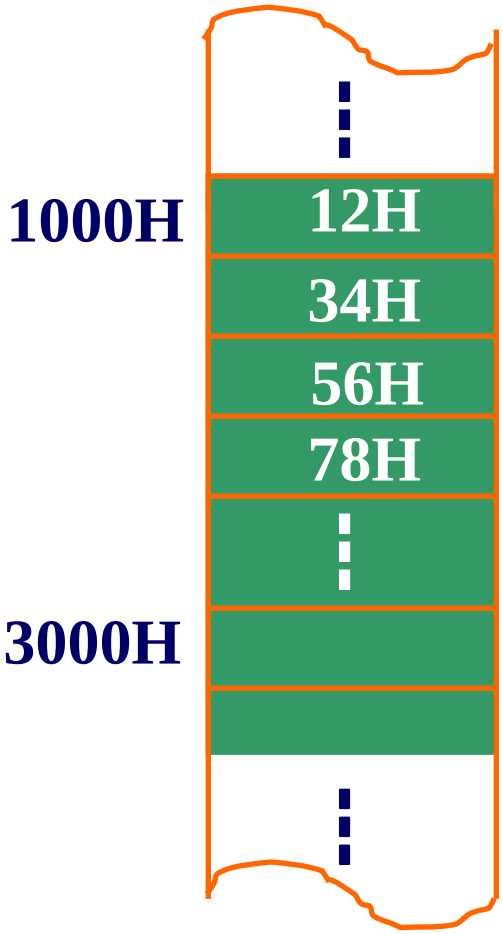
- 它将目标操作数视为无符号数，其操作是将目的操作数顺序向右移1位或CL指定的位数，每右移1位，右边的最低位移入标志位CF，而在左边的最高位补0。
- SHR指令也影响标志位CF和OF。如果移动次数为1，且移位之后新的最高位和次高位不相等，则标志位OF=1，否则OF=0。若移位次数不为1，则OF状态不定。

➢ 例1

```
MOV AL，68H
MOV CL，2
SHR AL，CL
```



【例】将1000H开始存放的4个压缩BCD码转换为ASCII码存放在3000H开始的单元中去。



```
        MOV SI,1000H
        MOV DI,3000H
        MOV CX,4
Next:   MOV AL,[SI]
        MOV BL,AL
        AND AL,0FH
        OR AL,30H
        MOV [DI],AL
        INC DI
        MOV AL,BL
        PUSH CX
        MOV CL,4
        SHR  AL,CL
        OR   AL,30H
        MOV  [DI],AL
        INC  DI
        INC SI
        POP CX
        DEC CX
        JNZ  Next
        HLT
```

为什么要或30H

算术右移

- 格式：

```
SAR   OPRD，1
SAR   OPRD，CL
```

有符号数的右移

- SAR指令将目标操作数视为有符号数，格式与SHR相同。指令的操作是将目标操作数顺序向右移1位或CL指定的位数，操作数最低位移入标志位CF。
- 与SHR指令的区别是：算术右移时最高位不是补零，而是保持不变。
- 对标志位CF、OF、PF、SF和ZF有影响，但使AF值不定。

SAR和SHR的区别

汇编语言中SAR和SHR指令都是右移指令，SAR是算数右移指令（shift arithmetic right），而SHR是逻辑右移指令（shift logical right）。

两者的区别在于SAR右移时保留操作数的符号，即用符号位来补足，而SHR右移时总是用0来补足。

例：10000000算数右移1位是11000000，而逻辑右移1位是01000000。

（2）循环移位指令

不带进位的循环移位 { 左移 ROL；右移 ROR }

带进位的循环移位 { 左移 RCL；右移 RCR }

指令格式、对操作数的要求与非循环移位指令相同

不带进位的循环移位

左移 ROL
右移 ROR

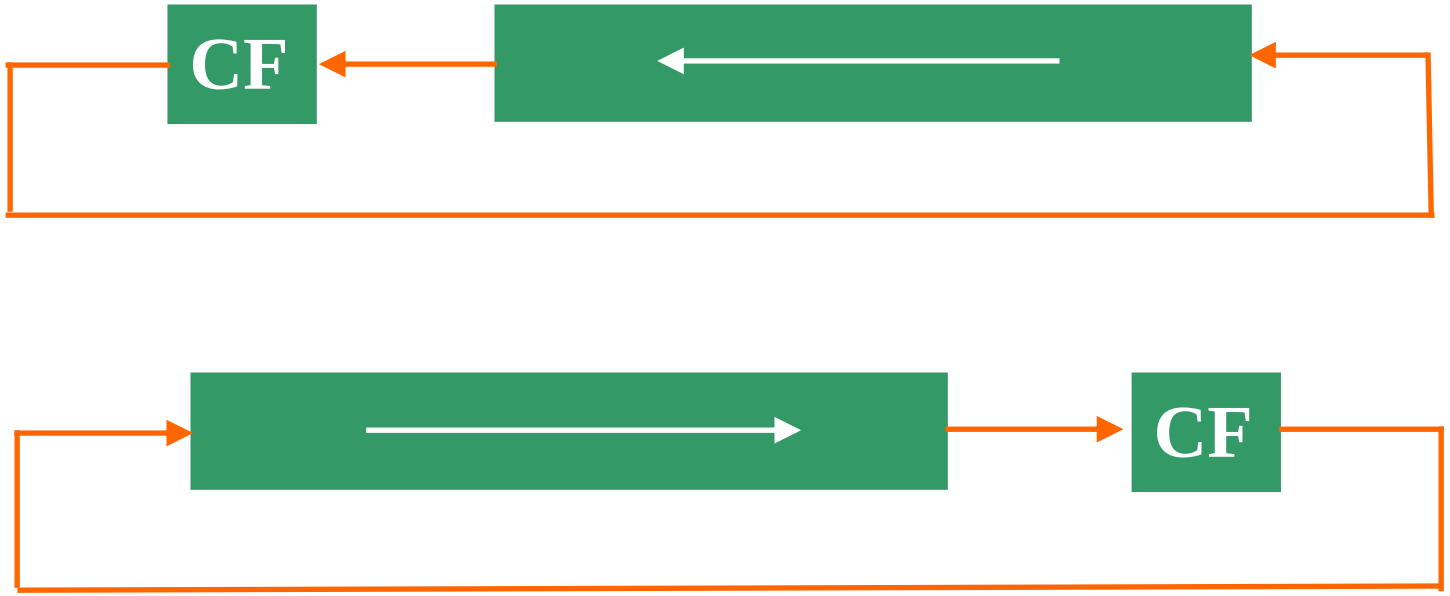
把目的地址中的数据循环左移CL次，每次从最高位（最左）移出的数据位都补充到最低位（最右），最后从最高位（最左）移出的数据位保存到CF标志位。

把目的地址中的数据循右移CL次，每次从最低位（最右）移出的数据位都补充到最高位（最左），最后从最低位（最右）移出的数据位保存到CF标志位。

ROL和ROR指令影响CF和OF标志位

带进位的循环移位

左移 RCL
右移 RCR



RCL和RCR指令影响CF和OF标志位

3.4.4 串操作指令

- 针对数据块或字符串的操作；
- 可实现存储器到存储器的数据传送；
- 待操作的数据串称为源串，目标地址称为目标串。
- 源串一般存放在DS段，偏移地址由SI指定，允许段重设；

