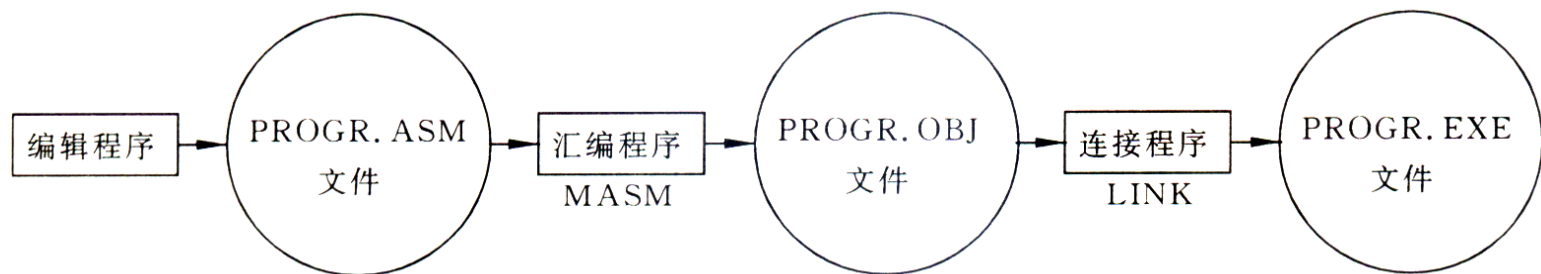


# 第4章 汇编语言及其程序设计

## • 4.1 汇编语言概述

- **指令**：指使计算机完成某种操作的命令。
- **程序**：完成某种功能的指令序列。
- **软件**：各种程序总称。
- **机器语言**：计算机能直接识别的语言。用机器语言写出的程序称为机器代码。
- **汇编语言**：用字符记号代替机器指令。用汇编语言编写的程序叫汇编语言源程序。
- **汇编程序**：一种翻译程序，把助记符翻译成机器语言。
- 在计算机上运行汇编语言程序的步骤：



## 4.2 汇编语言语句格式

- 汇编语句格式有4个字段：

[名字]    操作符    操作数；    [注释]

1. 一种标识符。

2. 组成：

A~Z, a~z;

0~9;

专用符号 ? . @ \_ \$

3. 限制：

1) 第一个字符不能是数字；

2) “.” 必须是第一个字符；

3) 前31个字符有效；

4) 不能为关键字。

4. 类型： 标号-指令符号地址  
          变量-数据符号地址

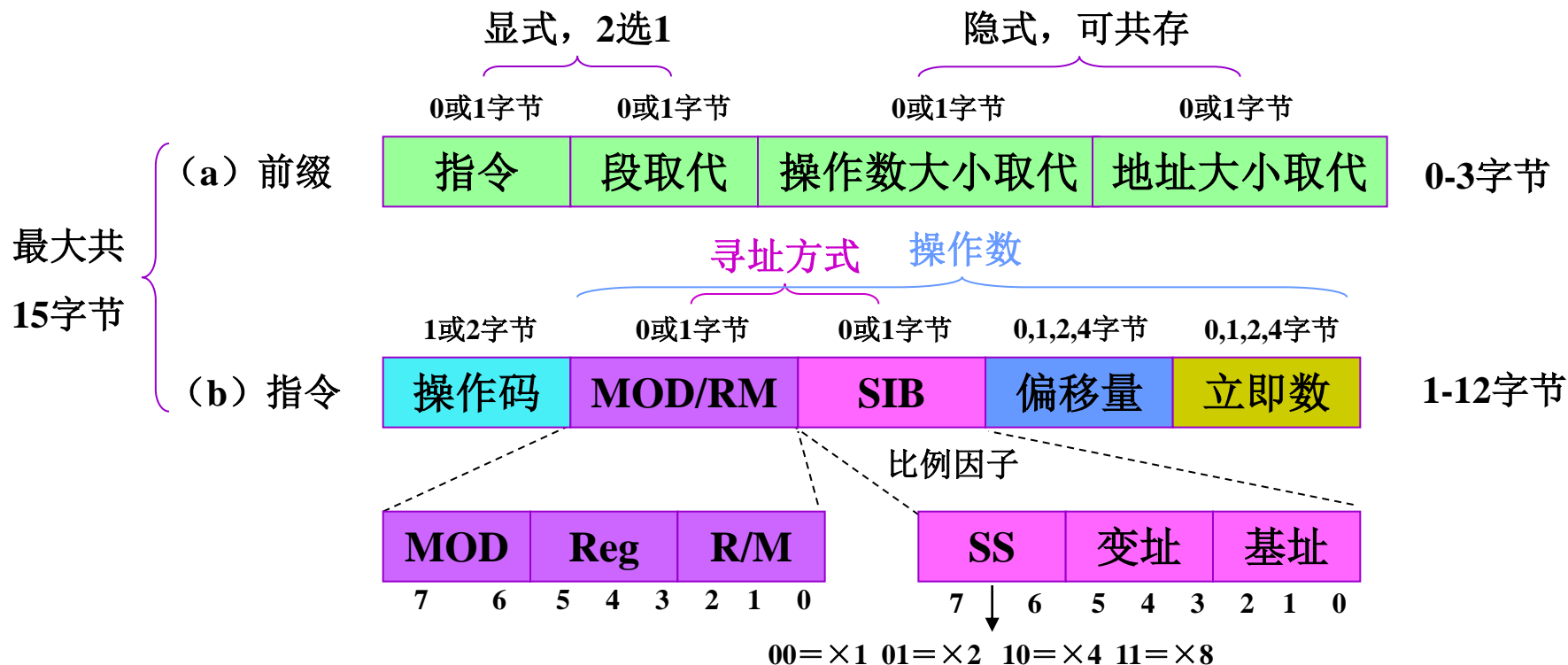
说明程序、指令的功能，  
增加程序的可读性

指定参与操作的数据，  
或数据所在单元地址

组成： CPU指令、伪指令、宏指令

## 4.3 指令格式

- 一条指令由五部分组成：前缀、操作码、寻址方式、偏移量、立即数。



# 1. 前缀

- 前缀：是对指令的某种限定或说明。

4种 {

- 指令前缀：说明指令的相关特性。如LOCK、REP、REPE/REPZ、REPNE/REPZ
- 段超越前缀：用超越段取代默认段。三种情况禁止段超越：串操作（ES）、堆栈指令（SS）、代码段（CS）。
- 操作数大小前缀：改变默认的数据长度。16→32， 32→16。
- 地址大小前缀：改变默认的地址长度。

- 在实地址、虚拟8086方式下，隐含寻址16位。
- 在保护方式下，隐含寻址由段描述符中的D位确定，D=0默认是16位，D=1默认是32位，操作数大小前缀和地址大小前缀用于改变隐含特性。

显式，2选1，由编程人员编写

隐式，可共存，由汇编程序自动加入

0或1字节

0或1字节

0或1字节

0或1字节

(a) 前缀

指令

段取代

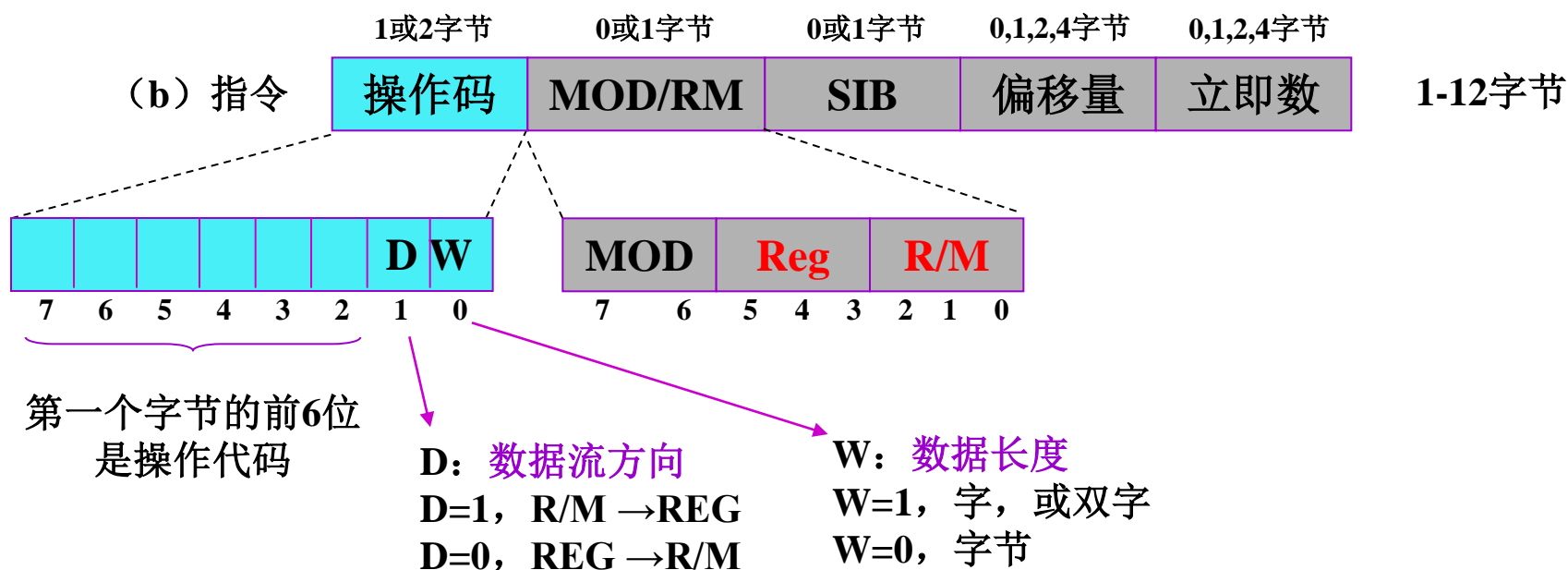
操作数大小取代

地址大小取代

0-3字节

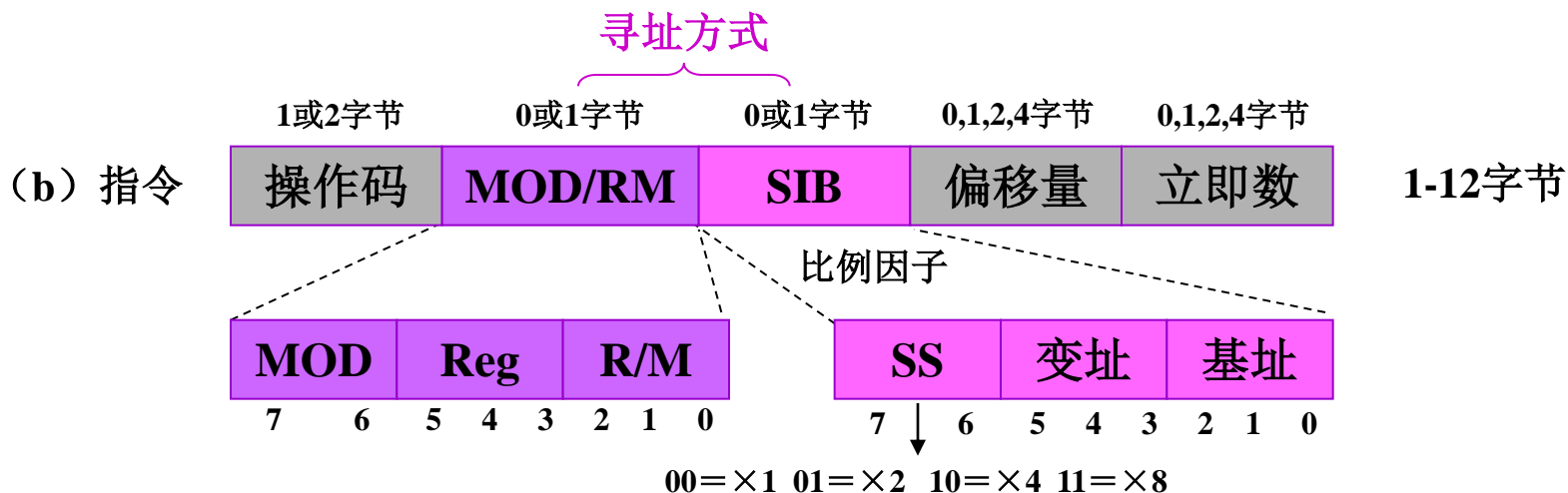
## 2. 操作码

- **操作码**：规定了微处理器执行的操作，如加、减、传送等。
- **Pentium**微处理器指令的操作码长为1或2个字节。第一个字节的前6位是操作码，其余两位中的D位用来指示数据流的方向、W位用来指示数据的长度是字节还是字或双字。



### 3. 寻址方式编码

- 寻址方式：用来指出CPU获取操作数的方式，如寄存器寻址、直接寻址、寄存器间接寻址等。



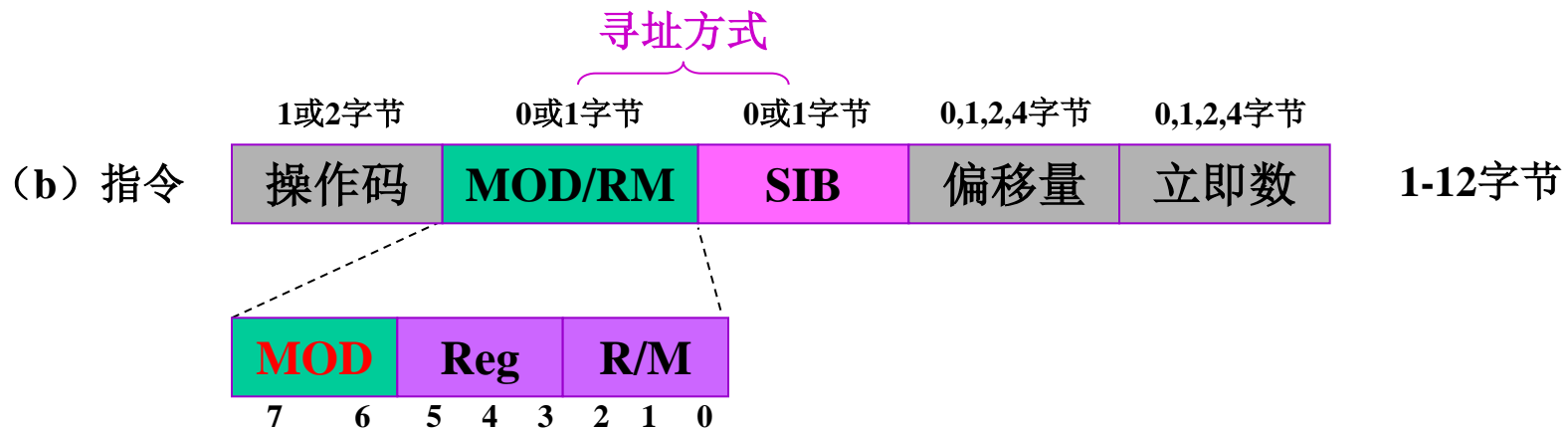
# 1) MOD域

- MOD域**: 规定所选指令的寻址方式, 选择寻址类型及所选的类型是否有位移量。

表 4.3.1 指令模式中的 MOD 域		
模式	16位指令功能	32位指令功能
00	没有位移量	没有位移量
01	8位符号扩展的位移量	8位符号扩展的位移量
10	16位的位移量	32位的位移量
11	R/M是寄存器	R/M是寄存器

数据存储器寻址,  
R/M中选M

寄存器寻址, R/M  
中选R



## 2) REG域

REG或R/M的代码

MOD=11时, R/M中选R

表 4.3.2 REG 域和 R/M 域的分配 (当 MOD = 11 时)

代码	W=0 (字节)	W=1 (字)	W=1 (双字)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI



到底是字还是双字, 依CPU而定, 16位机用字, 32位机用双字。



### 3) R/M域 -16位指令模式的存储器寻址 8088/86、286

表 4.3.3 16 位的 R/M 存储器寻址方式 (MOD≠11时, 用于存储器寻址)

MOD R/M	00 (无位移量)	01 (8位位移量)	10 (16位位移量)
000	(BX)+(SI) DS	(BX)+(SI)+D8 DS	(BX)+(SI)+D16 DS
001	(BX)+(DI) DS	(BX)+(DI) +D8 DS	(BX)+(DI) +D16 DS
010	(BP)+(SI) SS	(BP)+(SI) +D8 SS	(BP)+(SI) +D16 SS
011	(BP)+(DI) SS	(BP)+(DI) +D8 SS	(BP)+(DI) +D16 SS
100	(SI) DS	(SI) +D8 DS	(SI) +D16 DS
101	(DI) DS	(DI) +D8 DS	(DI) +D16 DS
110	D16 (直接寻址) DS	(BP) +D8 SS	(BP) +D16 SS
111	(BX) DS	(BX) +D8 DS	(BX) +D16 DS