目标串必须在ES段，偏移地址由DI指定；指令自动修改地址指针，修改方向由DF决定。

DF=0 → 增地址方向

DF=1 → 减地址方向

- 数据块长度值由CX指定
- 可增加自动重复前缀以实现自动修改CX内容。

➢ 重复前缀

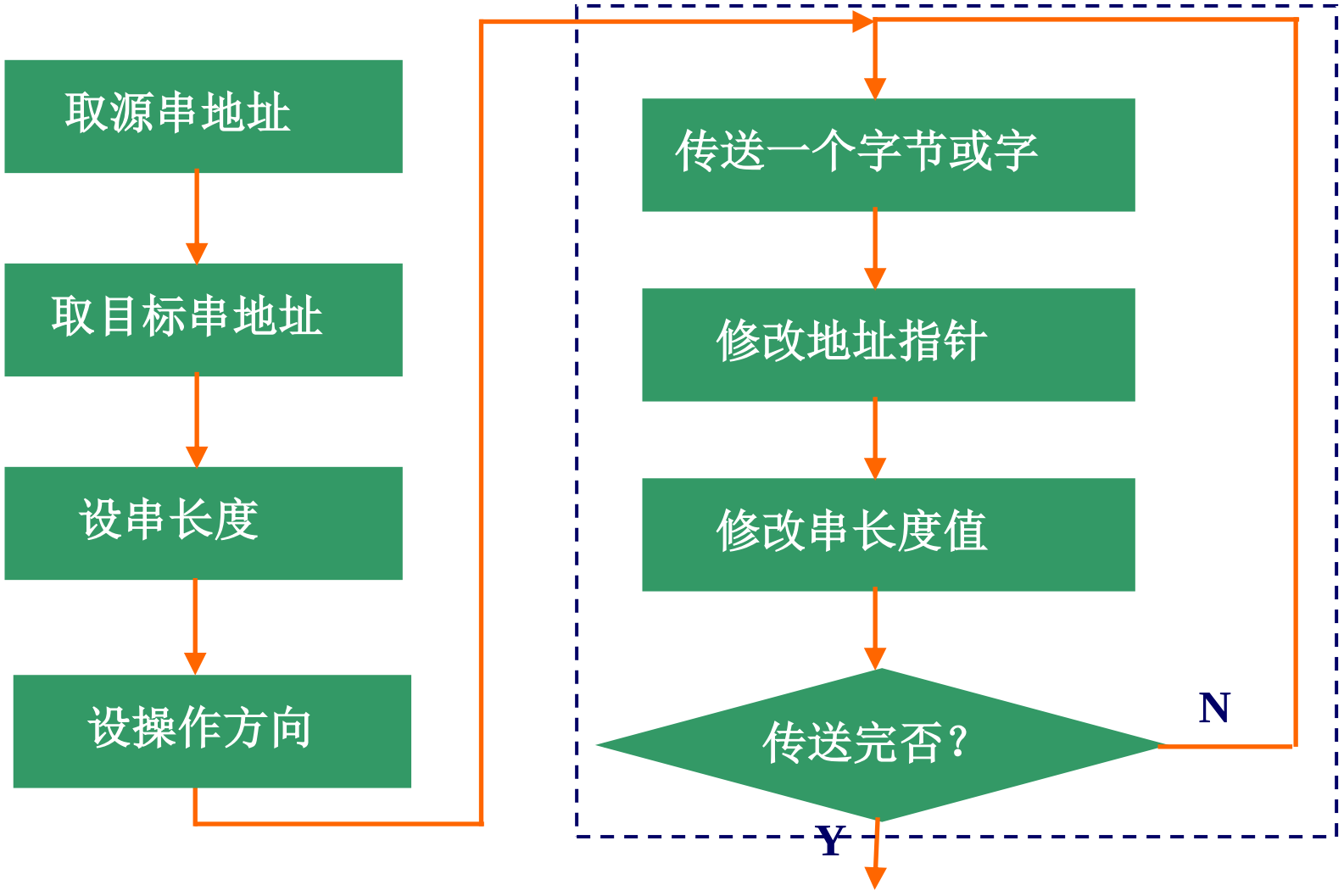
- 无条件重复
  - ⑩REP ⟶ **CX≠0 重复**
- 条件重复
  - REPE 相等重复
  - REPZ 为零重复

  } CX≠0 且 ZF=1

  - REPNE 不相等重复
  - REPNZ 不为零重复

  } CX≠0 且 ZF=0

➢ 串操作指令流程(以传送操作为例)



➢ 串操作指令

- 串传送 MOVS
- 串比较 CMPS
- 串扫描 SCAS
- 串装入 LODS
- 串送存 STOS

（1）串传送指令

- 格式：

  MOVS　　OPRD1，OPRD2 ；多用在需要段超越的场合

  MOVSB ；隐含了操作数地址，源串和目标串地址取默认值

  MOVSW ；与MOVSB类似，差别是该指令一次传送一个字

- 串传送指令常与无条件重复前缀连用

串传送指令允许进行内存单元到内存单元的数据传送，解决了MOV指令不能直接在内存单元之间传送数据的限制。

【例】设变量ADDR1和ADDR2为字类型时，下面两条指令是等效的。

MOVS ADDR1, ADDR2

MOVSW

上述第一条指令中的操作数仅指出了指令为字操作类型，实际目标操作数与源操作数的逻辑地址由ES:DI和DS:SI指出

其中第二，三种格式明确地注明是传送字节或字，第一种格式则应在操作数中表明是字还是字节操作，例如：

MOVS ES：BYTE PTR[DI]，DS：[SI]

【例】对比用MOV指令和MOVS指令实现将200个字节数据从内存的一个区域送到另一个区域的程序段。

```
LEA   SI，MEM1
LEA   DI，MEM2
MOV   CX，200
CLD
REP   MOVSB
HLT
```

（2）串比较指令

- 格式：

  CMPS　　OPRD1，OPRD2

  CMPSB

  CMPSW

- 串比较指令常与条件重复前缀连用，指令的执行不改变操作数，仅影响标志位。
- 前缀的操作对标志位不影响

将源串地址中的字节（或字）与目标串地址中的字节（或字）相比较，但比较（相减）结果不送回目标串地址中，而是反映在标志位上。

【例】测试200个字节数据是否传送正确：

```
LEA   SI，MEM1
LEA   DI，MEM2
MOV   CX，200
CLD
REPE    CMPSB
TEST   CX，00FFH
        JZ   STOP
        ……
STOP：HLT
```

（3）串扫描指令

- 格式：

  SCAS　　OPRD

  SCASB

  SCASW

目　标
操作数

- 执行与CMPS指令相似的操作，只是这里的源操作数是AX或AL。
- 常用于在指定存储区域中寻找某个关键字。

【例】　要求从段首元素的偏移地址为0100H的一个字符串（字符个数为256）中找出指定的字符（如$）。可用REPNZ SCASB指令实现。程序如下：

```
CLD                     ; DF＝0，地址自动递增
MOV    DI, 0100H        ; 目标串首元素的偏移地址
MOV    CX, 256          ; 设置计数器初始值
MOV    AL, ‘$’          ; 设关键字
REPNZ SCASB             ; 找关键字，若未找到则重复查找
```

（4）串装入指令

- 格式：

  LODS　　OPRD

  LODSB

  LODSW

源操作数

串装入指令不影响标志位，且一般不带重复前缀

- 操作：
  - 对字节：AL ← [DS:SI]
  - 对　字：AX ← [DS:SI]

- 用于将内存某个区域的数据串依次装入累加器，以便显示或输出到接口。
- LODS指令一般不加重复前缀。

（5）串存储指令

- 格式：

  STOS　　OPRD

  STOSB

  STOSW

目　标
操作数

- 操作：
  - 对字节：　AL → [ES:DI]
  - 对　字：　AX → [ES:DI]

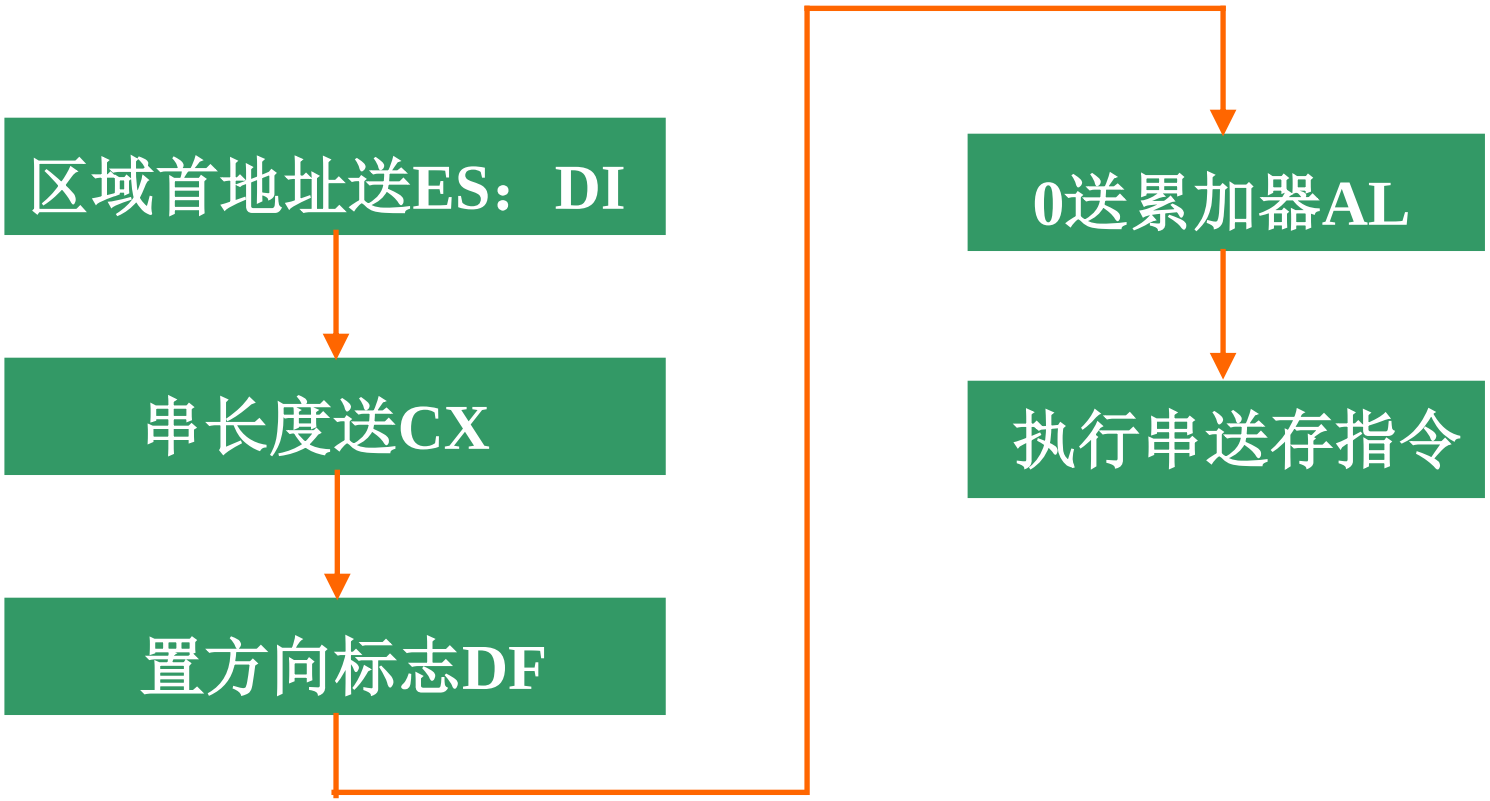
- 常用于将内存某个区域置同样的值
- 将待送存的数据放入AL（字节数）或AX（字数据）
- 确定操作方向（增地址/减地址）和区域大小（串长度值）
- 使用串存储指令+无条件重复前缀，实现数据传送

【例】 将字符$送入附加段中偏移地址为0100H的连续五个单元中。

```
CLD                 ; DF＝0，地址自动递增
MOV CX, 5           ; 设置计数器
MOV DI, 0100H       ; 目标串首元素的偏移地址
MOV AL, ‘$’         ; 设置关键字
REP STOSB           ; 连续将‘$’写入相应存储单元中
```

- 例

将内存某单元清零。设计思想如下：



3.4.5 程序控制指令

- 转移指令
- 循环控制
- 过程调用
- 中断控制

程序控制类指令与程序执行方向

- 程序控制类指令的本质是：控制程序的执行方向
- 决定程序执行方向的因素：CS，IP
- 控制程序执行方向的方法：

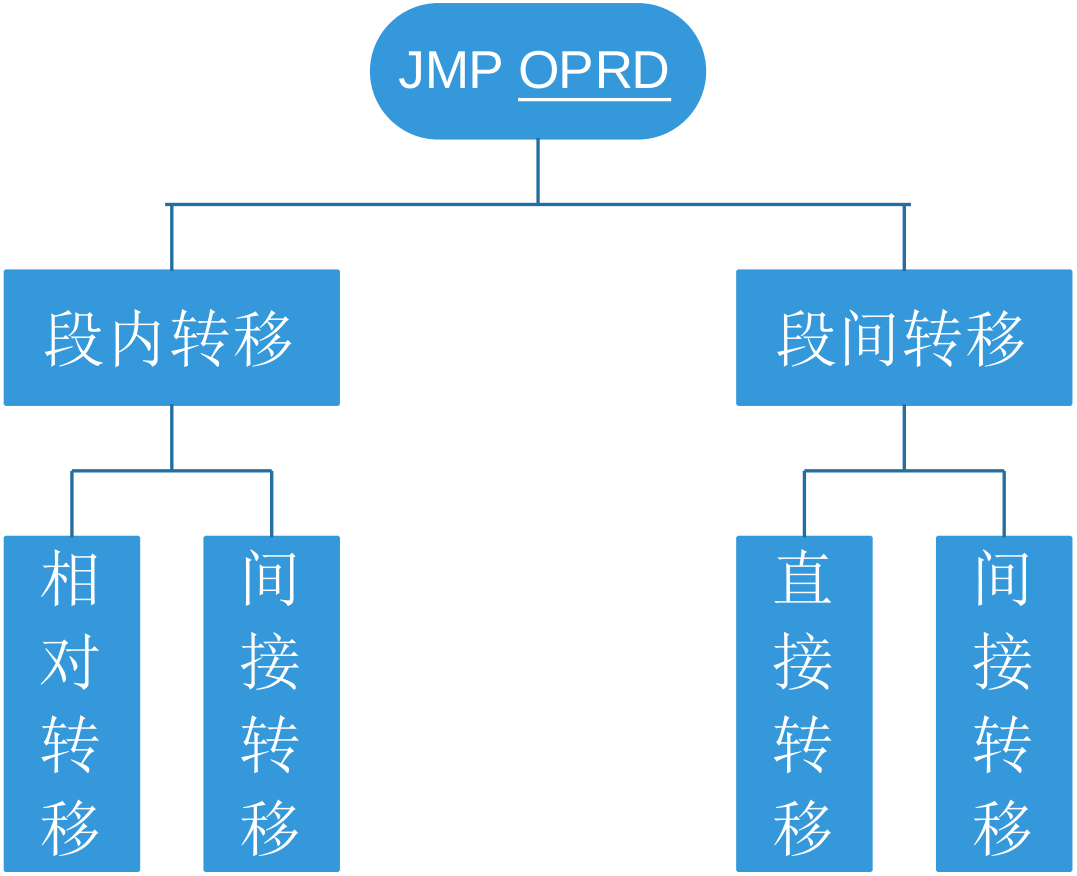
修改CS和IP，则程序转向另一个代码段执行；

仅修改IP，则程序将改变当前的执行顺序，转向本代码段内其它某处执行。

3.4.5.1 转移指令

- 无条件转移指令
  - 无条件转移到目标地址，执行新的指令
- 有条件转移指令
  - 在具备一定条件的情况下转移到目标地址

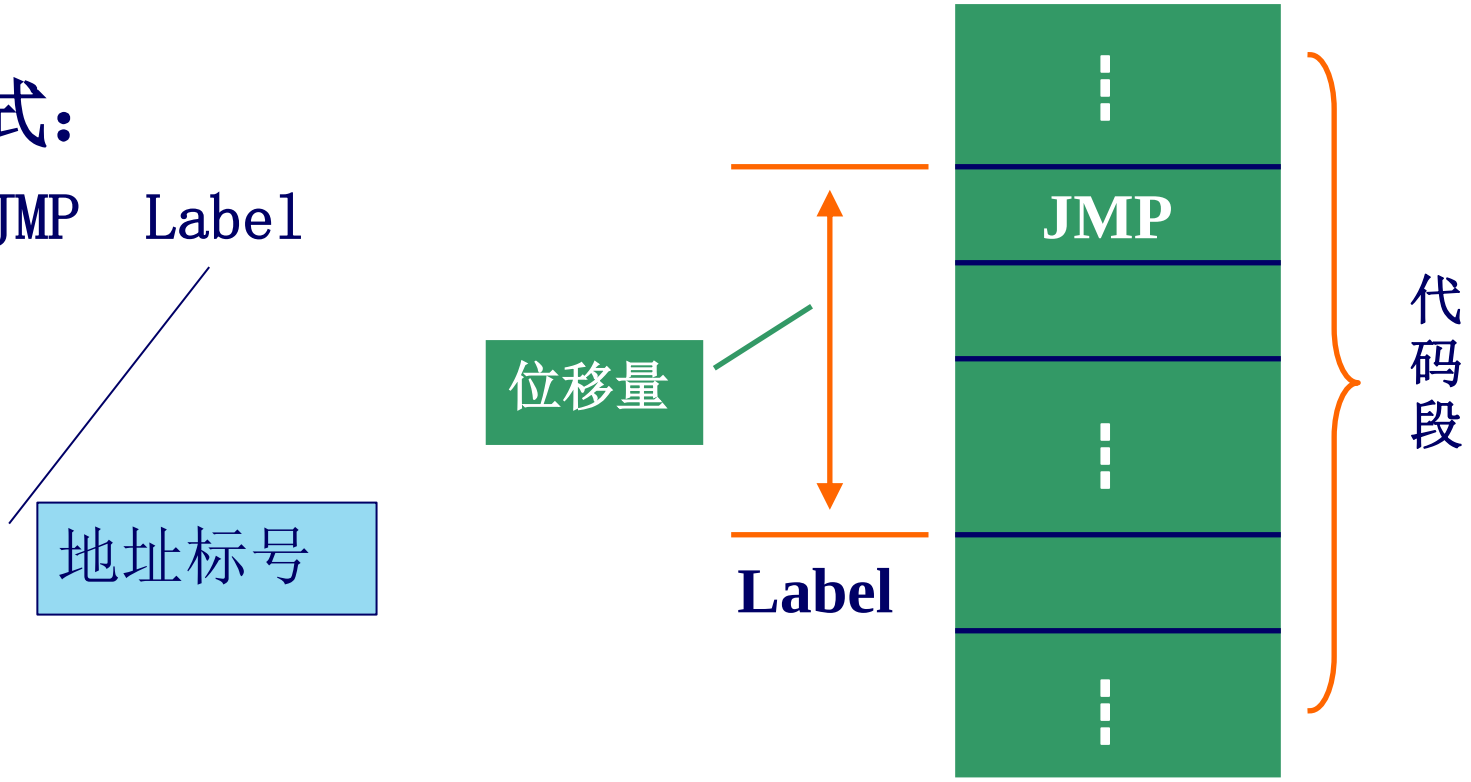
无条件转移指令



原则上可实现在整个内存空间的转移

（1）段内相对转移

- 转移的目标地址由指令直接给出
- 格式：
  - JMP　Label



下一条要执行指令的偏移地址=当前IP+位移量

位移量
（可正可负）

8位长度
（-127~+127）

段内相对
短转移

SHORT

JMP SHORT NEXT

16位长度
（-32768~+32767）

段内相对
近转移

[NEAR]
缺省默认

JMP NEAR NEXT

【例】

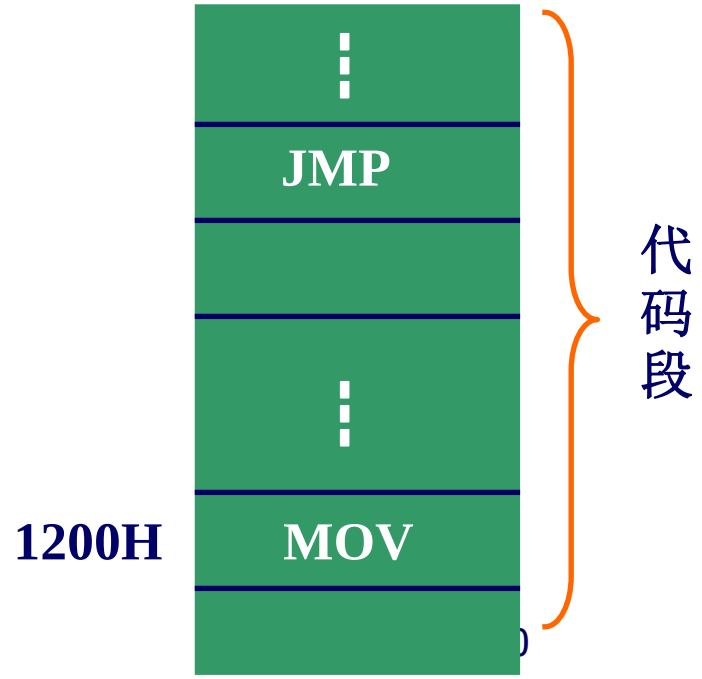
```
        ……
        MOV AX, BX
        JMP SHORT NEXT
        AND CL, 0FH
        ……
NEXT: OR CL, 7FH
```