## 4. 位移量 / 偏移量及立即数

- **位移量/偏移量**：允许指令中直接给出寻址方式所需的位/偏移量。
- **偏移量**：**直接寻址**的地址，**无符号数**，如 `MOV AX, [2000H]`
- **位移量**：**相对寻址**中的地址数据，**有符号数**，如 `MOV AX, [SI+6]`
- **立即数**：允许指令中直接给出操作数。

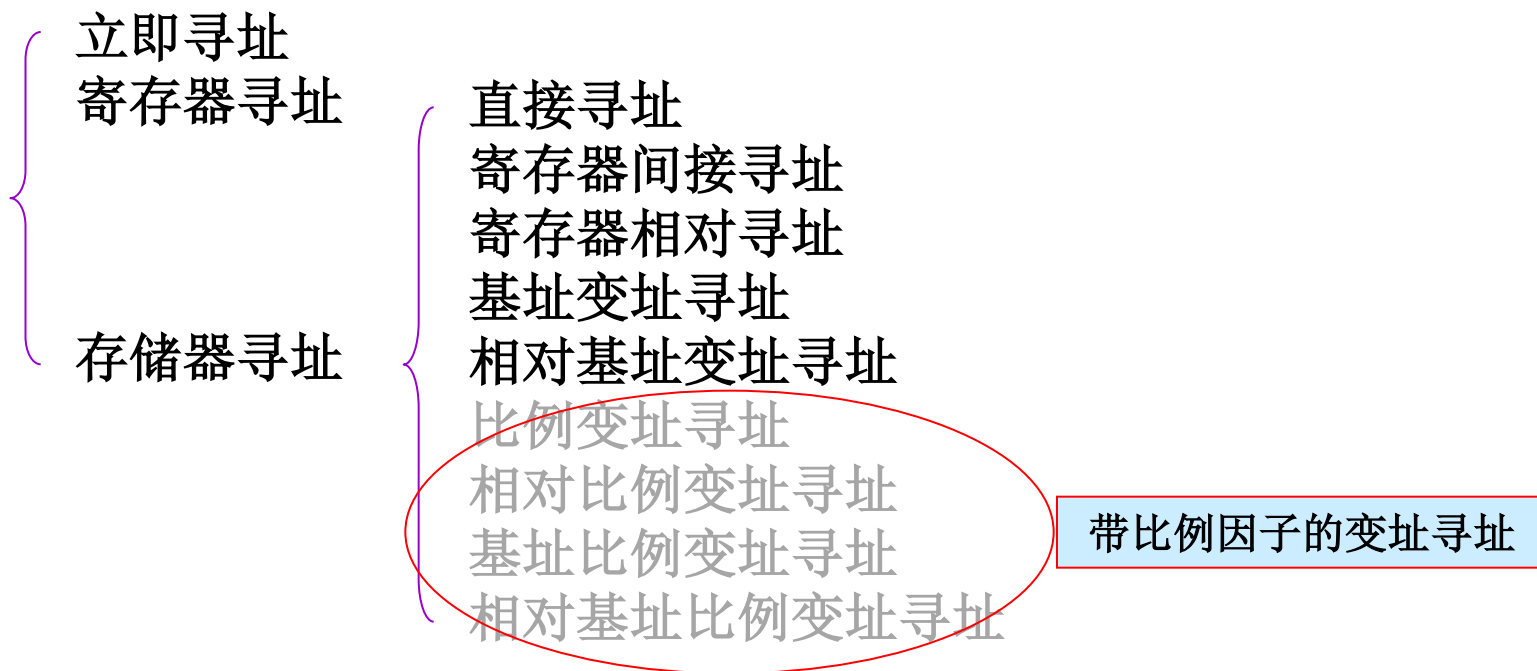


## 4.4 寻址方式

- 一条汇编指令要解决2个问题：
  - （1）做什么
  - （2）对象，或对象来源，即寻址方式
- 寻址方式掌握要点：侧重理解操作数的寻址过程，如何找到一个操作数。
- 有2类寻址：
  - （1）对操作数寻址
  - （2）对转移地址、调用地址寻址

# 数据的寻址方式

- Pentium系统中使用字节（8位）、字（16位）、双字（32位）和4倍字（64位）。采用低端存储方式。
- 与数据有关的寻址方式有11种（如果归纳为“带比例因子的变址寻址”，也可以是8种）：



# 1. 立即寻址

- **立即寻址：**操作数是指令的一部分。完整地取出该条指令之后也就获得了操作数。这种操作数称为立即数。
- 主要用于对寄存器赋值，速度快。
- **汇编语言规定：**立即数必须以数字开头，以字母开头的十六进制数前必须以数字0做前缀；数制用后缀表示，**B**表示二进制数，**H**表示十六进制数，**D**或者缺省表示十进制数，**Q**表示八进制数。
- 汇编程序在汇编时，对于不同进制的立即数一律汇编成等值的二进制数，有符号数以补码表示。
- 立即寻址的例子：

**MOV AL, 01010101B**

**MOV BX, 1234H**

**MOV EAX, 12315678H**

## 2. 寄存器寻址

- **寄存器寻址**：操作数在CPU的某个寄存器中，指令指定寄存器号。
- 对于8位操作数，寄存器有AH、AL、BH、BL、CH、CL、DH和DL；
- 对于16位操作数，寄存器可以是AX、BX、CX、DX、SP、BP、SI和DI；
- 对于32位操作数，寄存器有EAX、EBX、ECX、EDX、ESP、EBP、EDI和ESI。
- 有些用寄存器寻址的MOV、PUSH和POP指令可寻址16位的段寄存器(CS、ES、DS、SS、FS和GS)。
- 寄存器寻址的例子：

**MOV DS, AX**

**MOV EAX, ECX**

**MOV DL, BH**

# 3. 存储器寻址

- 存储器寻址：操作数在存储器中。
- 存储器地址：
  - { 物理地址
  - { 逻辑地址：      段基址:偏移量
- CPU要访问存储器操作数，必须先计算存储器的物理地址。
- 在实地址和保护方式下，段基址的获取方式不同，但偏移量的获取思想基本一致。
- 在存储器寻址方式中，要解决的问题是如何取得操作数的偏移地址。
- 有效地址EA：Effective Address，在8086~Pentium微处理器里，把操作数的偏移地址称为有效地址EA。

# 有效地址的计算方法

- 有效地址的计算公式：

$$EA = \text{基址} + (\text{变址} \times \text{比例因子}) + \text{位移量}$$

- 基址**：基址寄存器中的内容。基址寄存器通常用于编译程序指向局部变量区、数据段中数组或字符串的**首地址**。
- 变址**：变址寄存器中的内容。变址寄存器用于**访问**数组或字符串的**元素**。
- 比例因子**：32位机的寻址方式。寻址中，用变址寄存器的内容乘以比例因子得到变址值。该方式对访问数组特别有用。
- 位移量**：指令中的一个数，但不是立即数，而是一个地址。
- 8086/286是16位寻址，80386及以后机型是32位寻址，也可用16位寻址。

四种成分	16位寻址	32位寻址
位移量	0、8、16位	0、8、32位
基址寄存器	BX、BP	任何32位通用寄存器
变址寄存器	SI、DI	除ESP以外的任何32位通用寄存器
比例因子	无	1、2、4、8



# 表 4.4.1 存储器寻址方式

寻址方式	基址	变址	比例因子	位移量
1. 直接寻址				✓
2. 寄存器间接寻址	✓*	✓*		
3. 寄存器相对寻址	✓*	✓*		✓
4. 基址变址寻址	✓	✓		
5. 相对基址变址寻址	✓	✓		✓
6. 比例变址寻址		✓	✓	
7. 相对比例变址寻址		✓	✓	✓
8. 基址比例变址寻址	✓	✓	✓	
9. 相对基址比例变址寻址	✓	✓	✓	✓

一对，  
寻址  
及  
相对寻址

四种成分	16位寻址	32位寻址
位移量	0、8、16位	0、8、32位
基址寄存器	BX、BP	任何32位通用寄存器
变址寄存器	SI、DI	除ESP以外的任何32位通用寄存器
比例因子	无	1、2、4、8

任取一个

# 存储器寻址方式举例

寻址方式	有效地址EA如何计算?	基址	变址	比例因子	位移量
1. 直接寻址	MOV BX, [1234H]				✓
2. 寄存器间接寻址	MOV AL, [BX]	✓*	✓*		
3. 寄存器相对寻址	MOV BX, [SI+40H]	✓*	✓*		✓
4. 基址变址寻址	MOV DX, [BX+SI]	✓	✓		
5. 相对基址变址寻址	MOV AX, [BP+DI+2]	✓	✓		✓
6. 比例变址寻址	MOV EAX, [4×ECX]		✓	✓	
7. 相对比例变址寻址	MOV EDX, [4×ECX+5]		✓	✓	✓
8. 基址比例变址寻址	MOV AL, [EBX+2×EDI]	✓	✓	✓	
9. 相对基址比例变址寻址	MOV AL, [EBX+2×EDI-2]	✓	✓	✓	✓

四种成分	16位寻址	32位寻址
位移量	0、8、16位	0、8、32位
基址寄存器	BX、BP	任何32位通用寄存器
变址寄存器	SI、DI	除ESP以外的任何32位通用寄存器
比例因子	无	1、2、4、8

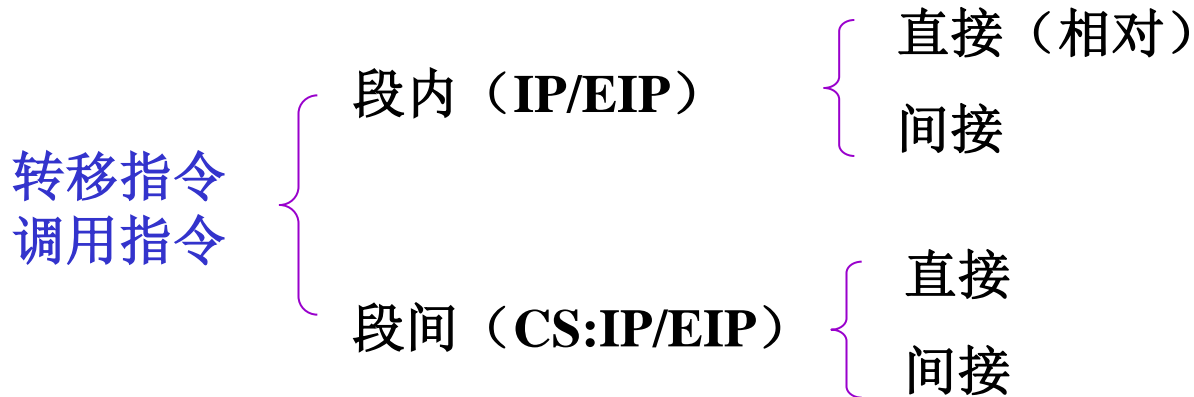
# 寻址方式的段约定和段超越

- **MOV AL, DS: [BP]** ; 用BP间址访问数据段
- **MOV AL, ES: [BP]** ; 用BP间址访问ES附加段
- **MOV AL, FS: [EBP]** ; 用EBP间址访问FS附加段
- **MOV AL, CS: [BX]** ; 用BX间址访问代码段
- **MOV AL, ES: [SI+5]** ; 用SI变址寻址, 访问ES附加段
- **MOV AL, GS: [EAX+10]**; 用EAX基址寻址, 访问GS附加段
- 为提高程序的可读性和效率, 要尽量少使用段超越。

存储器操作类型	隐含段	超越段	段内偏移量
取指令	CS	无	IP
堆栈操作	SS	无	SP
数据变量	DS	CS、SS、ES	有效地址EA
源串变量	DS	CS、SS、ES	SI
目的串变量	ES	无	DI
基址BP指针	SS	CS、SS、ES	有效地址EA

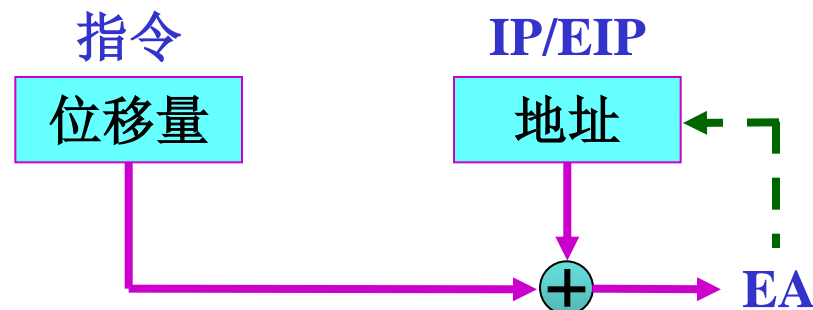
## 4. 转移地址的寻址方式

- **转移地址寻址**：用于确定转移指令和CALL指令的转向地址。
- 转移指令使程序不再顺序执行，而是按指令中给出的操作数转移到相应的目的地址。
- **转移地址**：转移指令中的操作数是**转移的目的地址**，称为转移地址。



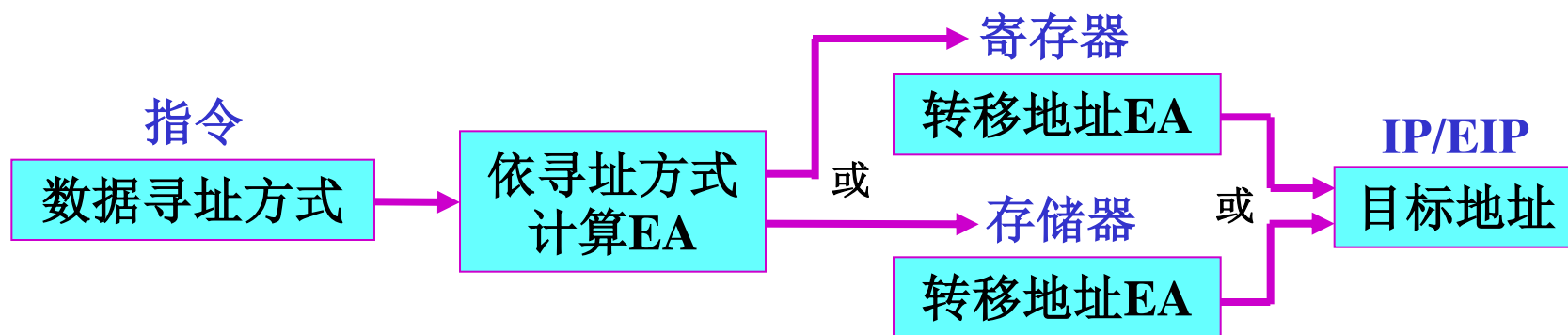
# 1) 段内相对寻址

- 位移量+IP=EA→IP/EIP
  - 位移量是相对值，所以称为段内相对寻址。
  - 因为是相对值，所以便于程序再定位。
  - 用于条件转移、无条件转移，条件转移只能用此方式。
- 短跳转: **JMP SHORT AA9** ; 位移量 8位, -128~+127
- 近跳转: **JMP NEAR PTR AA8** ; 位移量 16位±32K , 32位±2G



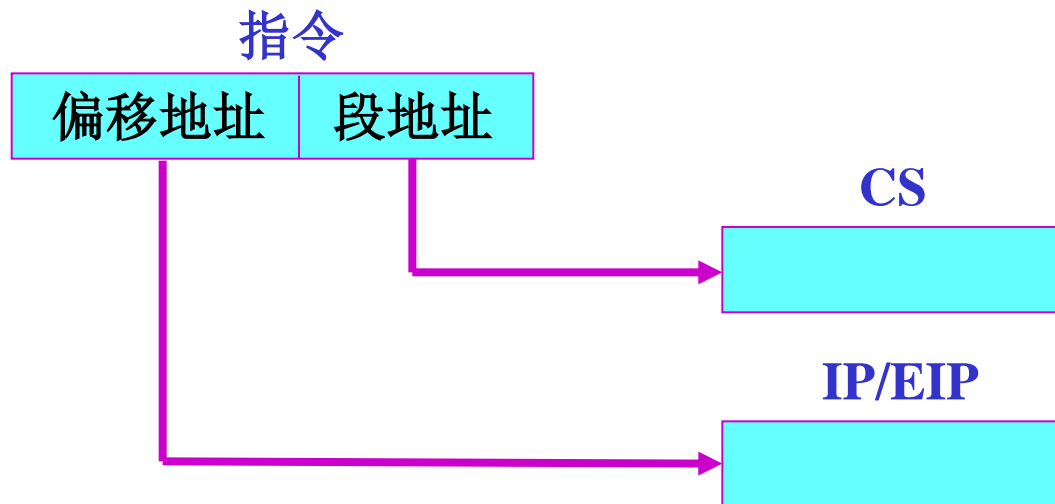
## 2) 段内间接寻址

- 寄存器（存储器）内容=EA→IP/EIP
- 这种寻址方式不能用于条件转移指令。
- **JMP BX**
- **JMP WORD PTR[BX+AA7]**
- **JMP ECX**