（2）段内间接转移

- 段内间接转移
  - 转移的目标地址存放在某个16位寄存器或存储器的某两个单元中

【例】

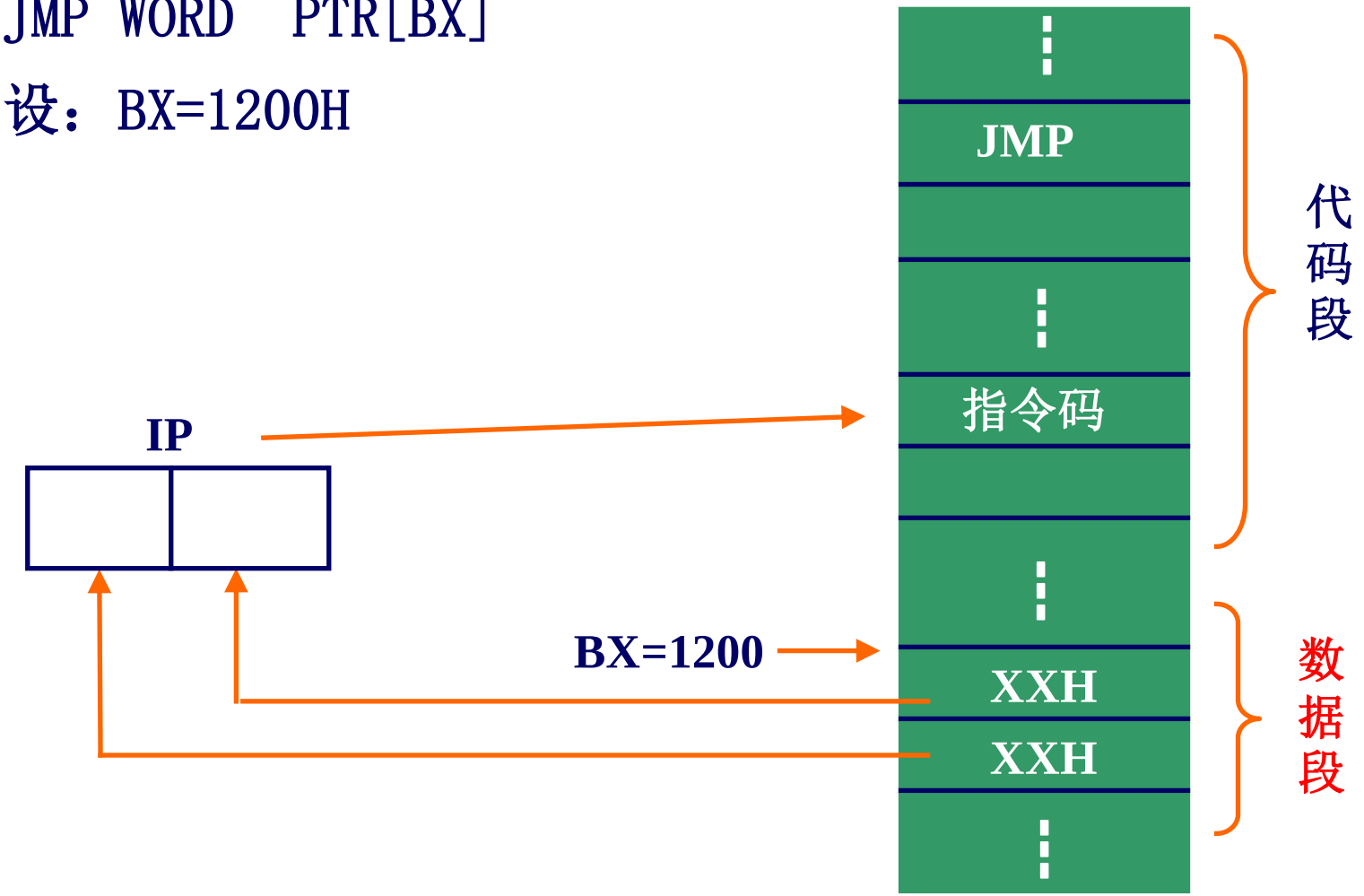
- JMP　BX
- 若：BX=1200H
- 则：转移的目标地址=1200H



【例】

JMP WORD PTR[BX]

设：BX=1200H



【例】

```
2000:0100            MOV   AX,1200H
2000:0103            JMP NEXT
                     ---
2000:0120   NEXT:  MOV BX,1200H
                     JMP   BX
                     ---
2000:1200
```

（3）段间直接转移

- 段内直接转移
  - ⑩转移的目标地址由指令直接给出。指令操作码后的连续两个字作为立即地址，低字作为偏移地址送入IP，高字作为段地址送入CS。
- 格式：
  - JMP　FAR Label

Label必须用一个32位数表达，叫做32位远指针，它就是逻辑地址。

【例】JMP 8000H：1200H

（4）段间间接转移

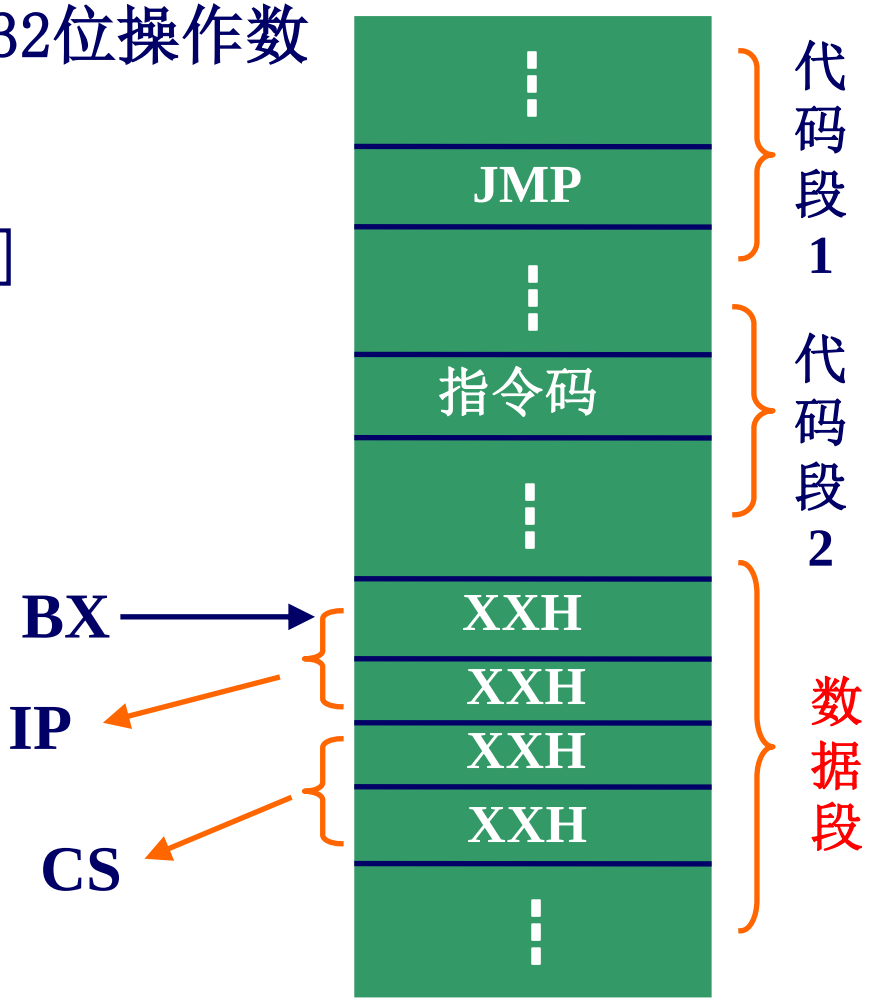
转移的目标地址由指令中的32位操作数给出

【例】JMP　DWORD　PTR[BX]