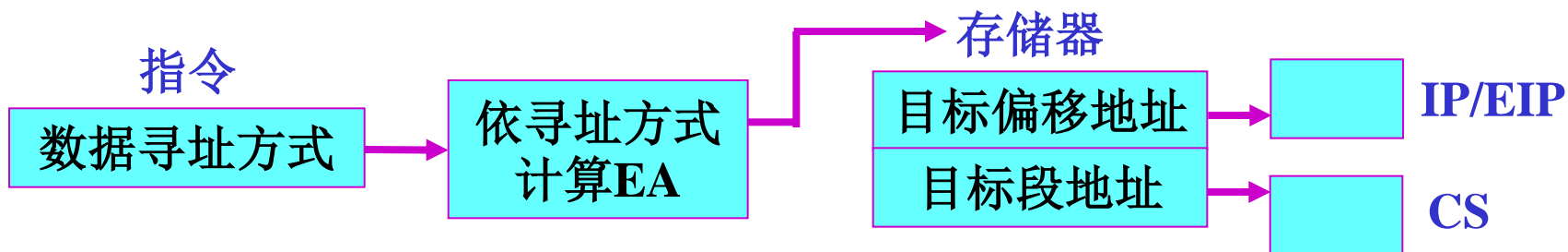
### 3) 段间直接寻址

- 目标段地址:偏移量=CS:IP/EIP
- **JMP FAR PTR AA6**



## 4) 段间间接寻址

- 连续的存储器单元内容= **CS:IP/EIP**
- **JMP DWORD PTR [SI]** ; [SI]指向的字送入IP, [SI+2] 指向的字送入CS。





## 5) 堆栈地址寻址

- **堆栈：**以“先进后出”方式工作的一个特定的存储区。
- 一端固定，称为栈底；另一端浮动，称为栈顶，只有一个出入口。
- **堆栈作用：**保存传递参数、现场参数、寄存器内容、返回地址。
- **80x86规定：**
  - (1) 堆栈向小地址方向增长。
  - (2) 必须使用堆栈段SS。
  - (3) **PUSH**指令压入数据时，先修改指针，再按照指针指示的单元存入数据。
  - (4) **POP**指令弹出数据时，先按照指针指示的单元取出数据，再修改指针。
  - (5) 压入和弹出的数据类型（数据长度）不同，堆栈指针修改的数值也不同。
- 16位堆栈地址操作，还是32位堆栈地址操作，由堆栈段SS寄存器内的数据段描述符的属性说明，确定用SP还是ESP，是一种隐含属性。

## 4.5 指令系统

- **指令系统：**指令的集合。软硬件交界面。
- **指令系统掌握要点：**侧重掌握指令的功能和使用方法，为后续程序设计打基础。
- **Pentium指令系统按功能分为10类：**

数据传送指令、算术运算指令、BCD 码调整指令、逻辑运算指令、移位循环指令、控制转移指令、条件设置指令、串操作指令、处理器控制指令、保护模式系统控制指令。

## 4.5.1 数据传送指令

- 数据传送指令功能：源 → 目。
- 指令执行后，源操作数不变，不影响状态标志。
- 1. MOV 传送 (move)
- 格式：MOV DST, SRC ; (DST) ← (SRC)
- 式中DST (Destination) 为目的操作数，SRC (Source) 为源操作数。
- 源、目长度相等。CS可出不可入。
- MOV WORD PTR [BX], 10H; 立即数10H送入BX指向的字存储单元

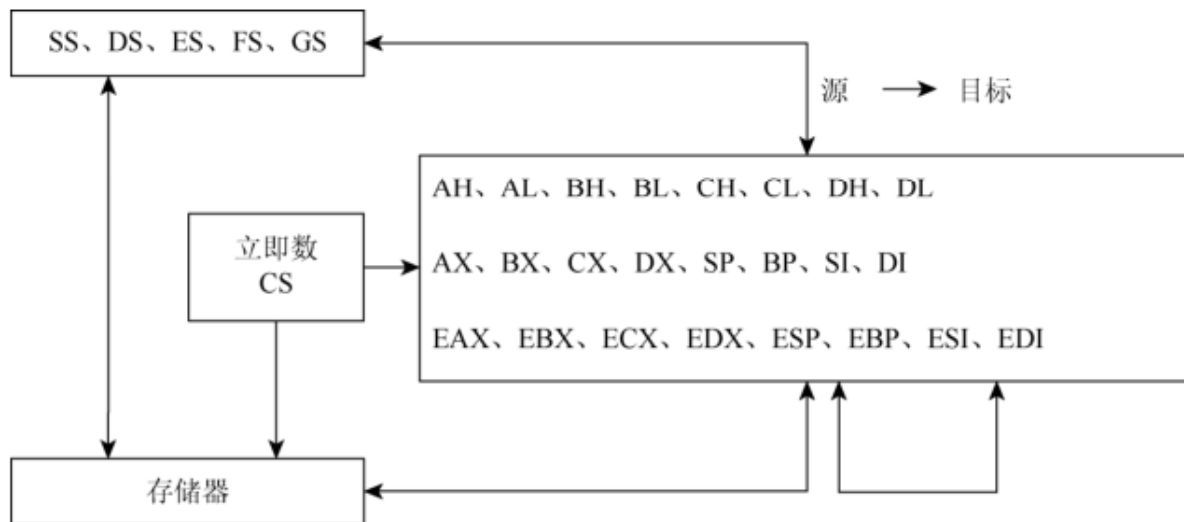


图 4.5.1 数据传送示意图

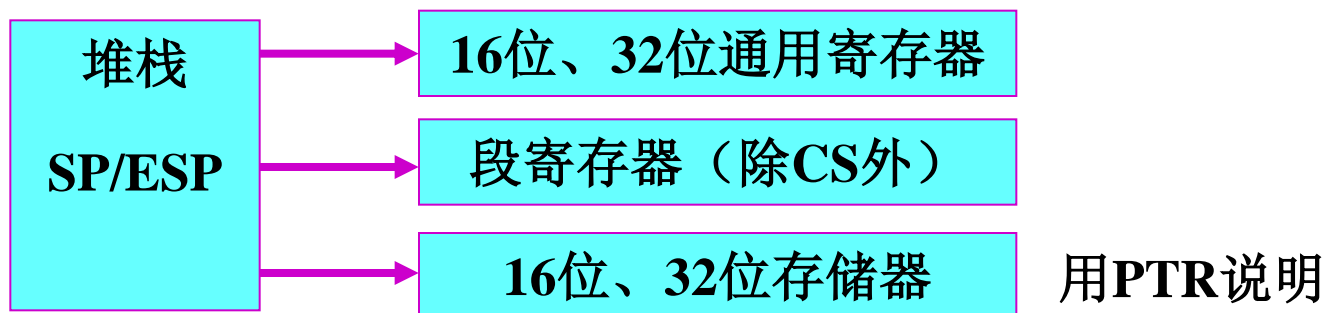
# PUSH

- **2. PUSH 进栈** (push onto stack)
- **格式:** PUSH SRC ; 堆栈  $\leftarrow$  (SRC)
- **操作:**
  - 16位指令:  $(SP) \leftarrow (SP) - 2$   
 $((SP)+1, (SP)) \leftarrow (SRC)$
  - 32位指令:  $(ESP) \leftarrow (ESP) - 4$   
 $((ESP)+3, (ESP)+2, (ESP)+1, (ESP)) \leftarrow (SRC)$
- **类型:**



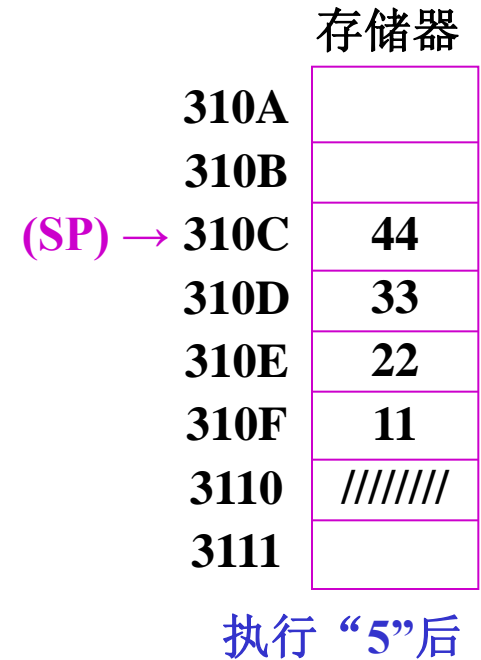
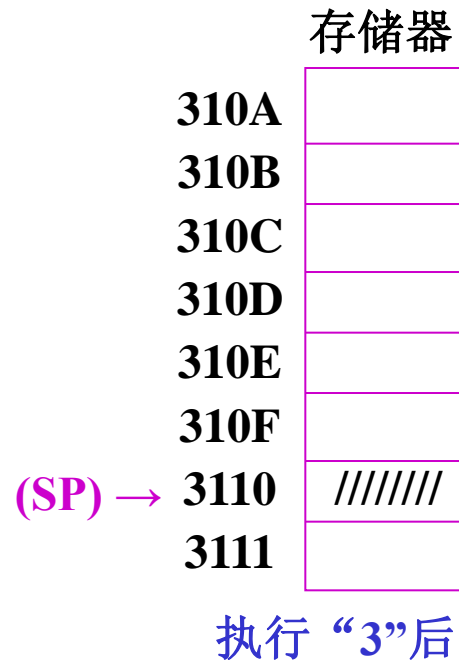
# POP

- **3. POP 出栈** (pop from the stack)
- **格式:** POP DST ; (DST)  $\leftarrow$  堆栈栈顶内容
- **操作:**
  - 16位指令: (DST)  $\leftarrow$  ( (SP)+1, (SP) )  
(SP)  $\leftarrow$  (SP)+2
  - 32位指令: (DST)  $\leftarrow$  ( (ESP)+3, (ESP)+2, (ESP)+1, (ESP) )  
(ESP)  $\leftarrow$  (ESP)+4
- **类型:**



# 堆栈指令举例

- 1 MOV AX, 1122H
- 2 MOV BX, 3344H
- 3 MOV SP, 3110H
- 4 PUSH AX
- 5 PUSH BX
- 6 MOV AX, 0
- 7 MOV BX, 0
- 8 POP BX
- 9 POP AX



- 问题：1) 运行结果    2) 进、出栈平衡    3) 交换

# XCHG

- 4. PUSHF/POPF 16位标志寄存器进栈/出栈

(push the flags/pop the flags)

- 格式:

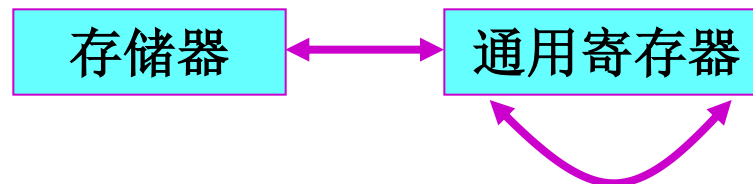
**PUSHF** ; 标志寄存器低16位压入堆栈

**POPF** ; 从栈顶弹出2个字节送标志寄存器低16位

- 5. XCHG 交换 (exchange)

- 格式: **XCHG OPR1, OPR2** ; (OPR1)  $\longleftrightarrow$  (OPR2)

- 类型:



# LEA

- **6. LEA 有效地址送寄存器** (load effective address)
- **格式:** LEA REG, SRC ; (REG)  $\leftarrow$  (SRC)的有效地址
- **类型:**

16位、32位通用寄存器

16位、32位存储器有效地址

- **操作规则:**

操作数长度 (寄存器)	地址长度 (EA)	操作规则
16	16	16位EA $\rightarrow$ 16位REG
16	32	32位EA的低16位 $\rightarrow$ 16位REG
32	16	16位EA, 零扩展 $\rightarrow$ 32位REG
32	32	32位EA $\rightarrow$ 32位REG

1 MOV AX, MEM ;AX=3412H

2 LEA AX, MEM ;AX=3100H

3 MOV BX, 2000H

4 MOV DS, BX

5 MOV SI, 1000H

6 MOV AX, [BX+SI+102H] ; AX=7856H

7 LEA BX, [BX+SI+102H] ; BX=3102H

MEM 2000:3100

12H

3101

34H

3102

56H

3103

78H

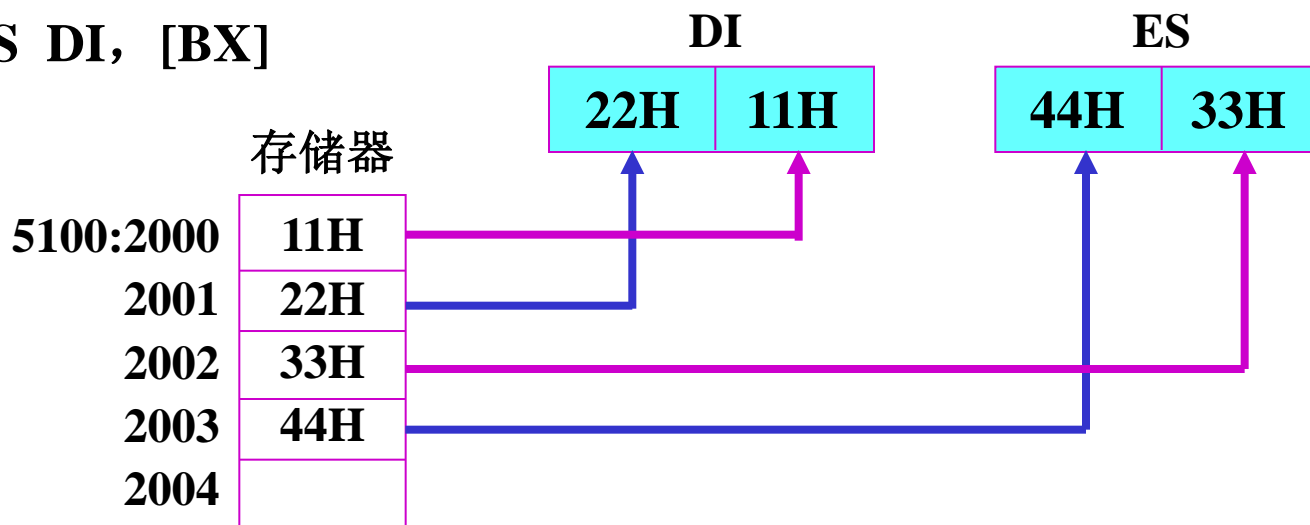
3104



# LDS、LES、LFS、LGS和LSS

- 7. LDS、LES、LFS、LGS和LSS 指针送寄存器和段寄存器  
(load ds with pointer)
- 格式: **LDS REG, SRC**
- 功能:  $(REG) \leftarrow [SRC]$   
 $(DS) \leftarrow [SRC+2]$  (16位EA)  
 $(DS) \leftarrow [SRC+4]$  (32位EA)
- 例: 已知  $(DS)=5100H$ ,  $(BX)=2000H$

**LES DI, [BX]**



# XLAT

- **8. XLAT 换码** (translate)
- 格式: XLAT
- 功能: 完成代码转换, 偏移量8位, 表长 $\leq 256$

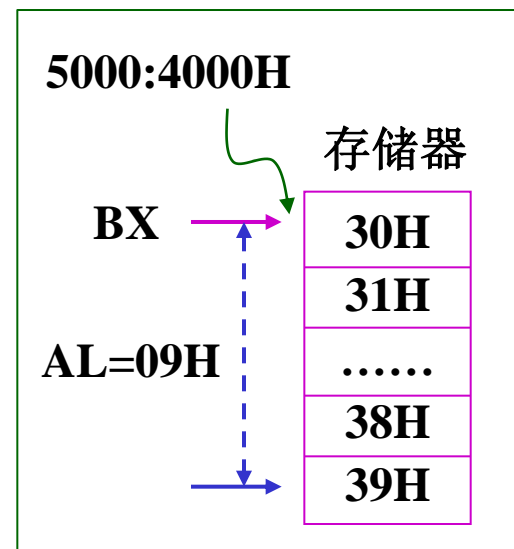
16位指令  $(AL) \leftarrow [(BX) + (AL)]$

32位指令  $(AL) \leftarrow [(EBX) + (AL)]$

- 例: 非压缩BCD码转换成ASCII码,  $09H \rightarrow 39H$

已知  $(DS) = 5000H$ ,  $(BX) = 4000H$ ,  $(AL) = 09H$

执行      XLAT      ; AL = 39H



# IN

- 9. IN/OUT 输入/输出 (不需要段地址)

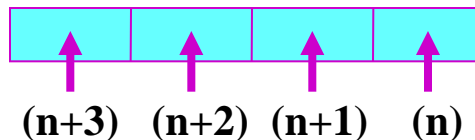
- (1) IN 输入指令 (input)

- 1) 直接寻址 (端口地址  $\leq 255$ )

IN AL, n ; 字节,  $(AL) \leftarrow (n)$

IN AX, n ; 字,  $(AH) \leftarrow (n+1)$ ,  $(AL) \leftarrow (n)$

IN EAX, n ; 双字, EAX

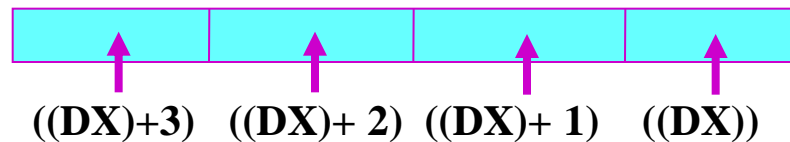


- 2) 间接寻址 (全部端口地址可用)

IN AL, DX ; 字节,  $(AL) \leftarrow ((DX))$

IN AX, DX ; 字,  $(AH) \leftarrow ((DX+1))$ ,  $(AL) \leftarrow ((DX))$

IN EAX, DX ; 双字, EAX



# OUT

- (2) **OUT 输出指令** (output)

- 1) **直接寻址** (端口地址 $\leq 255$ )

**OUT n, AL** ; 字节,  $(n) \leftarrow (AL)$

**OUT n, AX** ; 字

**OUT n, EAX** ; 双字

- 2) **间接寻址** (全部端口地址可用)

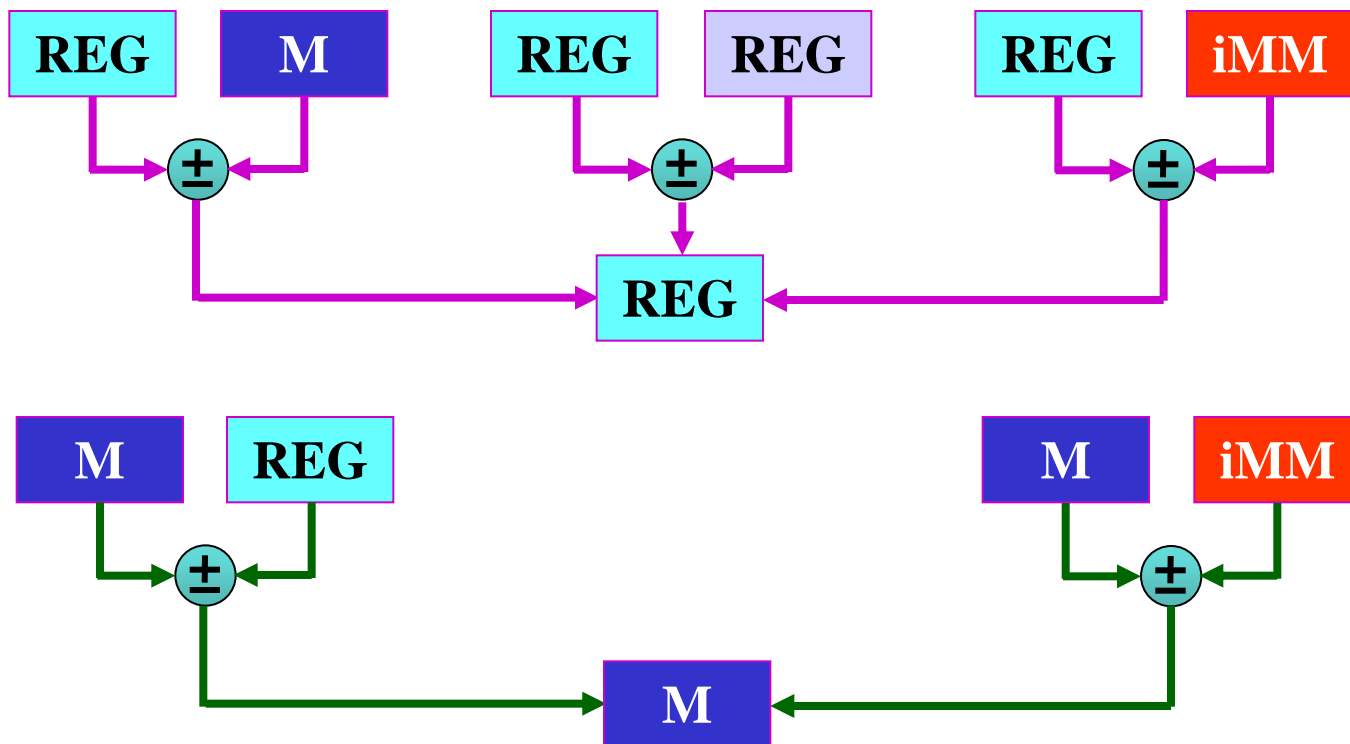
**OUT DX, AL** ; 字节,  $((DX)) \leftarrow (AL)$

**OUT DX, AX** ; 字

**OUT DX, EAX** ; 双字

## 4.5.2 算术运算指令

- 1. 加法、减法指令
- 1) **ADD** 加法 (add)
- 格式: **ADD DST, SRC** ;  $(DST) \leftarrow (SRC) + (DST)$
- 类型: 影响CF、PF、AF、ZF、SF、OF。可进行8、16、32位操作。



# SUB

- **2) ADC 带进位加法** (add with carry)
- **格式:** ADC DST, SRC ;  $(DST) \leftarrow (SRC) + (DST) + CF$
- **3) SUB 减法** (subtract)
- **格式:** SUB DST, SRC ;  $(DST) \leftarrow (DST) - (SRC)$
- **4) SBB 带借位减法** (subtract with borrow)
- **格式:** SBB DST, SRC ;  $(DST) \leftarrow (DST) - (SRC) - CF$
- **例:**

MOV AX, 7626H

MOV BX, 6615H

ADD AX, BX ; AX=0DC3BH

CF=? PF=? AF=? ZF=? SF=? OF=?

000011

# INC、DEC

- 2. 加 1、减 1 指令
- 1) INC 加1 (increment)
- 格式: INC OPR ;  $(\text{OPR}) \leftarrow (\text{OPR}) + 1$
- 2) DEC 减1 (decrement)
- 格式: DEC OPR ;  $(\text{OPR}) \leftarrow (\text{OPR}) - 1$
- 2条指令共性说明:
  - 1) 目的操作数: 通用寄存器、存储器单元, 可以是8位、16位或32位。
  - 2) 执行INC、DEC指令后, 影响AF、OF、PF、SF、ZF, 但对CF没有影响。

# CMP

- **3. CMP 比较 (compare)**

- **格式:** `CMP OPR1, OPR2 ; (OPR1) - (OPR2)`

- **4. MUL 无符号数乘法 (unsigned multiple)**

- **格式:** `MUL SRC`

- **功能:**

字节乘  $(AX) \leftarrow (AL) \times (SRC)$

字乘  $(DX:AX) \leftarrow (AX) \times (SRC)$

双字乘  $(EDX:EAX) \leftarrow (EAX) \times (SRC)$

- **说明:**

1) 乘数和被乘数必须等长。

2) **SRC**: 通用寄存器、存储器单元。

3) 被乘数: **默认AL、AX、EAX**之一。

4) 若乘积的高半部分结果为0, 则**CF=0、OF=0**; 否则, **CF=1、OF=1**。  
而**AF、PF、ZF、SF**无定义, 即状态不定。



# DIV

- **5. DIV 无符号数除法** (unsigned divide)
- **格式:** DIV SRC
- **功能:** 字节除  $(AL) \leftarrow (AX)/(SRC)$  的商,  $(AH) \leftarrow (AX)/(SRC)$  的余数  
字除  $(AX) \leftarrow (DX:AX)/(SRC)$  的商,  $(DX) \leftarrow (DX:AX)/(SRC)$  的余数  
双字除  $(EAX) \leftarrow (EDX:EAX)/(SRC)$  的商,  $(EDX) \leftarrow (EDX:EAX)/(SRC)$  的余数
- **说明:** 被除数默认在AX、DX:AX、EDX:EAX中。SRC是通用寄存器、存储器单元。所有标志位无定义。

## 4.5.3 BCD 码调整指令

压缩BCD码

7	4	3	0
BCD		BCD	

非压缩BCD码

7	4	3	0
0000		BCD	

- **1. DAA 压缩BCD码加法调整** (decimal adjust for addition)
- 格式: DAA
- 功能:
  - 1) 如果AL的低4位大于9或AF=1, 则(AL)+6 → (AL) 和 1→AF。
  - 2) 如果AL的高4位大于9或CF=1, 则(AL)+60H → (AL) 和 1→CF。
- 说明:
  - 1) 只对AL调整。
  - 2) OF无定义, 其他5个状态受影响。
- 例
- ```
MOV AL, 54H ; 54H代表十进制数54
MOV BL, 37H ; 37H代表十进制数37
ADD AL, BL  ; AL中的和为十六进制数8BH
DAA          ; AL=91H, AF=1, CF=0
```

# AAA

- **2. DAS 压缩BCD码减法调整** (decimal adjust for subtraction)
- 格式: DAS
- 功能:
  - 1) 如果AL的低4位大于9或AF=1, 则 $(AL)-6 \rightarrow (AL)$  和  $1 \rightarrow AF$ 。
  - 2) 如果AL的高4位大于9或CF=1, 则 $(AL)-60H \rightarrow (AL)$  和  $1 \rightarrow CF$ 。
- **3. AAA 非压缩BCD码加法调整** (ASCII adjust for addition)
- 格式: AAA
- 功能:
  - ① 如果AL的低4位小于等于9, 并且AF=0, 则转步骤③;
  - ②  $(AL)+6 \rightarrow AL$ ,  $1 \rightarrow AF$ ,  $(AH)+1 \rightarrow AH$ ;
  - ③ AL高4位清0;
  - ④  $AF \rightarrow CF$ 。
- 说明: 影响AF和CF, 而PF、SF、ZF、OF无定义。

# AAS

- **4. AAS 非压缩BCD码减法调整** ( ASCII adjust for subtraction )
- 格式: AAS
- 功能:
  - ① 如果AL的低4位小于等于9, 并且AF=0, 则转步骤③;
  - ②  $(AL)-6 \rightarrow AL$ ,  $1 \rightarrow AF$ ,  $(AH)-1 \rightarrow AH$ ;
  - ③ AL高4位清0;
  - ④  $AF \rightarrow CF$ 。

## 4.5.4 逻辑运算指令

- 1. 与、或、异或、比较逻辑运算指令
- 1) **AND** 按位逻辑与运算
- 格式: **AND DST, SRC** ;  $(DST) \leftarrow (DST) \wedge (SRC)$
- 2) **OR** 按位逻辑或运算
- 格式: **OR DST, SRC** ;  $(DST) \leftarrow (DST) \vee (SRC)$
- 3) **XOR** 按位逻辑异或运算 (exclusive or)
- 格式: **XOR DST, SRC** ;  $(DST) \leftarrow (DST) \oplus (SRC)$
- 4) **TEST** 按位逻辑比较运算
- 格式: **TEST OPR1, OPR2** ;  $(OPR1) \wedge (OPR2)$
- 4条指令类型: 同ADD。
- 4条指令说明: **CF、OF置0**, 影响SF、ZF、PF, AF无定义。

# NOT

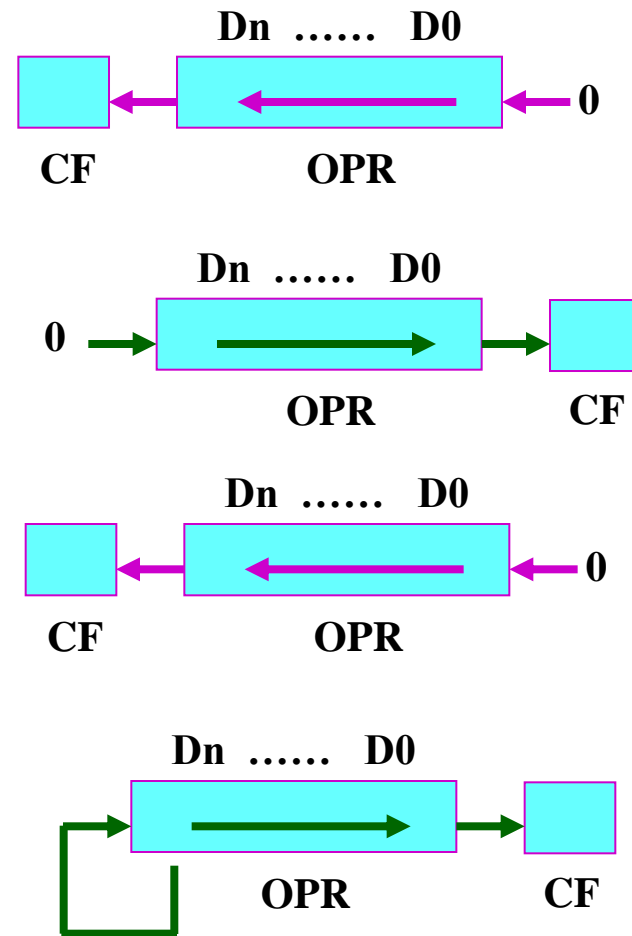
- 2. NOT 按位取反指令
- 格式: NOT OPR ;  $(\text{OPR}) \leftarrow \overline{\text{OPR}}$
- 类型: 通用寄存器、存储器单元, 8位、16位、32位。
- 说明: 不影响标志位。

## 4.5.5 移位循环指令

- 1. 移位指令
- 1) **SHL** 逻辑左移 (shift logical left)
- 格式: **SHL OPR, CNT**
- 2) **SHR** 逻辑右移 (shift logical right)
- 格式: **SHR OPR, CNT**
- 3) **SAL** 算术左移 (shift arithmetic left)
- 格式: **SAL OPR, CNT**
- 4) **SAR** 算术右移 (shift arithmetic right)
- 格式: **SAR OPR, CNT**

- 移位指令说明:

- 1) **OPR**: 通用寄存器、存储器单元。8位、16位、32位。
- 2) **CNT**: 8位立即数 (1-31, **8086中只能为1**)、CL。
- 3) **CF**依操作设置。**OF**当**CNT=1**时有效。



# 移位指令举例

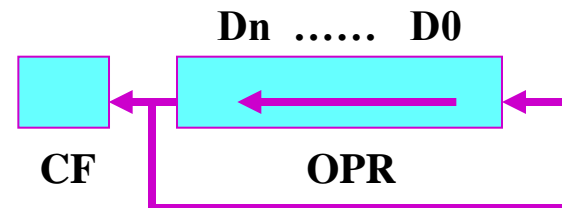
- 1)  $AL \times 10$ 
  - **MOV AH, 0** ; AL扩展为字, 无符号
  - **SAL AX, 1** ;  $2X$
  - **MOV BX, AX**
  - **SAL AX, 1** ;  $4X$
  - **SAL AX, 1** ;  $8X$
  - **ADD AX, BX** ;  $8X+2X$
- 2) ASCII码 $\leftarrow$ AL组合BCD
  - **MOV AH, AL** ; AL组合BCD, 保存
  - **AND AL, 0FH**
  - **OR AL, 30H** ; 低位ASCII码在AL中
  - **MOV CL, 4**
  - **SHR AH, CL**
  - **OR AH, 30H** ; 高位ASCII码在AH中。AX为2个ASCII码



## 2. 循环指令

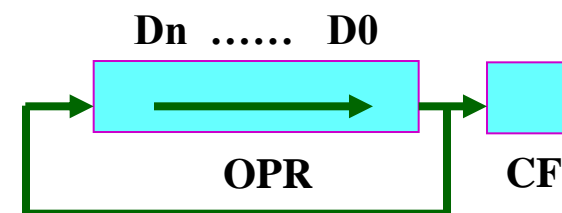
- 1) **ROL** 循环左移 (rotate left)

- 格式: **ROL** OPR, CNT



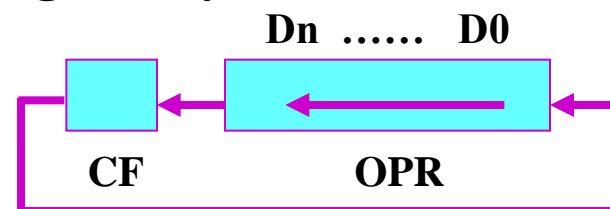
- 2) **ROR** 循环右移 (rotate right)

- 格式: **ROR** OPR, CNT



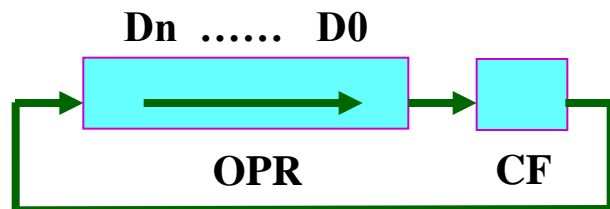
- 3) **RCL** 带进位循环左移 (rotate left through carry)