指令执行前：（DS）=3000H，（BX）=3000H，（33000H）=0BH，（33001H）=20H，（33002H）=10H，（33003H）=80H

则指令执行后，（IP）=200BH，（CS）=8010H。

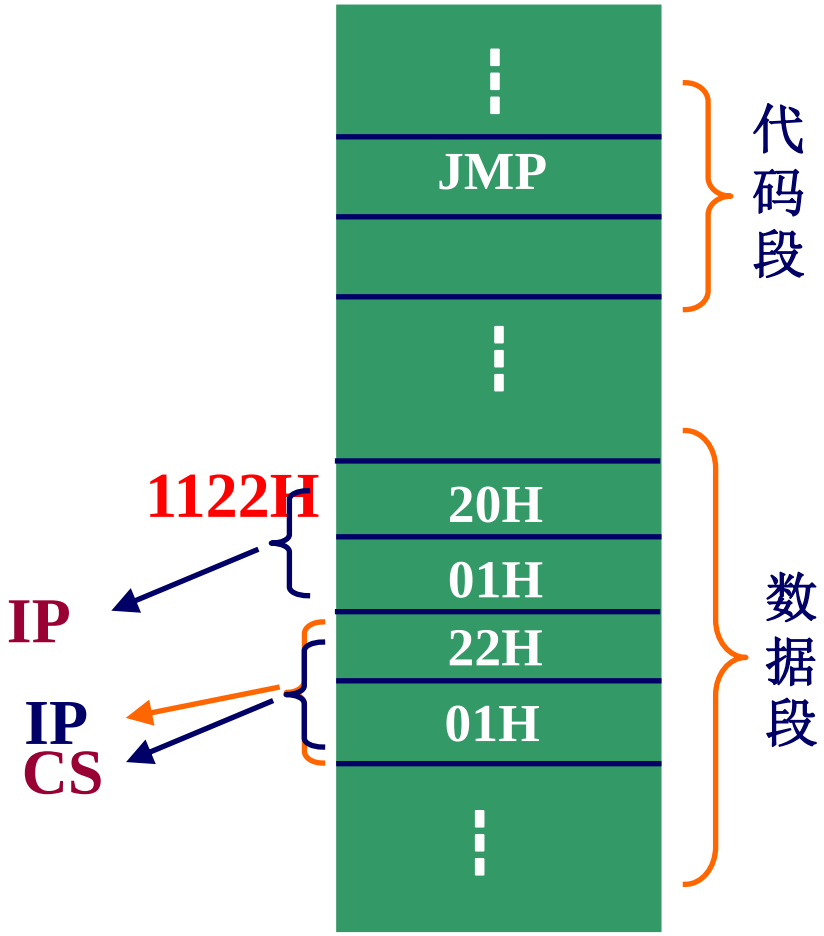
转移后的目标地址=8210BH



【例】

```
MOV   SI，1122H
MOV   WORD PTR[SI]，0120H
ADD   SI，2
MOV   WORD PTR[SI]，0122H

JMP   DWORD PTR[SI-2]
JMP   WORD  PTR[SI]
```



条件转移指令

- 在满足一定条件下，程序转移到目标地址继续执行
- 条件转移指令均为段内短转移，即转移范围为：

-128-------+127

四种转移条件

（1）以单个状态标志作为转移条件

助记符	转移条件
JZ /JE	ZF=1
JS	SF=1
JO	OF=1
JP	PF=1
JC	CF=1

助记符	转移条件
JNZ /JNE	ZF=0
JNS	SF=0
JNO	OF=0
JNP	PF=0
JNC	CF=0

（2）以CX的值为0作为转移条件

助记符	转移条件
JCXZ	CX=0

（3）以两个无符号数比较的结果作为转移条件

助记符	转移条件	
JA /JNBE	CF ∨ZF=0	高于/不低于等于
JAE /JNB /JNC	CF=0	高于等于/不低于/无进位
JB/JNAE /JC	CF=1	低于/不高于等于/有进位
JBE /JNA	CF ∨ZF=1	低于等于/不高于

（4）以两个有符号数比较的结果作为转移条件

助记符	转移条件	
JG /JNLE	（SF ⊕OF ）∨ZF=0	大于/不小于等于
JGE /JN L	SF ⊕OF=0	大于等于/不小于
JL/JNGE	SF ⊕OF=1	小于/不大于等于
JLE /JNG	（SF ⊕OF ）∨ZF=1	小于等于/不大于

【例】JZ/JNZ指令

```
        test al,80h     ; 测试最高位
        jz next0        ; D7＝0（ZF＝1），转移
        mov ah,0ffh     ; D7＝1，顺序执行
        jmp done        ; 无条件转向
next0:  mov ah,0
done:   ...
```

```
        test al,80h     ; 测试最高位
        jnz next1       ; D7＝1（ZF＝0），转移
        mov ah,0        ; D7＝0，顺序执行
        jmp done        ; 无条件转向
next1:  mov ah,0ffh
done:   ...
```

【例】：JS/JNS指令

```
；计算|X－Y|（绝对值）
；X和Y为存放于X单元和Y单元的16位操作数
；结果存入result
        mov ax,X
        sub ax,Y
        jns nonneg
        neg ax        ；neg是求补指令
nonneg:mov result,ax
```

【例】：JO/JNO指令

计算X－Y；

；X和Y为存放于X单元和Y单元的16位操作数

；若溢出，则转移到overflow处理

```
            mov  ax,X
            sub  ax,Y
            jo overflow
            ...      ；无溢出，结果正确
overflow:   ...      ；有溢出处理
```

【例】JC/JNC指令

记录BX中1的个数

```
        xor al,al；AL＝0，CF＝0
again:  test bx,0ffffh ；等价于
                       ；cmp bx,0
        je next
        shl bx,1
        jnc again
        inc al
        jmp again
next:   ...；AL保存1的个数
```

```
        xor al,al；AL＝0，CF＝0
again:  cmp bx,0
        jz next
        shl bx,1；也可使用
                 ；shr bx,1
        adc al,0
        jmp again
next:   ...；AL保存1的个数
```

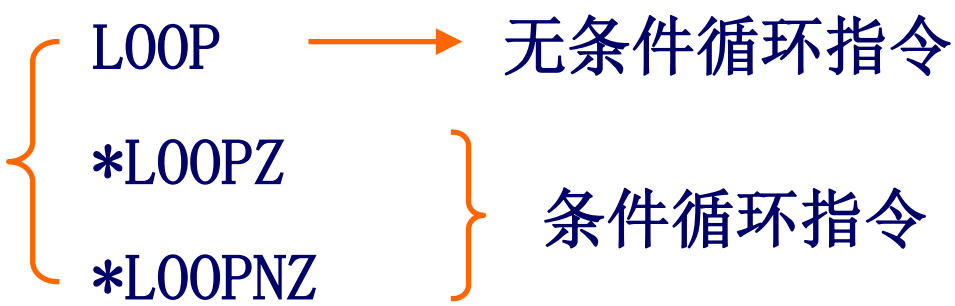
3.4.5.2 循环控制指令

- 循环范围：

  以当前IP为中心的-128～+127范围内循环。

- 循环次数由CX寄存器指定。

- 循环指令：



（1）无条件循环指令

- 格式：

  LOOP　LABEL

- 循环条件：

  CX ≠ 0

- 操作：

  DEC　CX
  JNZ　符号地址

（2）条件循环指令

格式：

LOOPZ LABEL

LOOPE LABEL

循环条件：

CX ≠ 0 ZF=1

格式：

LOOPNZ LABEL

LOOPNE LABEL

循环条件：

CX ≠ 0 ZF=0

功能：CX－1→CX，若**CX≠0且ZF=1**则转移至目标，否则停止循环(CX=0或ZF=0)。

不影响标志位

LOOPxx　　目标地址

助记符	转移条件	不转移条件
LOOP	CX≠0	CX=0
LOOPZ	CX≠0 且 ZF=1	CX=0 或 ZF=0
LOOPNZ	CX≠0 且 ZF=0	CX=0 或 ZF=1

- LOOP退出循环条件是（CX）= 0
- LOOPZ和LOOPNZ提供了提前结束循环的可能，不一定要等到(CX)= 0才退出循环：
- 在串中查找字符，不相等时继续查找，使用指令LOOPNZ。
- 比较两串时，当有字符不等就可退出，使用指令LOOPZ。

【例】记录空格个数

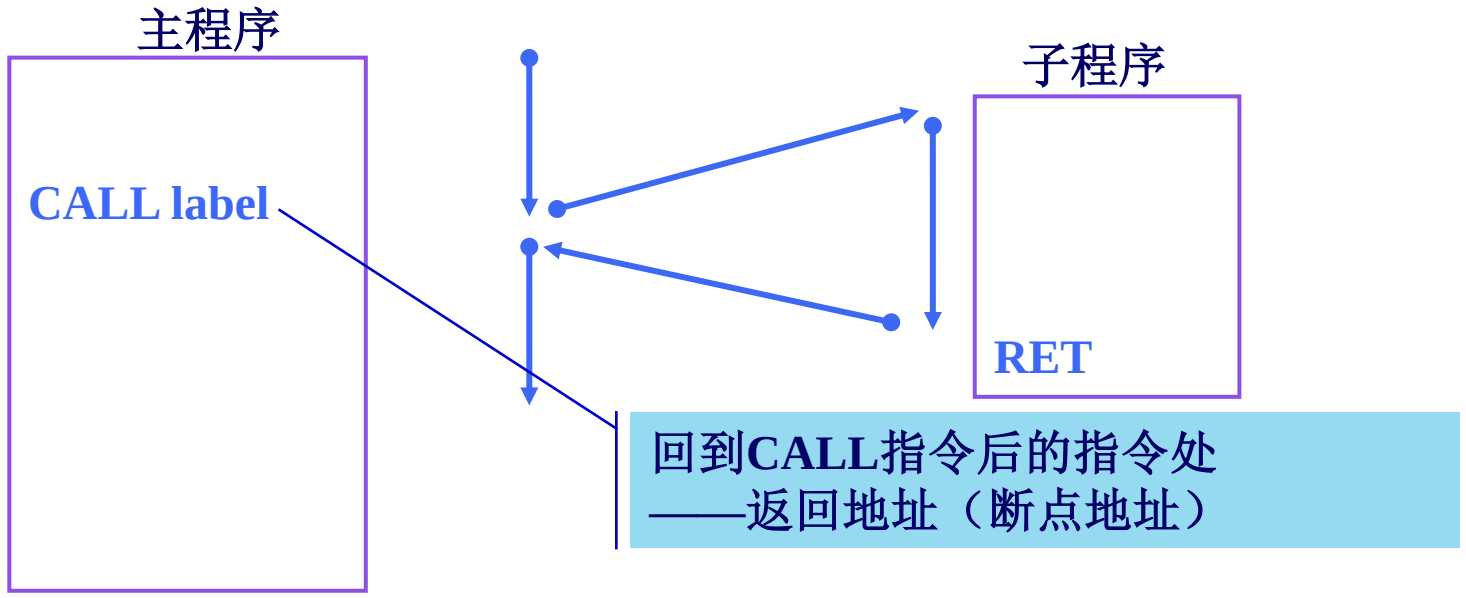
```
        mov cx,count    ; 设置循环次数
        mov si,offset string
        xor bx,bx       ; bx＝0，记录空格数
        jcxz done       ; 如果长度为0，退出
        mov al,20h
again:  cmp al,es:[si]
        jnz next        ; ZF＝0非空格，转移
        inc bx          ; ZF=1是空格，个数加1
next:   inc si
        loop again
        ; 字符个数减1，不为0继续循环
done:   hlt
```

```
cmp cx,0
jz done
```

```
dec cx
jnz again
```