- 格式: **RCL** OPR, CNT



- 4) **RCR** 带进位循环右移 (rotate right through carry)

- 格式: **RCR** OPR, CNT



## 4.5.6 控制转移指令

- 1. 无条件段内相对短转移 (jump)
- 格式: **JMP SHORT OPR**
- 功能:  $(IP) \leftarrow (IP) + 8\text{位位移量}$   
或  $(EIP) \leftarrow (EIP) + 8\text{位位移量}$
- 说明: **SHORT**是短转移属性描述符, 8位位移量是一个带符号数, 范围是-128到+127字节。
- 2. 无条件段内相对近转移
- 格式: **JMP NEAR PTR OPR**
- 功能:  $(IP) \leftarrow (IP) + 16\text{位位移量}$   
或  $(EIP) \leftarrow (EIP) + 32\text{位位移量}$
- 说明: **NEAR**是近转移属性描述符, 16位、32位位移量是一个带符号数。

# 3. 无条件段内间接转移

- 3. 无条件段内间接转移

- 格式: **JMP OPR** ; OPR可以是寄存器、存储器
- 功能:  $(IP) \leftarrow (EA)$  或  $(EIP) \leftarrow (EA)$

- 4. 无条件段间直接转移

- 格式: **JMP FAR PTR OPR** ; OPR为指令的标号（即指令地址）
- 功能:  $(IP/EIP) \leftarrow OPR$ 的段内偏移地址  
 $(CS) \leftarrow OPR$ 所在段的段地址

- 5. 无条件段间间接转移

- 格式: **JMP DWORD PTR OPR** ; OPR为存储器单元
- 功能:  $(IP/EIP) \leftarrow [EA]$   
 $(CS) \leftarrow [EA+2]/[EA+4]$

## 6. 条件相对转移

- 6. 条件相对转移 (8位偏移量)

- 1) 单条件相对转移

① JZ / JE OPR ; ZF=1 转移(结果为零或相等转移)

② JNZ / JNE OPR ; ZF=0 转移(结果不为零或不相等转移)

③ JS OPR ; SF=1 转移(结果为负转移)

④ JNS OPR ; SF=0 转移(结果为正转移)

⑤ JO OPR ; OF=1 转移(结果溢出转移)

⑥ JNO OPR ; OF=0 转移(结果不溢出转移)

⑦ JP / JPE OPR ; PF=1 转移(结果为偶转移)

⑧ JNP / JPO OPR ; PF=0 转移(结果为奇转移)

⑨ JC OPR ; CF=1 转移(有借位或有进位转移)

⑩ JNC OPR ; CF=0 转移(无借位或无进位转移)

- J:jump Z:zero E:equal N:not S:sign P:parity C:carry  
O:overflow PE:parity even PO:parity odd

# 无符号数、有符号数比较条件

- 2) 无符号数比较条件相对转移(A-B) ( A:above高于 B:below低于)
  - ① JB / JNAE / JC OPR ; CF=1 (A<B转移)
  - ② JAE / JNB / JNC OPR ; CF=0 (A≥B转移)
  - ③ JBE / JNA OPR ; (CF ∨ ZF)=1 (A≤B转移)
  - ④ JA / JNBE OPR ; (CF ∨ ZF)=0 (A>B转移)
- 3) 有符号数比较条件相对转移(A-B) ( G:greater大于 L:less小于)
  - ① JL / JNGE OPR ; (SF ⊕ OF)=1 (A<B转移)
  - ② JGE / JNL OPR ; (SF ⊕ OF)=0 (A≥B转移)
  - ③ JLE / JNG OPR ; ((SF ⊕ OF) ∨ ZF)=1 (A≤B转移)
  - ④ JG / JNLE OPR ; ((SF ⊕ OF) ∨ ZF)=0 (A>B转移)

# LOOP

- 7. LOOP 循环控制相对转移
- 格式: **LOOP OPR**
- 功能:  $(CX/ECX) \leftarrow (CX/ECX) - 1; (CX/ECX) \neq 0$  转移
- 说明: 上述循环控制相对转移指令中的转移地址为 (IP/EIP)  
 $\leftarrow (IP/EIP) + 8$  位带符号数; 8 位位移量是由目标地址 OPR 确定的。

# 8. 子程序调用与返回

- 8. 子程序调用与返回
- 子程序：具有独立功能的程序模块。
- 1) 段内相对调用
- 格式：CALL DST ； DST为直接入口地址，指令中给出  
CALL NEAR PTR DST
- 功能：
  - 操作数16位：
$$SP \leftarrow (SP) - 2, [SP] \leftarrow (IP)$$
$$IP \leftarrow (IP) + 16 \text{位位移量}$$
  - 操作数32位：
$$ESP \leftarrow (ESP) - 4, [ESP] \leftarrow (EIP)$$
$$EIP \leftarrow (EIP) + 32 \text{位位移量}$$
- 说明：16位、32位位移量是一个有符号数。

# 段内间接调用

- 2) 段内间接调用
- 格式: **CALL DST** ; DST为R、M
- 功能:

操作数16位:

$SP \leftarrow (SP) - 2, [SP] \leftarrow (IP)$

$(IP) \leftarrow (EA)$

操作数32位:

$ESP \leftarrow (ESP) - 4, [ESP] \leftarrow (EIP)$

$(EIP) \leftarrow (EA)$

- 类型: 寄存器、存储器单元。



# 段间直接调用

- 3) 段间直接调用
- 格式: **CALL DST** ; DST为直接入口地址, 指令中给出
- 功能:

操作数16位:

$SP \leftarrow (SP) - 2, [SP] \leftarrow (CS)$

$SP \leftarrow (SP) - 2, [SP] \leftarrow (IP)$

$IP \leftarrow \text{DST的偏移地址}$

$CS \leftarrow \text{DST所在段的段地址}$

操作数32位:

$ESP \leftarrow (ESP) - 2, [ESP] \leftarrow (CS)$

$ESP \leftarrow (ESP) - 4, [ESP] \leftarrow (EIP)$

$EIP \leftarrow \text{DST的偏移地址}$

$CS \leftarrow \text{DST所在段的段地址}$

- 说明: 该指令先将CS、IP或EIP压栈保护返回地址。然后转移到由DST (DST为汇编语言中的过程名) 指定的转移地址。

# 段间间接调用

- 4) 段间间接调用
- 格式: **CALL DST** ; DST为M
- 功能:

操作数16位:

$SP \leftarrow (SP) - 2$  ,  $[SP] \leftarrow (CS)$

$SP \leftarrow (SP) - 2$  ,  $[SP] \leftarrow (IP)$

$IP \leftarrow (EA)$

$CS \leftarrow (EA + 2)$

操作数32位:

$ESP \leftarrow (ESP) - 2$  ,  $[ESP] \leftarrow (CS)$

$ESP \leftarrow (ESP) - 4$  ,  $[ESP] \leftarrow (EIP)$

$EIP \leftarrow (EA)$

$CS \leftarrow (EA + 4)$

- 说明: 先保护返回地址, 然后转移到由DST指定的转移地址。EA由DST确定的任何内存寻址方式。

# 段内返回

- 5) 段内返回
- 格式: **RET**
- 功能:
  - 操作数16位:  $IP \leftarrow \text{栈弹出2字节}, SP \leftarrow (SP) + 2$
  - 操作数32位:  $EIP \leftarrow \text{栈弹出4字节}, ESP \leftarrow (ESP) + 4$
- 6) 段内带参数返回
- 格式: **RET N**
- 功能:
  - 操作数16位:  $IP \leftarrow \text{栈弹出2字节}, SP \leftarrow (SP) + 2$   
 $SP \leftarrow (SP) + N$
  - 操作数32位:  $EIP \leftarrow \text{栈弹出4字节}, ESP \leftarrow (ESP) + 4$   
 $ESP \leftarrow (ESP) + N$
- 说明: N是一个16位的常数（偶数）。

# RET N 举例

； 主程序

```
MOV SP, 1009H
MOV AX, 2000H
MOV BX, 4000H
PUSH AX
PUSH BX
CALL NEAR PTR ADDPRG
MOV AX, MEM1
HLT
```

； 子程序

```
ADDPRG:
    PUSHF
    MOV BP, SP
    MOV AX, [BP+4]
    ADD AX, [BP+6]
    MOV MEM1, AX
    POPF
    RET 4
```

执行后：AX=6000H    SP=1009H

| 存储器         |          |             |
|-------------|----------|-------------|
| 1001        | FL       | } FLAG入栈    |
| 1002        | FH       |             |
| 1003        | IP低      | } IP入栈，断点地址 |
| 1004        | IP高      |             |
| 1005        | 00       | } BX        |
| 1006        | 40       |             |
| 1007        | 00       | } AX        |
| 1008        | 20       |             |
| (SP) → 1009 | //////// |             |

# 段间返回

- 7) 段间返回

- 格式: **RET**

- 功能:

操作数16位:  $IP \leftarrow \text{栈弹出2字节}, SP \leftarrow (SP) + 2$

$CS \leftarrow \text{栈弹出2字节}, SP \leftarrow (SP) + 2$

操作数32位:  $EIP \leftarrow \text{栈弹出4字节}, ESP \leftarrow (ESP) + 4$

$CS \leftarrow \text{栈弹出2字节}, ESP \leftarrow (ESP) + 2$

- 8) 段间带参数返回

- 格式: **RET N**

- 功能:

操作数16位:  $IP \leftarrow \text{栈弹出2字节}, SP \leftarrow (SP) + 2$

$CS \leftarrow \text{栈弹出2字节}, SP \leftarrow (SP) + 2$

$SP \leftarrow (SP) + N$

操作数32位:  $EIP \leftarrow \text{栈弹出4字节}, ESP \leftarrow (ESP) + 4$

$CS \leftarrow \text{栈弹出2字节}, ESP \leftarrow (ESP) + 2$

$ESP \leftarrow (ESP) + N$

# 9. 中断指令

- 9. 中断指令
- 中断指令作用：调用中断服务程序。中断向量地址=中断类型码 $\times 4$ 。
- 1) INT 中断指令
- 格式：INT n ； 不影响标志位
- 功能：
  - ①  $SP \leftarrow (SP) - 2$
  - ② PUSH (FR) ； 标志寄存器FR进栈 ①
  - ③  $SP \leftarrow (SP) - 2$
  - ④ PUSH (CS) ； 断点段地址CS进栈 ②
  - ⑤  $SP \leftarrow (SP) - 2$
  - ⑥ PUSH (IP) ； 断点地址指针IP进栈 ③
  - ⑦  $TF \leftarrow 0$  ； 禁止单步
  - ⑧  $IF \leftarrow 0$  ； 禁止中断 ④
  - ⑨  $IP \leftarrow [n \times 4]$  ； 转向中断服务程序 ⑤
  - ⑩  $CS \leftarrow [n \times 4 + 2]$

# IRET

- 2) IRET中断返回
- 格式: IRET ; 标志位随标志寄存器出栈操作而变
- 功能:
  - ①  $IP \leftarrow \text{栈弹出2字节}$  ; 断点偏移量出栈
  - ②  $SP \leftarrow (SP) + 2$
  - ③  $CS \leftarrow \text{栈弹出2字节}$  ; 断点段地址出栈
  - ④  $SP \leftarrow (SP) + 2$
  - ⑤  $FR \leftarrow \text{栈弹出2字节}$  ; 标志寄存器出栈
  - ⑥  $SP \leftarrow (SP) + 2$

# INT 21H

- **3) INT 21H 系统功能调用**
- **系统功能调用：**DOS为系统程序员及用户提供的一组中断服务程序。
- **DOS规定用中断指令INT 21H作为进入各功能调用中断服务程序的总入口，再为每个功能调用规定一个功能号，以便进入相应各个中断服务程序的入口。**
- **程序员使用系统功能调用的过程：**
  - 1) **AH ← 功能调用编号**
  - 2) **设置入口参数**
  - 3) **CPU执行 INT 21H**
  - 4) **给出出口参数**
- **DOS 提供的主要功能调用见表 4.5.1 。**



# INT 21H调用

- (1) **功能号: 1** ; 等待键盘输入, 并回送显示器  
MOV AH, 1 ; (AH) ← 功能号01H  
INT 21H ; 调用21H号软中断
- **说明:** 出口参数 (AL) = 键入字符的ASCII码。
- (2) **功能号: 2** ; 输出字符送显示器  
MOV DL, 41H ; 入口参数: (DL) ← 字符A的ASCII码  
MOV AH, 2 ; (AH) ← 功能号02H  
INT 21H ; 在屏幕上显示输出字符A
- **说明:** 无出口参数。
- (3) **功能号: 4CH** ; 终止程序, 返回  
MOV AH, 4CH ; (AH) ← 功能号4CH  
INT 21H ; 调用21H号软中断

## 4.5.7 串操作指令

- 串操作指令处理连续存放在存储器中的数据。
- 7个，其中与存储器相关的5个，与I/O相关的2个。
- 存储器相关的5个：传送MOVS、比较CMPS、扫描SCAS、装入LODS、存储STOS
- 与I/O相关的2个：输入INS、输出OUTS
- 共同特性：
  - 1) 数据类型：字节、字、双字
  - 2) 源串： DS:SI/ESI，允许段超越
  - 3) 目的串： ES:DI/EDI
  - 4) 每执行一次串操作指令，SI/ESI、DI/EDI自动修改，指向下一位置。
  - 5) 地址修改方向：由DF控制，DF=0增址（CLD），DF=1减址（STD）
  - 6) 带B、W、D为隐式，否则为显式，如MOVS BYTE PTR [DI], [SI]
  - 7) 指令中只给出一个操作数时，另一个隐含为AL、AX、EAX之一。
  - 8) 可加相应的重复前缀，赋值CX/ECX为计数值。

# MOVS

- **1. MOVS串传送** (move string)
- **格式:** MOVS DST, SRC  
    MOVSB (字节)  
    MOVSW (字)  
    MOVSD (双字) (自386起有)
- **功能:** ES:[DI/EDI] ← DS:[SI/ESI] ; 不影响标志位  
    字节传送:  $(DI/EDI) \leftarrow (DI/EDI) \pm 1$   
                   $(SI/ESI) \leftarrow (SI/ESI) \pm 1$   
    字传送:  $(DI/EDI) \leftarrow (DI/EDI) \pm 2$   
                   $(SI/ESI) \leftarrow (SI/ESI) \pm 2$   
    双字传送:  $(DI/EDI) \leftarrow (DI/EDI) \pm 4$   
                   $(SI/ESI) \leftarrow (SI/ESI) \pm 4$

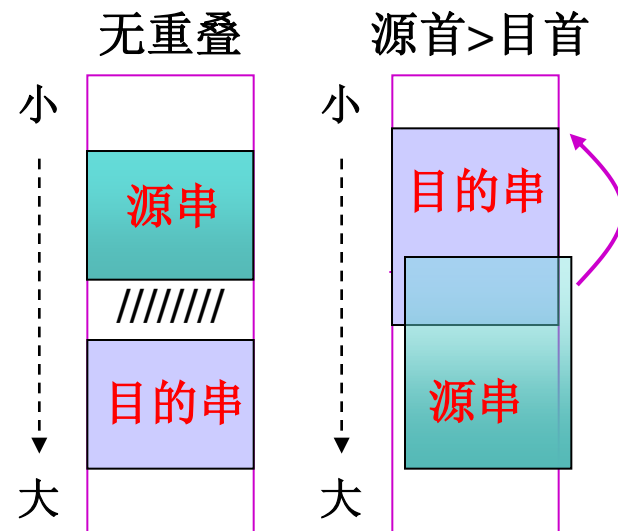
# REP

- **2. LODS串装入** (load from string )
  - 格式: **LODS SRC** ; 字节装入  $AL \leftarrow DS:[SI/ESI]$
- **3. STOS 串存储** (store into string)
  - 格式: **STOS DST** ; 字节存储  $ES:[DI/EDI] \leftarrow AL$
- **4. CMPS 串比较** (compare string)
  - 格式: **CMPS DST, SRC** ;  $DS:[SI/ESI]-ES:[DI/EDI]$ , 影响标志位
- **5. SCAS 串扫描** (scan string)
  - 格式: **SCAS DST** ; 字节扫描  $(AL) - ES:[DI/EDI]$ , 影响标志位
- **6. REP 计数重复串操作** (repeat)
  - 格式: **REP OPR** ; OPR是MOVS、STOS、LODS、INS、OUTS
  - 功能: ① 如果 $(CX)=0$ , 则退出REP, 否则往下执行。
    - ②  $(CX) \leftarrow (CX) - 1$
    - ③ 执行其后的串指令
    - ④ 重复①~③

# MOVS举例

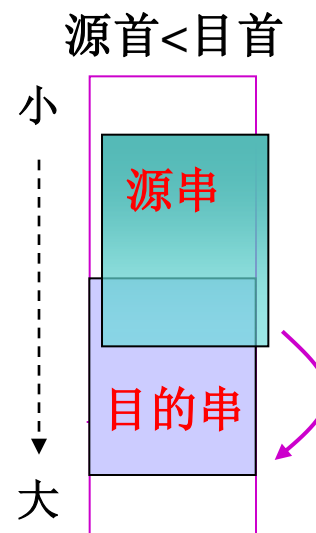
(1) MEM1→MEM2, 无重叠, 或源首>目首

```
MOV SI, OFFSET MEM1
MOV DI, OFFSET MEM2
MOV CX, 100
CLD                ; 增址
LOOP1: MOVSB        } 可改成 REP MOVSB
      LOOP LOOP1
HLT
```



(2) MEM1→MEM2, 有重叠, 源首<目首

```
MOV SI, OFFSET MEM1
MOV DI, OFFSET MEM2
MOV CX, 100
ADD SI, 99
ADD DI, 99
STD                ; 减址
REP MOVSB
HLT
```



# STOS举例

例：将MEM1开始的100个字单元清0。

```
LEA  DI, MEM1
MOV  AX, 0
MOV  CX, 100
CLD                      ; 增址
REP STOSW
HLT
```

## 4.5.8 处理器控制指令

- 标志处理指令只设置或清除本指令的标志位，而不影响其他标志位。
- **CLC** ； 进位位置0，  $CF \leftarrow 0$  (clear carry)
- **STC** ； 进位位置1，  $CF \leftarrow 1$  (set carry)
- **CLD** ； 方向标志位置0，  $DF \leftarrow 0$  (clear direction)
- **STD** ； 方向标志位置1，  $DF \leftarrow 1$  (set direction)
- **CLI** ； 中断标志置0，  $IF \leftarrow 0$ ， 禁止中断 (clear interrupt)
- **STI** ； 中断标志置1，  $IF \leftarrow 1$ ， 允许中断 (set interrupt)
- **NOP** ； 空操作， 不执行任何操作。
- **HLT** ； 暂停， 暂停指令停止软件的执行。有三种方式退出暂停： 中断、硬件复位、DMA操作。

## 4.6 伪指令

- **伪指令功能：**指示汇编程序完成规定的操作。如选择处理器、定义数据、分配存储区等。
- **1. END 源程序结束伪指令**
- **格式：**END [标号]
- **功能：**表示源程序结束，**不可缺**，源程序最后一条语句。
- **说明：**
  - 1) 标号指示程序开始执行的起始地址。
  - 2) 主程序缺省值为代码段的第一条指令的地址。
  - 3) 多个模块链接，主程序用标号，其他程序不用。



# SEGMENT

- **2. SEGMENT 和 ENDS**（段定义伪指令）
- **段定义：**确定代码组织与数据存储的方式。
- **格式：**段名 **SEGMENT** [定位类型] [组合类型] [字长类型] ['类别']
- ↓
- 段名 **ENDS**
- **功能：**定义段名、段属性，并表示段的开始位置、结束位置。
- **说明：**
  - 1) **SEGMENT**和**ENDS**必须成对出现，而且伪指令前面的段名也要相同
  - 2) 段名是段的标识符，指明段的基址，由程序员指定。
  - 3) 一般情况下，选项可以不用，使用默认值。但若需链接程序，就必须使用这些说明。

# 定位类型

- 2. SEGMENT 和 ENDS （段定义伪指令）

- 格式:

段名 SEGMENT [定位类型] [组合类型] [字长类型] ['类别']

⋮

段名 ENDS

- 定位类型: 指定段起始地址边界。

- 5种定位类型:

1) **BYTE**: 任意地址。

2) **WORD**: 偶地址, 地址最低1位为0。

3) **DWORD**: 4的倍数地址, 地址最低2位为0。

4) **PARA**: 16的倍数地址, 地址最低4位为0。默认值。

5) **PAGE**: 256的倍数地址, 地址最低8位为0。一页的起点。

# 组合类型

- 2. SEGMENT 和 ENDS （段定义伪指令）

- 格式：

段名 SEGMENT [定位类型] [组合类型] [字长类型] [‘类别’ ]

⋮

段名 ENDS

- 组合类型：表示本段与其他模块段之间，具有相同段名的各段的组合关系。
- 注意3个前提：连接时，类型相同，段名相同。

- 6种组合类型：

- 1) **PUBLIC**：连接时，把不同模块中类型相同、段名相同的段连接到同一物理存储段中，共用一个段地址。连接次序由连接命令指定。即同名段连接在一起，有共同段地址。
- 2) **STACK**：用于说明堆栈段。与PUBLIC的处理方式一样，长度为各原有段的总和。LINK自动将新段的段地址送SS，长度送SP/ESP。若未定义STACK类型，需在程序中用指令设置SS、SP。
- 3) **COMMON**：连接时，使类型相同、段名相同的段具有同一个起始地址，即产生一个覆盖段。新段的长度是最长COMMON段的长度，新段的内容取决于依次覆盖的最后内容。
- 4) **MEMORY**：表示该段定位在所有段的最下面（即地址最大的区域），多个MEMORY段产生覆盖，连接时与PUBLIC类型同等对待。
- 5) **PRIVATE**：独立段，不与同名段合并。默认值。
- 6) **AT 表达式**：指定本段起始地址为“表达式”，偏移量为0，不能用于代码段

# 字长类型

- 2. SEGMENT 和 ENDS （段定义伪指令）
- 格式：

段名 SEGMENT [定位类型] [组合类型] [字长类型] ['类别']

⋮

段名 ENDS

- 字长类型： 386以后，说明使用16位寻址还是32位寻址。
- 2种字长类型：
  - 1) **USE16**: 16位寻址，段长≤64KB，16位段地址，16位偏移量。默认值。
  - 2) **USE32**: 32位寻址，段长≤4GB，16位段地址，32位偏移量。

# 类别

- 2. SEGMENT 和 ENDS （段定义伪指令）

- 格式:

段名 SEGMENT [定位类型] [组合类型] [字长类型] ['类别']

⋮

段名 ENDS

- 类别: 引号括起来的字符串。连接时, 类别相同的段 (它们可能不同名) 放在连续的存储空间中, 但它们仍然是不同的段。

- 4种类别:

- 1) DATA: 段类别是数据段。
- 2) CODE: 段类别是代码段。
- 3) STACK: 段类别是堆栈段。
- 4) EXTRA: 段类别是附加数据段。

# ASSUME

- 3. ASSUME（段分配伪指令）

- 格式：

ASSUME 段寄存器名：段名，段寄存器名：段名， .....

- 功能：说明某个段使用哪一个段寄存器。

- 说明：

1) 程序段必须用CS，堆栈段必须用SS。

2) 该语句一般放在代码段最前面。

3) 说明性语句，除CS外（初始化赋值），各段寄存器在程序中赋值。

4) 取消语句：ASSUME NOTHING

# ORG、EVEN

- **4. ORG 地址计数器设置（起始地址定义）**
- **格式：**ORG 数值表达式
- **功能：**定义指令或数据的起始地址， $S \leftarrow$ 表达式的值。
- **说明：**数值表达式取值范围在0~65535之间。
  
- **5. EVEN 偶数地址定义（使地址计数器成为偶数）**
- **格式：**EVEN
- **功能：**使下一个变量或指令从偶地址开始。
- **说明：**便于字存储对准。EVEN在代码段中可能多出一个NOP语句。

# 数据定义伪指令

- 6. 数据定义伪指令

- 格式:

[变量名] 操作符 操作数 [; 注释]

- 功能: 为操作数分配存储单元, 用变量与存储单元相联系。

- 变量名和注释是可有可无的。

- 操作符: 定义操作数类型。

**DB:** 一个操作数占有1个字节单元（8位），定义的变量为字节变量。

**DW:** 一个操作数占有1个字单元（16位），定义的变量为字变量。

**DD:** 一个操作数占有1个双字单元（32位），定义的变量为双字变量。

**操作数:** 常数、表达式、字符串、? 等。



# 例 常数与表达式

- 例 操作数为常数与表达式

```
ORG 200H

DATA1 DB 12H, 2+6, 34H
      EVEN

DATA2 DW 789AH
      ALIGN 4

DATA3 DD 12345678H

DATA4 DW $, 6699H
```

| 汇编结果   |      |         |
|--------|------|---------|
| 变量名    | 偏移量  | 存储单元内容  |
| DATA1→ | 200H | 12H     |
|        | 201H | 08H     |
|        | 202H | 34H     |
|        | 203H | (即保留原值) |
| DATA2→ | 204H | 9AH     |
|        | 205H | 78H     |
|        | 206H | (即保留原值) |
|        | 207H | (即保留原值) |
| DATA3→ | 208H | 78H     |
|        | 209H | 56H     |
|        | 20AH | 34H     |
|        | 20BH | 12H     |
| DATA4→ | 20CH | 0CH     |
|        | 20DH | 02H     |
|        | 20EH | 99H     |
|        | 20FH | 66H     |

# 例 “？”

- 例 操作数是“？”定义形式。
- 此时只分配存储空间，  
但不定义初值。

**ORG 400H**

**DATA1 DB 1, 2, ?, 4**

**DATA2 DW 5, ?, 6**

**DATA3 DF ?**

**DATA4 DB 8**

| 汇编结果   |      |         |
|--------|------|---------|
| 变量名    | 偏移量  | 存储单元内容  |
| DATA1→ | 400H | 01H     |
|        | 401H | 02H     |
|        | 402H | (即保留原值) |
|        | 403H | 04H     |
| DATA2→ | 404H | 05H     |
|        | 405H | 00H     |
|        | 406H | (即保留原值) |
|        | 407H | (即保留原值) |
|        | 408H | 06H     |
|        | 409H | 00H     |
| DATA3→ | 40AH | (即保留原值) |
|        | 40BH | (即保留原值) |
|        | 40CH | (即保留原值) |
|        | 40DH | (即保留原值) |
|        | 40EH | (即保留原值) |
|        | 40FH | (即保留原值) |
| DATA4→ | 410H | 08H     |

# 例 DUP

- 例 操作数用复制操作符DUP定义形式
- 此时表示操作数重复若干次。

**ORG 300H**

**DATA1 DB 2 DUP (12H, 34H, 56H)**

| 汇编结果   |      |        |
|--------|------|--------|
| 变量名    | 偏移量  | 存储单元内容 |
| DATA1→ | 300H | 12H    |
|        | 301H | 34H    |
|        | 302H | 56H    |
|        | 303H | 12H    |
|        | 304H | 34H    |
|        | 305H | 56H    |

# 例 DUP嵌套

- 例 操作数用复制操作符

**DUP嵌套**定义形式。

**ORG 100H**

**DATA1 DB 12H,34H,**

**2 DUP(56H,3 DUP(9AH) ,78H)**

| 汇编结果   |      |        |
|--------|------|--------|
| 变量名    | 偏移量  | 存储单元内容 |
| DATA1→ | 100H | 12H    |
|        | 101H | 34H    |
|        | 102H | 56H    |
|        | 103H | 9AH    |
|        | 104H | 9AH    |
|        | 105H | 9AH    |
|        | 106H | 78H    |
|        | 107H | 56H    |
|        | 108H | 9AH    |
|        | 109H | 9AH    |
|        | 10AH | 9AH    |
|        | 10BH | 78H    |
|        |      |        |

# PROC、ENDP

- 7. PROC 和 ENDP （过程定义伪指令）
- 格式：

过程名 **PROC** [属性]

⋮ （过程体）

过程名 **ENDP**

- 功能：用于定义子程序结构，定义一段程序的入口（过程名）及属性。
- 说明：过程名是该过程（子程序）的入口，是CALL的操作数。
- 属性：**FAR**、**NEAR**（默认）

# EQU

- **8. EQU（赋值伪指令）**
- **格式：**符号常数名 EQU 表达式；将表达式的值赋给符号常数
- **说明：**表达式可以是有效的操作数格式，也可以是任何可求出数值常数的表达式，还可以是任何有效的符号（如操作符、寄存器名、变量名等）。
- **注意：**只能定义一次。
- DATA1 EQU 88
- AAA1 EQU CX
- DATA2 EQU DATA1+12

# 返回值运算符

- **9. 返回值运算符**
- **返回值运算符**：返回变量或标号的段地址（SEG）、返回变量或标号的偏移地址（OFFSET）、返回变量或标号的类型值（TYPE）、返回变量的单元数（LENGTH）、返回变量的字节数（SIZE）。
- 操作数 **SEG** 变量/标号           ； 段地址值赋给操作数。
- 操作数 **OFFSET** 变量/标号       ； 偏移量值赋给操作数。
- 操作数 **TYPE** 变量/标号           ； 代表变量/标号类型的值赋给操作数。
- 操作数 **LENGTH** 变量            ； 第一个数占用的单元数赋给操作数。
- 操作数 **SIZE** 变量                ； 第一个数占用的字节数赋给操作数。
- **TYPE**：变量DB返回1、DW返回2.....标号NEAR返回-1、FAR返回-2。
- **LENGTH**：只对**DUP定义**的变量有意义，返回分配给该变量的元素的个数（不是字节数），其他情况均送1。
- **SIZE**：只对**DUP定义**的变量有意义，返回分配给该变量的字节数。

$$\text{SIZE} = \text{LENGTH} \times \text{TYPE}.$$

# 例-返回值

```
DATA SEGMENT
    ORG 3000H
    AA1 DW 100 DUP(0)
    BB1 DW 1, 2
    CC1 DB 'ABCD'
    DD1 DW 1000 DUP(2, 3)
    EE1 DB 50 DUP(5, 6)
    FF1 DW 1, 2, 100 DUP(?)
    GG1 DD 5 DUP(6 DUP(?))
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END HH1
```

```
HH1: MOV AX, DATA ; 1000H
      MOV DS, AX
      MOV AX, TYPE BB1 ; 2
      MOV BX, OFFSET AA1 ; 3000H
      MOV CL, TYPE AA1 ; 2
      MOV CH, TYPE CC1 ; 1
      MOV AL, TYPE GG1 ; 4
      MOV DX, LENGTH AA1 ; 100
      MOV AX, SIZE AA1 ; 200
      MOV DX, LENGTH BB1 ; 1
      MOV AX, SIZE BB1 ; 2
      MOV DX, LENGTH CC1 ; 1
      MOV AX, SIZE CC1 ; 1
      MOV DX, LENGTH DD1 ; 1000
      MOV AX, SIZE DD1 ; 2000
      MOV DX, LENGTH EE1 ; 50
      MOV AX, SIZE EE1 ; 50
      MOV DX, LENGTH FF1 ; 1
      MOV AX, SIZE FF1 ; 2
      MOV DX, LENGTH GG1 ; 5
      MOV AX, SIZE GG1 ; 20
```



# PTR

- **10. PTR**（临时改变类型属性运算符）
- **格式：**类型 PTR 变量/标号；临时改变类型属性
- **说明：**变量：字节BYTE、字WORD、双字DWORD、三字FWORD、四字QWORD、五字TWORD；  
标号：近类型NEAR、远类型FAR。

- **例**

```
DATA1 DW      1234H, 5678H
DATA2 DB      99H, 88H, 77H, 66H
DATA3 EQU     BYTE PTR DATA1
MOV  AX, WORD PTR DATA2
MOV  BL, BYTE PTR DATA1
MOV  BL, DATA3           ; 34H→(BL)类型正确
MOV  DX, DATA1+2         ; 5678H→(DX)
MOV  [BX], 8              ; ×
MOV  BYTE PTR [BX], 8     ; 存入字节单元
MOV  WORD PTR [BX], 8     ; 存入字单元
```

## 4.7 汇编语言源程序的结构

- **80x86汇编语言源程序结构特点：**

- (1) 一个程序由若干逻辑段组成，各逻辑段由伪指令语句定义和说明。
- (2) 整个源程序以**END**伪指令结束。
- (3) 每个逻辑段由语句序列组成，语句可以是：

**指令语句：** CPU指令，可执行语句。汇编时译成目标码。

**伪指令语句：** CPU不执行，提供汇编信息，不产生目标代码。

**宏指令语句：** 一个指令序列，汇编时产生对应的目标代码序列。

**注释语句：** 以分号“；”开始，说明性语句，汇编程序不予处理。

**空行语句：** 保持程序书写清晰，仅包含回车换行符。

- 每个源程序必须有返回操作系统的指令语句，使程序执行完后能自动返回系统。

- **DOS环境下有2种程序结构：**

- (1) **.COM文件：** 长度限制为一个段长（64KB），在加载过程中没有段重定位，结构紧凑、装入速度快，适合小型程序。
- (2) **.EXE文件：** 长度仅受内存空间限制，在加载过程中需要段重定位，占用盘空间大，装入速度慢，适合中、大型程序。