3.4.5.3 过程调用和返回

- 子过程（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



➢ 调用指令的执行过程

- 保护断点；

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

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]←IP
SP←SP－2，SS:[SP]←CS

CALL OPRD

- 过程调用
  - 段内调用
    - 段内直接调用
    - 段内间接调用
  - 段间调用
    - 段间直接调用
    - 段间间接调用

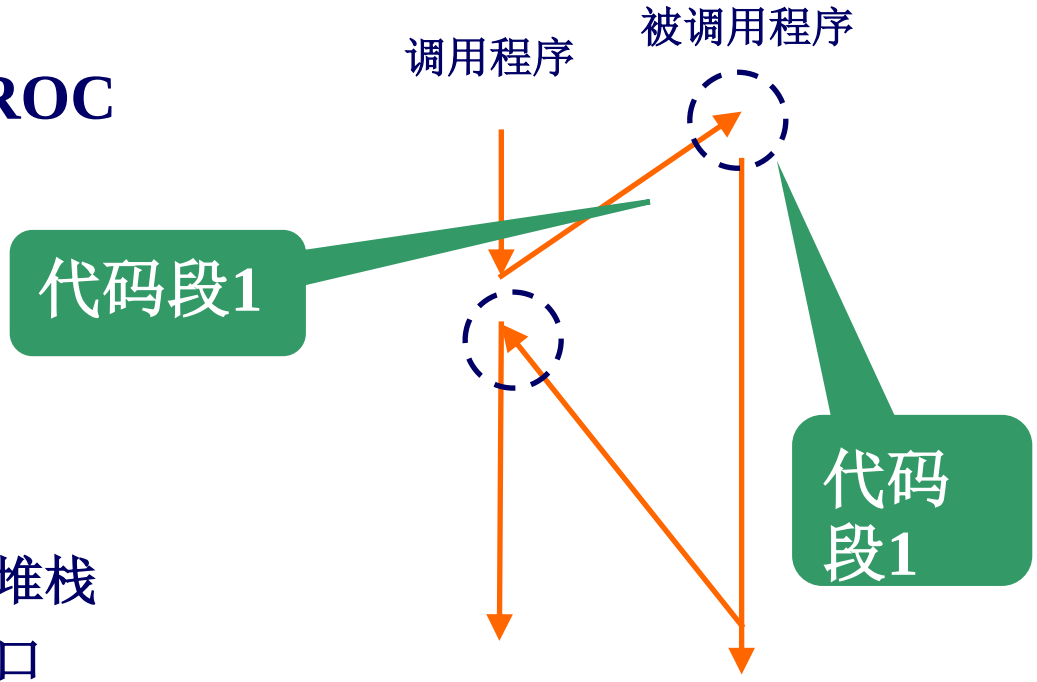
（1）段内调用

PUSH IP
JMP NEAR PTR PROC

- 被调用程序与调用程序在同一代码段
  - ⑩调用前只需保护断点的偏移地址
- 格式：

  CALL NEAR PROC

  近过程名



- 执行过程：
  - 将断点的偏移地址压入堆栈
  - 根据过程名找子程序入口

【例】

（1）CALL　NEAR PROC ⟶ 直接调用（NEAR可省略）

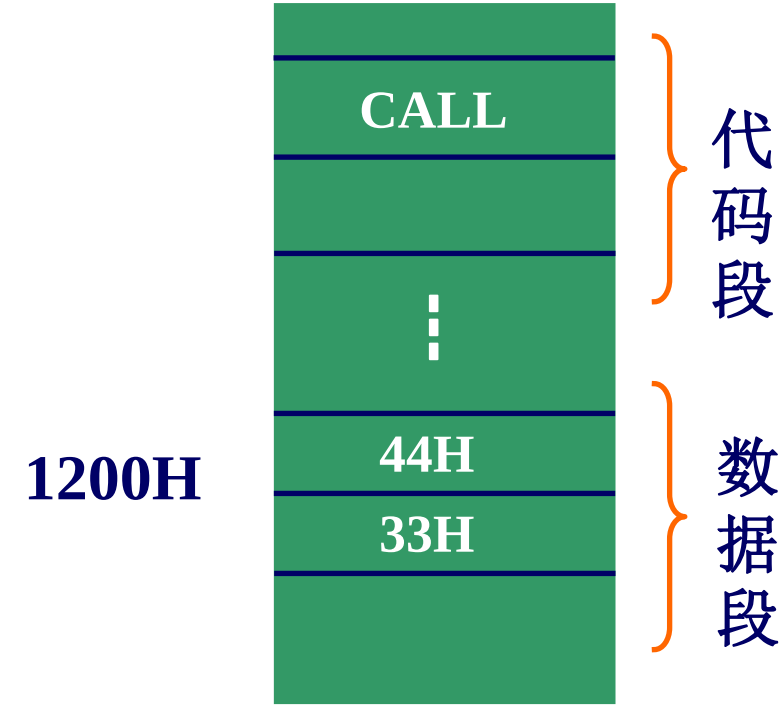
（2）CALL　WORD　PTR[SI] ⟶ 间接调用

设：SI=1200H
　　CS=6000H

执行第（2）条指令后：

CS =6000H

IP =3344H



（2）段间调用

- 子过程与原调用程序不在同一代码段

调用前需保护断点的段基地址和偏移地址

- 断点保护时的压栈顺序：先将断点的CS压栈，再压入IP。

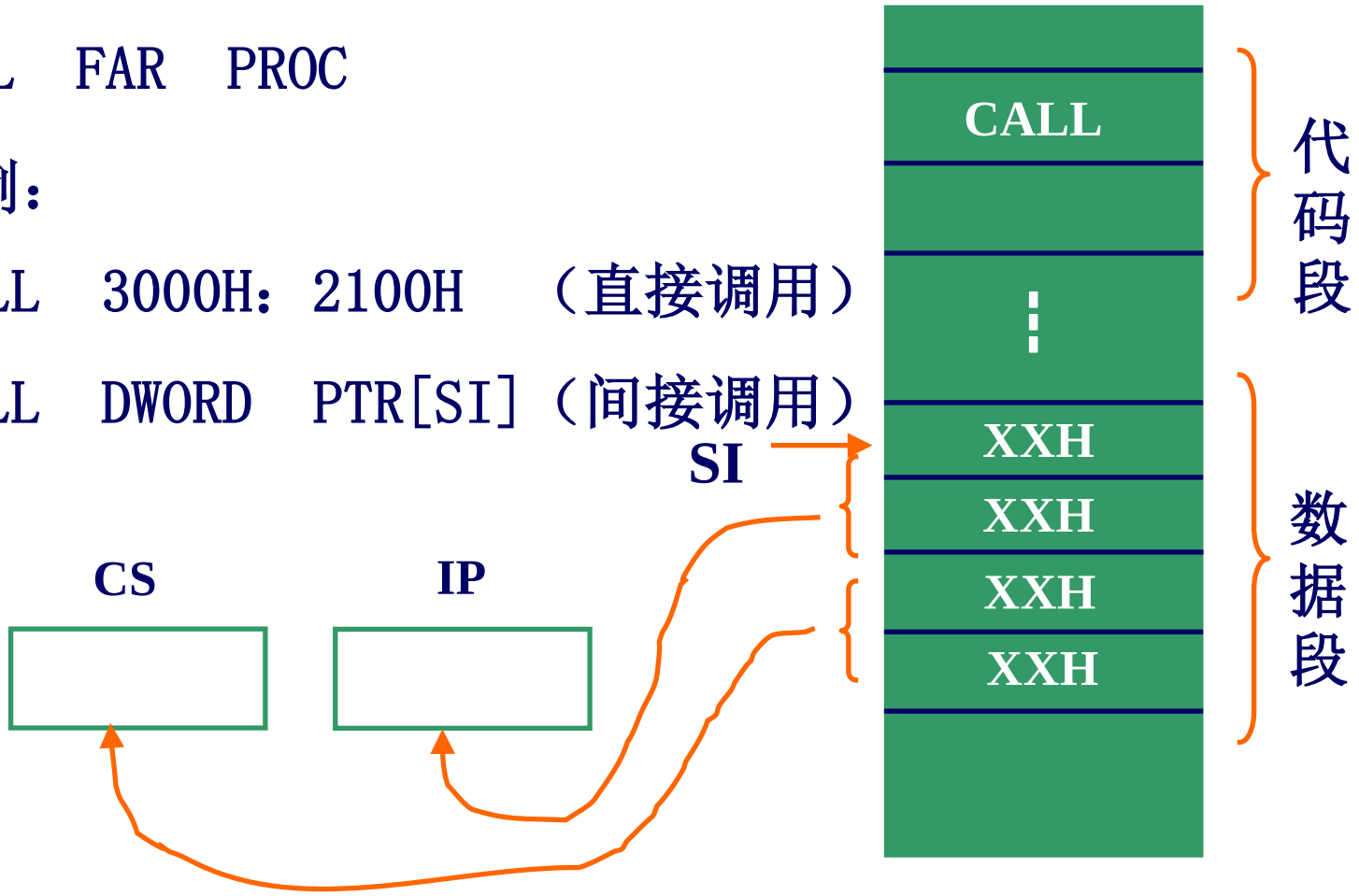
【例】

PUSH CS
PUSH IP
JMP FAR PTR PROC

- 格式：

  CALL FAR PROC

- 格式例：
  - CALL 3000H：2100H （直接调用）
  - CALL DWORD PTR[SI]（间接调用）



（3）返回指令

- 根据段内和段间、有无参数，分成4种类型

```
RET          ；无参数段内返回
RET i16      ；有参数段内返回
RETF         ；无参数段间返回
RETF i16     ；有参数段间返回
```

POP IP
SP=SP+i16

- 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；POP IP
  - 段间返回——出栈偏移地址IP和段地址CS

    IP←SS:[SP]，SP←SP＋2； POP IP

    CS←SS:[SP]，SP←SP＋2； POP CS

【例】子程序

```
; 主程序
        mov al,0fh      ; 提供参数AL
        call htoasc     ; 调用子程序
        ...
; 子程序：将AL低4位的一位16进制数转换成ASCII码
htoasc: and al,0fh      ; 只取al的低4位
        or al,30h       ; al高4位变成3
        cmp al,39h      ; 是0～9，还是0Ah～0Fh
        jbe htoend
        add al,7        ; 是0Ah～0Fh，加上7
htoend: ret             ; 子程序返回
```

3.4.5.4 中断指令

- 中断
- 中断源
- 中断的类型
- 中断指令

➢ 中断与过程调用

- 中断是随机事件或异常事件引起，调用则是事先已在程序中安排好；
- 响应中断请求不仅要保护断点地址，还要保护FLAGS内容；
- 调用指令在指令中直接给出子程序入口地址，中断指令只给出中断向量码，入口地址则在向量码指向的内存单元中。