# 1. .com 文件结构

- .COM文件结构特点:

(1) 整个程序（包括数据和代码）在一个段（64KB）内，不准建立堆栈段。

(2) 段的偏移量从100H开始，且在偏移量100H处是一条可执行指令。

(3) 必须由END说明起始地址。

(4) .COM文件连接，所有目标模块必须具有同一代码段名、类别名、公共属性，主模块应具有100H的入口指针并优先连接。

(5) 程序中子程序属性必须为NEAR。

# .com源程序结构-例4.7.1

- 例 4.7.1 编制在阴极射线管显示器上显示 ABCDE 的源程序。

```
CODE    SEGMENT                                ; 只指定一个段
        ORG    100H                            ; 设置起始偏移量
        ASSUME CS:CODE, DS:CODE                ; 只指定一个段
START:  MOV    AX, CS
        MOV    DS, AX                          ; 设置数据段地址
        LEA    BX, BUF
        MOV    CX, 7
A1:     MOV    DL, [BX]
        MOV    AH, 2
        INT    21H
        INC    BX
        LOOP   A1
        MOV    AH, 4CH                        ; 返回
        INT    21H
BUF     DB     0AH, 0DH, ' ABCDE'             ; 被显示的数据串
CODE    ENDS
        END    START
```

## 2. .exe 文件结构

- .exe文件的结构的特点

- (1) **多个段**，允许建立若干个不同名的代码段、数据段、附加段、堆栈段。
- (2) **空间不限**，程序长度仅受当前内存可用空间的限制。
- (3) 程序的**入口随应用而定**，只需起始标号与END语句说明的起始地址一致即可。
- (4) 子程序属性可为NEAR、FAR，随段内、段间调用而定。
- (5) **可重定位**。连接程序LINK根据被连接的目标模块的连接参数，相应生成一个“控制信息块”（或称“重定位信息块”），并将其安装在程序的前头（俗称“文件头”）。该文件头的大小依程序加载时需重定位的段的指令条数而变化，通常是512字节的整数倍。换言之，至少占1个扇区的长度。
- (6) 只有主模块的END语句指出程序入口的起始标号，并至少有一个具有STACK属性的堆栈段（**有堆栈段**）。

# .exe源程序结构-例4.7.2

|        |                                                |             |
|--------|------------------------------------------------|-------------|
| DATA1  | SEGMENT                                        | ； 数据段定义开始   |
| BUF    | DB 0AH, 0DH, 'ABCDE'                           | ； 被显示的      |
| 数据串    |                                                |             |
| DATA1  | ENDS                                           | ； 数据段定义结束   |
| DATA2  | SEGMENT                                        | ； 附加数据段定义开始 |
| ！      |                                                |             |
| DATA2  | ENDS                                           | ； 附加数据段定义结束 |
| STACK  | SEGMENT                                        | ； 堆栈段定义开始   |
|        | DW 88 DUP(0)                                   |             |
| STACK  | ENDS                                           | ； 堆栈段定义结束   |
| CODE   | SEGMENT                                        | ； 段定义开始     |
|        | ASSUME CS:CODE, DS:DATA1, ES:DATA2, SS:STACK ; |             |
| START: | MOV AX, DATA1                                  |             |
|        | MOV DS, AX                                     | ； 设置DS的值    |
|        | MOV AX, DATA2                                  |             |
|        | MOV ES, AX                                     | ； 设置ES的值    |
|        | LEA BX, BUF                                    |             |
|        | MOV CX, 7                                      |             |
| A1:    | MOV DL, [BX]                                   |             |
|        | MOV AH, 2                                      |             |
|        | INT 21H                                        |             |
|        | INC BX                                         |             |
|        | LOOP A1                                        |             |
|        | MOV AH, 4CH                                    | ； 返回        |
|        | INT 21H                                        |             |

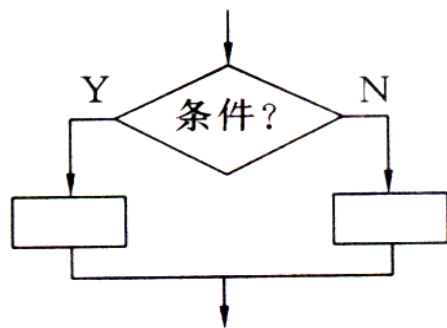
• 例4.7.2：编制在CRT上显示ABCDE的源程序

## 4.8 汇编语言程序设计

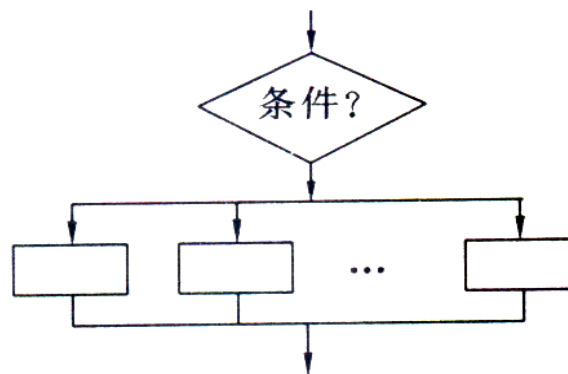
- 汇编语言程序设计的基本步骤：
  - (1) 描述问题
  - (2) 确定算法。
  - (3) 绘制流程图。
  - (4) 分配存储空间和工作单元。
  - (5) 编写程序。
  - (6) 上机调试。
- 程序基本结构形式：顺序、分支、循环、子程序。

## 4.8.1 分支程序设计

- **分支程序：**根据不同的情况或条件执行不同的功能，具有判断和转移功能，在程序中利用条件转移指令对运算结果的状态标志进行判断实现转移。
- **特点：**按条件判断、转移、分支。
- **分支程序有2种结构形式：**二路分支结构、多路分支结构。
- 两种结构的**共同特点：**在某一特定条件下，只执行多个分支中的一个分支。



二路分支结构



多路分支结构



# 1. 二路分支程序设计方法

- 例4.8.1: 有一函数，任意给定自变量X值 ( $-128 \leq X \leq 127$ )，当 $X \geq 0$ 时函数值 $Y=1$ 、当 $X < 0$ 时 $Y= -1$ ；设给定的X值存入变量X1单元，函数Y值存入变量Y1单元。

DATA SEGMENT

X1 DB X

Y1 DB ?

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

START: MOV AX, DATA

MOV DS, AX

MOV AL, X1

; 取X

CMP AL, 0

; 判0

JNS A1

;  $X \geq 0$ 转A1

MOV AL, 0FFH

; 否则 ( $X < 0$ ) ,  $AL = -1$

JMP A2

A1: MOV AL, 1

A2: MOV Y1, AL

; 存结果

MOV AH, 4CH

INT 21H

CODE ENDS

END START

{  
Y=1 , 当 $X \geq 0$   
Y= -1, 当 $X < 0$

## 2. 多路分支程序设计方法

- 三种方法：逻辑分解法、地址表法、段内转移表法。
- 例4.8.2：某工厂有5种产品的加工程序1到5，分别存放在WORK1、WORK2...WORK5为首地址的内存区域中，分别键入1、键入2...键入5选择其对应的加工程序。
- 问题解析：
  - （1）WORK1~WORK5分别对应5个加工程序。
  - （2）按1~5的输入代码转入相应程序。
- 用3种方法实现：逻辑分解法、地址表法、段内转移表法

# (1) 逻辑分解法

- 方法：逻辑判断、直接转移到目标程序。

```
CODE SEGMENT
    ASSUME CS:CODE
START: MOV     AH, 1    } 键盘输入
      INT      21H      }
      CMP      AL, 31H
      JZ        WORK1 ; 为1, 转WORK1
-----
      CMP      AL, 32H
      JZ        WORK2 ; 为2, 转WORK2
-----
      CMP      AL, 33H
      JZ        WORK3 ; 为3, 转WORK3
-----
      CMP      AL, 34H
      JZ        WORK4 ; 为4, 转WORK4
-----
      CMP      AL, 35H
      JZ        WORK5 ; 为5, 转WORK5
-----
      JMP      WORK0 ; 都不是, 退出
```

```
WORK1:  ⋮
WORK2:  ⋮
WORK3:  ⋮
WORK4:  ⋮
WORK5:  ⋮
WORK0: MOV     AH, 4CH }
      INT      21H      }
CODE  ENDS
      END      START
```

## (2) 地址表法

- **计算:**  $\text{表地址} = \text{表首址} + (\text{键号} - 1) \times 2$
- **方法:** 目标地址列表, 然后查表, 间接转移到目标程序。

```
DATA SEGMENT
TABLE DW WORK1,WORK2,WORK3,WORK4,WORK5; 地址表
DATA ENDS
```

```
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA }
      MOV DS, AX     }
      LEA BX, TABLE ; 取EA
      MOV AH, 1      ; 键盘输入
      INT 21H        }
      AND AL, 0FH    ; 屏蔽高4位
      DEC AL         ; 键号减1
      ADD AL, AL      ; 键号乘2
      SUB AH, AH      ; AH清0
      ADD BX, AX      ; 形成表地址
      JMP WORD PTR [BX]; 间接转移
```

```
WORK1:  |
WORK2:  |
WORK3:  |
WORK4:  |
WORK5:  |
      |
      MOV AH, 4CH }
      INT 21H
CODE ENDS
END START
```

### (3) 段内转移表法

- 段内短转移指令占2个字节，计算：表地址=表首址+（键号-1）× 2
- 段内近转移指令占3个字节，计算：表地址=表首址+（键号-1）× 3
- 方法：转移指令列表，通过执行转移指令最终执行目标程序。

CODE SEGMENT

ASSUME CS:CODE

START: LEA BX, WORK ; 转移表首址送BX

MOV AH, 1 ; 键盘输入

INT 21H

AND AL, 0FH

DEC AL ; 键号减1

(键号-1) { MOV AH, AL

× 3 { ADD AL, AL ; 键号乘2

{ ADD AL, AH ; 键号乘3

SUB AH, AH

ADD BX, AX ; 形成表地址

JMP BX ; 转移到列表

WORK: JMP NEAR PTR WORK1; 转移表

JMP NEAR PTR WORK2

JMP NEAR PTR WORK3

JMP NEAR PTR WORK4

JMP NEAR PTR WORK5

WORK1: ;

WORK2: ;

WORK3: ;

WORK4: ;

WORK5: ;

;

MOV AH, 4CH

INT 21H

CODE ENDS

END START

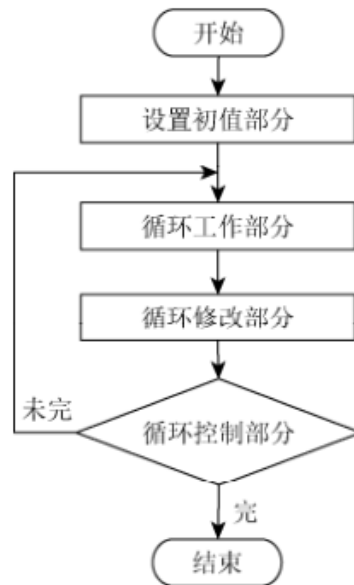
## 4.8.2 循环程序设计

- 循环程序由五部分组成：

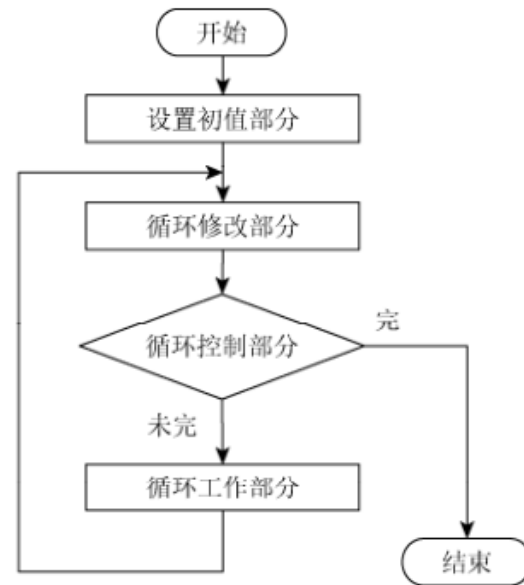
- (1) 初始化部分：设置初值。
- (2) 循环工作部分：核心。
- (3) 循环修改部分：修改循环工作变量、指针。
- (4) 循环控制部分：修改控制变量，确定程序走向。
- (5) 循环结束部分：结果处理。

- 常用结构2种：

(1) 先执行，后判断。至少执行一次循环体。



(a) 先执行后判断结构



(b) 先判断后执行结构

(2) 先判断，后执行。允许零次循环。

图 4.8.1 循环程序结构

## 例 4.8.3 循环程序的设计方法

- 循环条件控制方法3种：计数控制、条件控制（判断数据）、变量控制（判断地址，变量地址）
- 例4.8.3：某个数据段中，从偏移地址1000H单元开始，连续存放255个8位无符号整数x1、x2、...x254、x255。编写程序求这些数据的和，并将结果存入SUM1单元。
- 8位无符号数在0~255之间，255个数的和一定是一个不超过16位二进制的数。所以存放和的单元取字单元。
- 过程：AX为累加和，先清0，DL为相应的“加”数，DH清0，实现
$$AX \leftarrow AX + DX$$

# 1. 计数控制循环

- 方法：计数循环次数，按计数次数判断。
- (1) 先执行，后判断

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>DATA    SEGMENT         ORG    1000H NUMBER1 DB  x1, x2, ...x254, x255 SUM1    DW      ? DATA    ENDS CODE    SEGMENT         ASSUME CS:CODE, DS:DATA START:  MOV     AX, DATA }         MOV     DS, AX   }         LEA     BX, NUMBER1 ; 取数据EA         MOV     AX, 0      ; 累加和清0         MOV     DH, 0      ; 使用DX, 故DH先清0         MOV     CL, 255    ; 计数初值 A1:     MOV     DL, [BX]    ; 取数         ADD     AX, DX     ; 求和, AX ← AX+DX         INC     BX         ; 修改地址指针</pre> | <pre>        SUB     CL, 1      ; 计数         JNZ     A1         ; 判转         MOV     SUM1, AX   ; 存和         MOV     AH, 4CH    ;         INT     21H        ; CODE    ENDS         END     START</pre> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



# 计数控制循环

- (2) 先判断，后执行

```
DATA SEGMENT
    ORG 1000H
NUMBER1 DB x1, x2, ...x254, x255
SUM1 DW ?
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
```

```
START: MOV AX, DATA
        MOV DS, AX
```

```
    LEA BX, NUMBER1-1 ; 因为先减1判断，故首址调整
```

```
    MOV AX, 0
```

```
    MOV DH, 0
```

```
    MOV CL, 0 ; 因为先减1判断，故计数初值调整为256
```

```
A1: INC BX ; 修改地址指针
```

```
    SUB CL, 1 ; 第一次CL - 1 = 255，即0FFH
```

```
    JZ A2 ; 判转
```

```
    MOV DL, [BX] ; 取数
```

```
    ADD AX, DX ; 求和
```

```
    JMP A1
```

```
A2: MOV SUM1, AX; 存和
     MOV AH, 4CH }
     INT 21H
CODE ENDS
END START
```

## 2. 条件控制循环

- 方法：按终止数据或字符判断
- (1) 先执行，后判断

```
DATA    SEGMENT
        ORG      1000H
NUMBER1 DB      x1, x2, ...x254, x255
SUM1     DW      ?
DATA     ENDS
CODE     SEGMENT
        ASSUME CS:CODE, DS:DATA
START:   MOV      AX, DATA }
        MOV      DS, AX   }
        LEA      BX, NUMBER1 ; 取EA
        MOV      AX, 0
        MOV      DH, 0
-----
A1:      MOV      DL, [BX] ; 取数
        ADD      AX, DX ; 求和
        INC      BX ; 修改地址指针
        CMP      DL, x255 ; 终止数据
        JNZ      A1 ; 判转
-----
        MOV      SUM1, AX ; 存和
        MOV      AH, 4CH }
        INT      21H     }
CODE     ENDS
        END      START
```

# 条件控制循环

- (2) 先判断，后执行

```
DATA SEGMENT
    ORG 1000H
NUMBER1 DB x1, x2, ...x254, x255
SUM1 DW ?
DATA ENDS
CODE SEGMENT
```

```
    ASSUME CS:CODE, DS:DATA
```

```
START: MOV AX, DATA
        MOV DS, AX
        LEA BX, NUMBER1-1
```

```
        MOV AX, x255 ; 先将终止数据x255送累加和，相当于已累加
```

```
        MOV DH, 0
```

```
A1: INC BX ; 修改地址指针
```

```
        MOV DL, [BX]
```

```
        CMP DL, x255 ; 终止数据
```

```
        JZ A2 ; 判转
```

```
        ADD AX, DX ; 求和
```

```
        JMP A1
```

```
A2: MOV SUM1, AX; 存和
      MOV AH, 4CH
      INT 21H
CODE ENDS
      END START
```