（1）中断指令

- 格式：

  INT　n

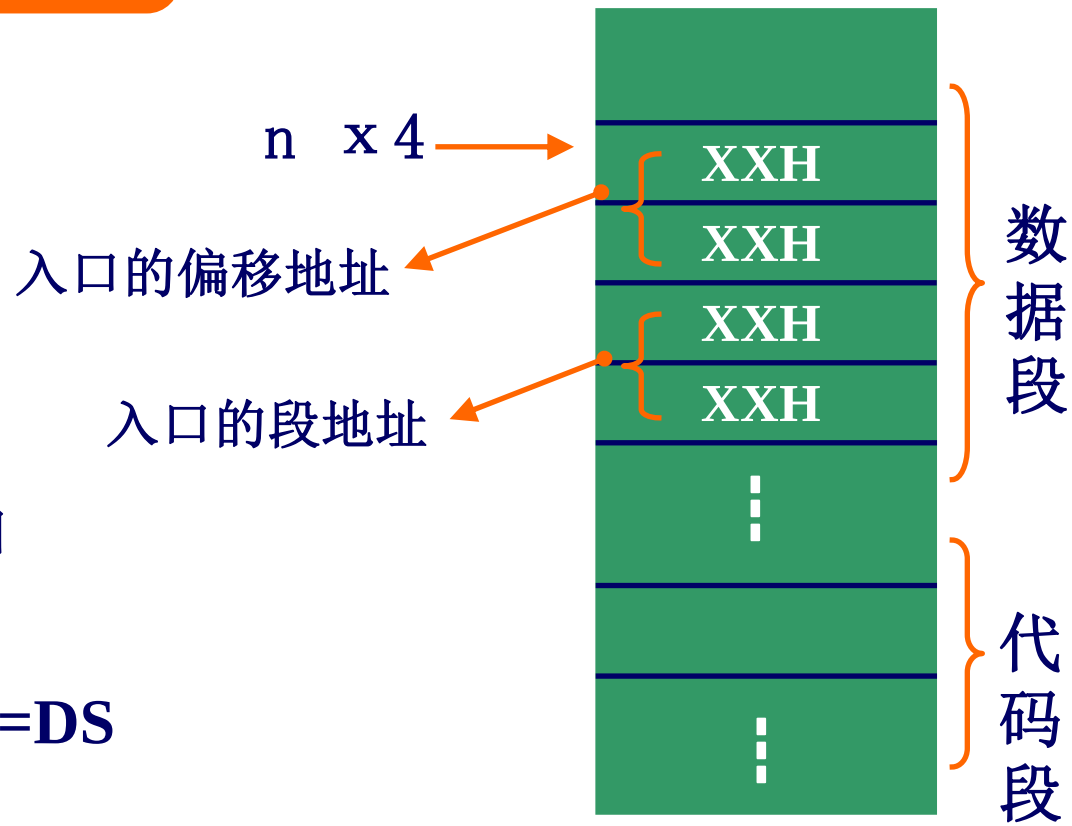
中断类型码
n=0 ～ 255

- 说明：

  n x 4

存放中断服务子程序入口地址的单元的偏移地址

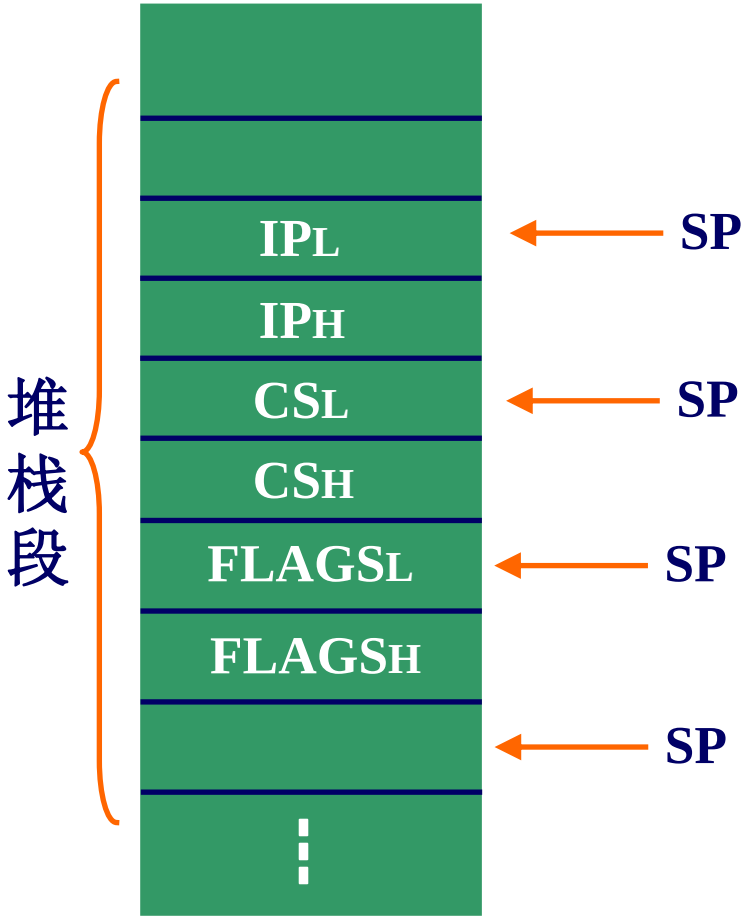
该单元在数据段，段地址=DS

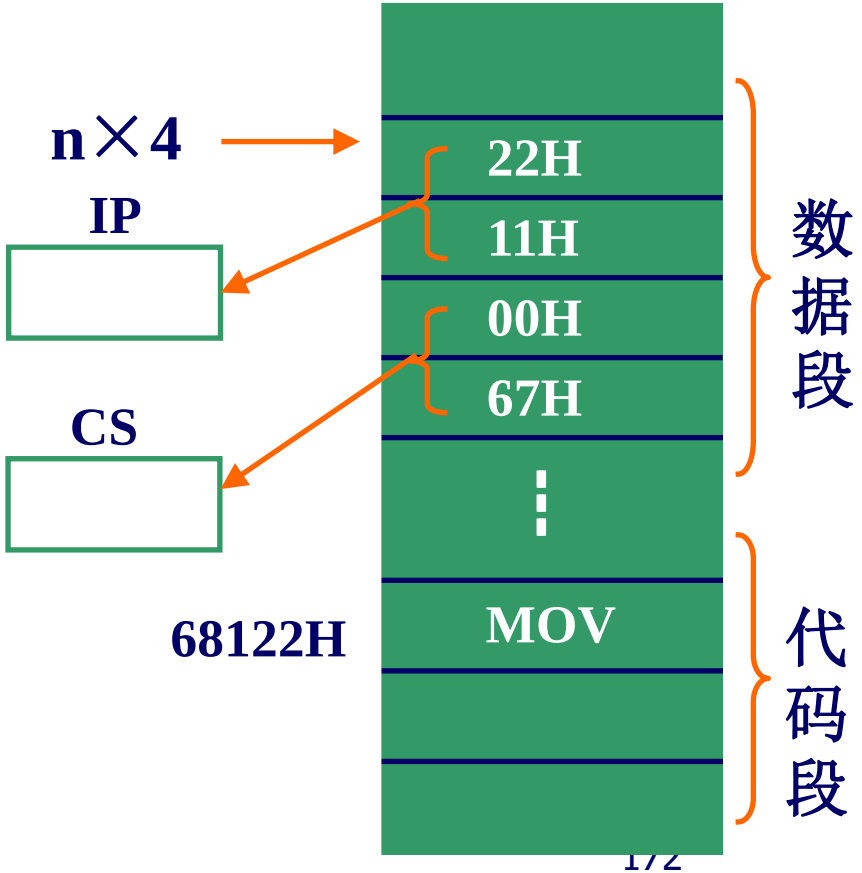


➢ 中断指令的执行过程

- 将FLAGS压入堆栈；
- 将INT指令的下一条指令的CS、IP压栈；
- 由n×4得到存放中断向量的地址；
- 将中断向量（中断服务程序入口地址）送CS和IP寄存器；
- 转入中断服务程序。

中断指令的执行过程


堆栈段
IPL
IPH
CSL
CSH
FLAGSL
FLAGSH
SP
SP
SP
SP


n×4
IP
CS
68122H
22H
11H
00H
67H
MOV
数据段
代码段

➢ 例

执行程序段：

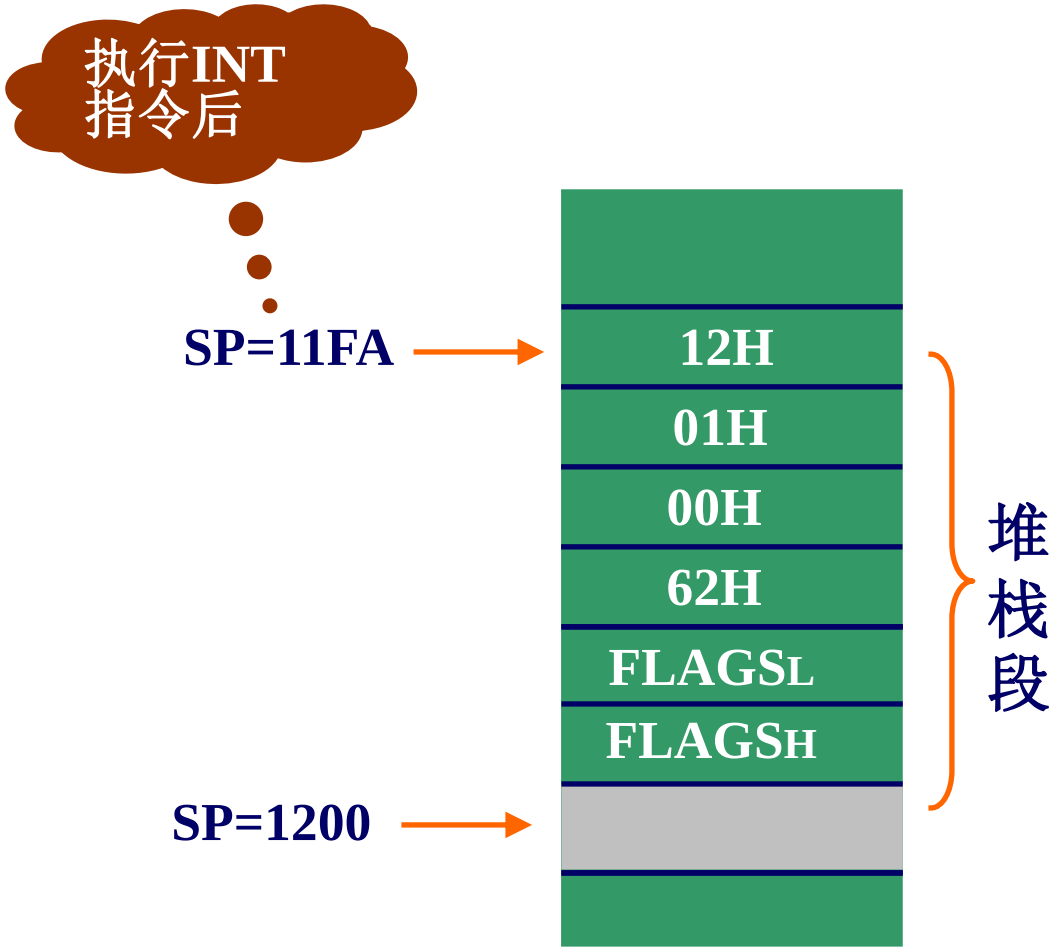
CS　　IP

⋮

6200H:0110H　INT 21H

6200H:0112H　MOV AX，BX

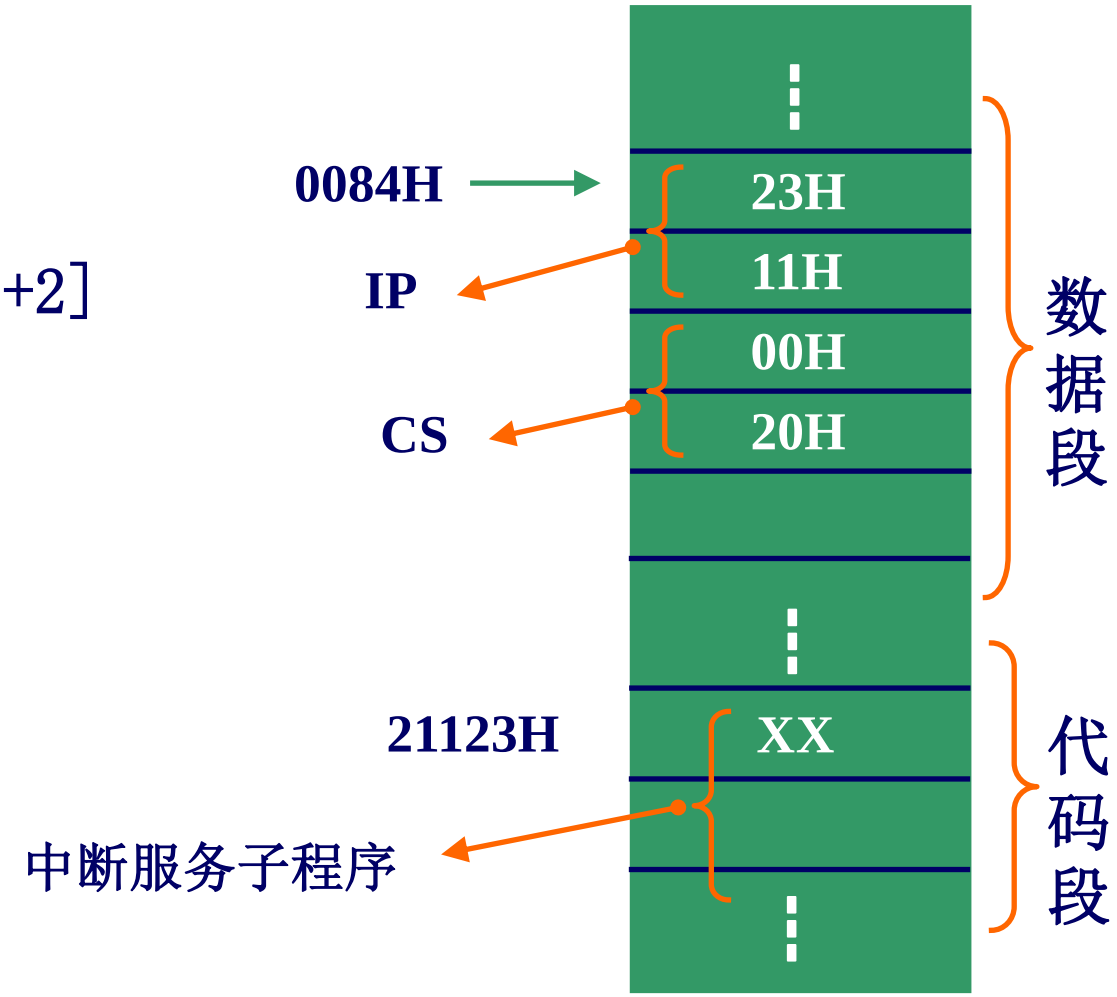
⋮



- 执行INT 21H指令后

  IP=[21H×4]

  CS==[（21H×4）+2]



（2）溢出中断指令

- 格式：

INTO 相当于 INT 4

- 若OF=1，则启动一个类型为4的中断过程，给出一个出错标志，如果OF=0，不做任何操作。
- INTO指令通常安排在有符号数加减运算指令之后。

（3）中断返回指令

- 格式：

  IRET

- 中断服务程序的最后一条指令，负责

  恢复断点

  恢复标志寄存器内容

（6）处理器控制指令

- 对标志位的操作
- 与外部设备的同步

结束语

掌握：

- 指令的格式及意义；
- 指令对操作数的要求及对标志位的影响；
- 指令的应用。





Thank You !