# 3. 变量控制循环

- 方法：按存储器偏移地址判断。最后数x255的偏移地址应为10FEH。
- (1) 先执行，后判断

|                                                                                                                                                                                                                                                                                                |                                                                                                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <pre>DATA    SEGMENT         ORG    1000H NUMBER1 DB  x1, x2, ...x254, x255 SUM1    DW      ? DATA    ENDS CODE    SEGMENT         ASSUME CS:CODE, DS:DATA START:  MOV     AX, DATA }         MOV     DS, AX   }         LEA     BX, NUMBER1         MOV     AX, 0         MOV     DH, 0</pre> | <pre>MOV     SUM1, AX; 存和 MOV     AH, 4CH } INT     21H      } CODE    ENDS END     START</pre> |
| <hr style="border-top: 1px dashed #000;"/>                                                                                                                                                                                                                                                     |                                                                                                 |
| <pre>A1:     MOV     DL, [BX] ; 取数         ADD     AX, DX  ; 求和         INC     BX      ; 修改地址指针         CMP     BX, 10FFH ; 存放x255之后的地址, 判断结束的地址         JNZ     A1      ; 判转</pre>                                                                                                           |                                                                                                 |

# 变量控制循环

- (2) 先判断，后执行

```
DATA SEGMENT
    ORG 1000H
NUMBER1 DB x1, x2, ...x254, x255
SUM1 DW ?
DATA ENDS
CODE SEGMENT
```

```
    ASSUME CS:CODE, DS:DATA
```

```
START: MOV AX, DATA }
        MOV DS, AX }
```

```
    LEA BX, NUMBER1-1 ; 因为先加1判断，故首址调整
```

```
    MOV AX, 0
```

```
    MOV DH, 0
```

---

```
A1: INC BX ; 修改地址指针
```

```
    CMP BX, 10FFH ; 结束地址
```

```
    JZ A2
```

```
    MOV DL, [BX] ; 判转
```

```
    ADD AX, DX ; 求和
```

```
    JMP A1
```

```
A2: MOV SUM1, AX; 存和
     MOV AH, 4CH }
     INT 21H
CODE ENDS
     END START
```

## 例 4.8.4 多重循环程序设计

- 例4.8.4: 无符号数从大到小排序。从首地址AA1开始存放N个无符号字节数据，用冒泡法编制将数据按由大到小重新排序的源程序。

(1) 冒泡排序法: N个数据，相邻两个单元大小判断，位置交换，N-1次后，排出最大（或最小）数据。经过N-1次的上述循环，便可实现数据排序。

(2) 关键步骤: 比较，判断，交换。

(3) 判断指令:

| 顺序   | 有符号              | 无符号                            |
|------|------------------|--------------------------------|
| 由大到小 | 大于<br><b>JGE</b> | 高于<br><b>JAE</b><br><b>JNC</b> |
| 由小到大 | 小于<br><b>JLE</b> | 低于<br><b>JBE</b>               |

|     |          |
|-----|----------|
| 小地址 | 最大       |
|     | //////// |
|     | //////// |
| 大地址 | 最小       |

由大到小排序

|     |          |
|-----|----------|
| 小地址 | 最小       |
|     | //////// |
|     | //////// |
| 大地址 | 最大       |

由小到大排序

## 例4.8.4 - (1) N-1遍，每遍比较N-1次

```

DATA    SEGMENT
NUMBER1 DB    100, 3, 90, 80, 99, 77, 44, 66, 50  ; 9个数据
N       EQU    $- NUMBER1-1      ; 数据个数减1 (8个), $是汇编指针
DATA    ENDS

CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA

START:  MOV     AX, DATA }
        MOV     DS, AX   }

        MOV     DX, N    ; 外循环计数初值, 即遍数

A1:     LEA     BX, NUMBER1 ; 数据首址
        MOV     CX, N    ; 内循环计数初值, 即比较次数

A2:     { MOV    AL, [BX]
        { CMP    AL, [BX+1] ; 相邻两个单元比较
        { JNC    A3        ; 前大后小转 (小地址数大, 大地址数小)
        { XCHG   AL, [BX+1] ; 前小后大, 交换
        { MOV    [BX], AL

A3:     INC     BX        ; 形成下一个比较地址
        LOOP    A2        ; 内循环计数判转, 即比较次数
        DEC     DX        ; 外循环计数, 即计遍数
        JNZ     A1        ; 外循环判转

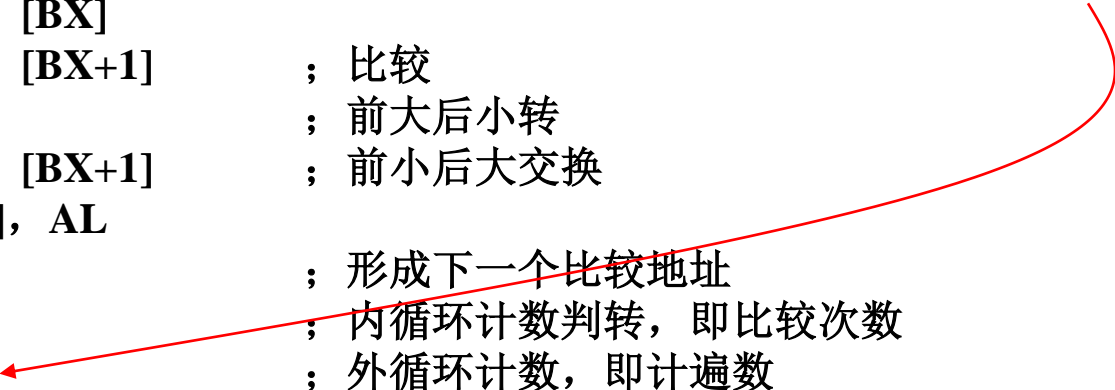
        MOV     AH, 4CH }
        INT     21H      }

CODE    ENDS
        END        START

```

## 例4.8.4 - (2) N-1遍，每遍比较次数少1

```
DATA    SEGMENT
NUMBER1 DB    100, 3, 90, 80, 99, 77, 44, 66, 50 ; 9个数据
N        EQU    $- NUMBER1-1 ; 数据个数减1 (8个), $是汇编指针
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     DX, N ; 外循环计数初值, 即遍数
A1:     LEA     BX, NUMBER1 ; 数据首址
        MOV     CX, DX ; 内循环计数初值, DX内容每循环一次减1
A2:     MOV     AL, [BX]
        CMP     AL, [BX+1] ; 比较
        JNC     A3 ; 前大后小转
        XCHG    AL, [BX+1] ; 前小后大交换
        MOV     [BX], AL
A3:     INC     BX ; 形成下一个比较地址
        LOOP    A2 ; 内循环计数判转, 即比较次数
        DEC     DX ; 外循环计数, 即计遍数
        JNZ     A1 ; 外循环判转
        MOV     AH, 4CH
        INT     21H
CODE    ENDS
        END     START
```





# 例4.8.4 - (3) 有交换标志，N-1遍，每遍比较次数少1

； 同前，数据段定义

```
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     DX, N                ; 外循环计数初值，即遍数
A1:     LEA     BX, NUMBER1           ; 数据首址
        SUB     AH, AH               ; 设置交换标志，初值0
        MOV     CX, DX               ; 内循环计数初值，即比较次数
A2:     MOV     AL, [BX]
        CMP     AL, [BX+1]           ; 比较
        JNC     A3                   ; 前大后小转
        XCHG    AL, [BX+1]           ; 前小后大交换
        MOV     [BX], AL
        MOV     AH, 1                ; 设置交换标志=1
A3:     INC     BX                    ; 形成下一个比较地址
        LOOP    A2                   ; 内循环计数判转，即比较次数
        OR      AH, AH               ; 交换标志的反码送ZF标志位
        JZ      A4                   ; 无交换转
        DEC     DX                   ; 外循环计数，即计遍数
        JNZ     A1                   ; 外循环判转
A4:     ;
        END     START
```

设置

改变

判断

## 4.8.3 子程序设计

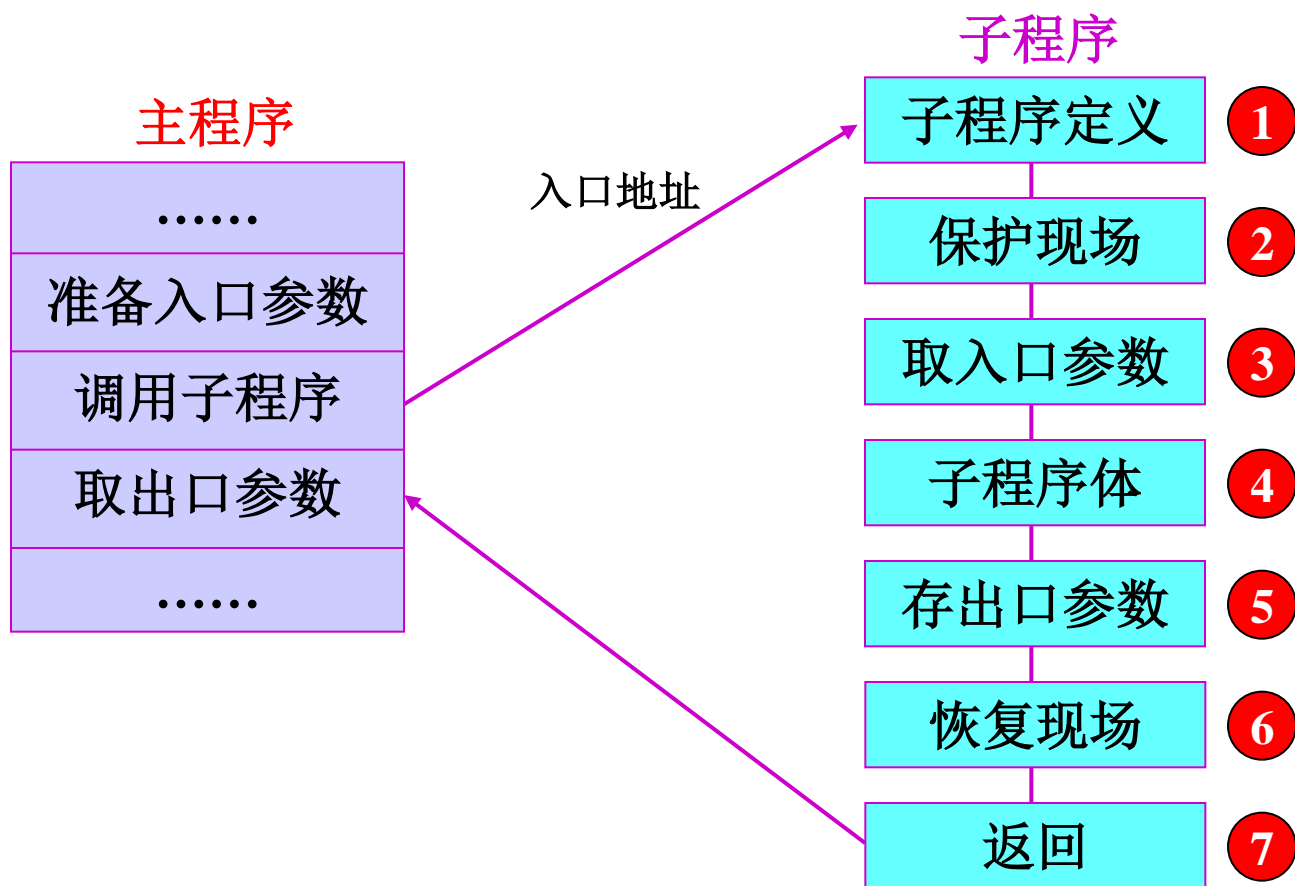
- **子程序（过程）**：可以被多次反复调用、能完成指定功能的程序段。
- **关键**：返回地址的保护与返回，涉及CALL、RET、堆栈。
- **子程序的说明文件**：
  - (1) **子程序名**：供调用子程序时使用。
  - (2) 子程序的**功能**：子程序完成的任务。
  - (3) 子程序**占用的寄存器和工作单元**：子程序执行时需要使用哪些寄存器；子程序执行后，各寄存器的变化情况。
  - (4) 子程序的**输入参数**：子程序运行所需的参数以及存放位置。
  - (5) 子程序的**输出参数**：子程序运行结束的结果参数及存放的位置。
  - (6) 子程序的运行时间：子程序运行所需的时钟周期数。
  - (7) 子程序的计算精度：子程序处理数据的位数。
  - (8) 子程序调用示例：子程序的调用格式。
  - (9) 可递归和可重入性：说明是否具有相关特性。

可递归调用：子程序能够调用其本身。

可重入性：子程序可被中断，且可被中断服务程序再次调用。

# 1. 子程序的定义

- 子程序组成：7部分，子程序定义（即子程序入口）、保护现场、取入口参数、子程序体、存输出参数、恢复现场、返回。



## 2. 现场的保存与恢复

- **保护现场：** 在子程序中，保护那些主程序要用、而子程序也要用的寄存器内容。
- **恢复现场：** 恢复现场保护的内容。
- **现场保护和现场恢复通常采用的方法：**
  - (1) 使用**堆栈指令**
  - (2) 使用**数据传送指令**

# 3. 子程序的参数传送

- 参数传送：主程序和子程序之间的数据传送。
- 参数传送的二种方法：
  - (1) 通过寄存器传送参数
  - (2) 通过堆栈传送参数或参数表地址

# 1) 通过寄存器传送参数

- **特点：**参数放在寄存器中，简便，实用，应用最多，适用于参数较少的情况。
- **例4.8.5：**用寄存器传送参数方式，编制键入**5位10进制数**加法程序。
- **设计说明：**

(1) **S1功能：**键盘输入，5位以内，10进制数→2进制数→CX，用非数字键结束本次数字输入。

10进制数→2进制数的转换算法：

$$((( (D5 \times 10 + D4) \times 10 + D3) \times 10 + D2) \times 10 + D1) \times 10 + D0$$

(2) **S2功能：**CX (2进制数) → 10进制数，在CRT上显示。

(3) 主程序**MAIN功能：**调用S1 → 求和 → 调用S2

约束条件：结果≤65535。

(4) MAIN、S1、S2同段，子程序可使用NEAR 定义。

# 例4.8.5 --1

```
CODE    SEGMENT
        ASSUME CS:CODE
MAIN    PROC    FAR                ; 主程序定义为FAR属性
        CALL    S1                ; 取被加数, 输出结果在CX中
        MOV     BX, CX            ; 保存被加数
        CALL    S1                ; 取加数
        ADD     CX, BX            ; 求和
        CALL    S2                ; 显示和
        MOV     AH, 4CH }
        INT     21H
```

# 例4.8.5 --2

S1 PROC NEAR ; 键入，十→二子程序S1，NEAR属性

PUSH AX  
PUSH BX  
PUSH DX

保护现场

XOR CX, CX ; 零送CX为每次键入值乘10作准备

A1: MOV AH, 1  
INT 21H ; 等待键入

CMP AL, 30H  
JC A2 ; 小于0键，返回  
CMP AL, 3AH  
JNC A2 ; 大于9键，返回

键入非数字键，AL<'0'，  
或AL≥'A'，返回

ADD CX, CX ; CX×2→CX  
MOV BX, CX ; CX→BX  
ADD CX, CX ; CX×2→CX  
ADD CX, CX ; CX×2→CX  
ADD CX, BX ; CX原内容（上次键入值）乘10目的

CX×10

AND AX, 0FH ; 取本次键入值  
ADD CX, AX ; 以前键入值逐乘10的值加本次值  
JMP A1 ; 转取下次键

CX×10+AX →CX类比  
(D5×10+D4)

A2: POP DX  
POP BX  
POP AX

恢复现场

RET ; 返主，输出结果在CX中

S1 ENDP



## 例4.8.5 --3

S2      **PROC NEAR**    ; 二→十，显示，NEAR属性，输入参数在CX中

**PUSH AX**            }

**PUSH BX**            }

**PUSH DX**            }

**CMP CX, 10000**

**JNC A3**                    ; 判是否有万位，避免最高位显0。

---

**CMP CX, 1000**

**JNC A5**                    ; 判是否有千位，避免最高位显0。

---

**CMP CX, 100**

**JNC A7**                    ; 判是否有百位，避免最高位显0。

---

**CMP CX, 10**

**JNC A9**                    ; 判是否有十位，避免最高位显0。

---

**JMP A11**                    ; 只有个位，转AA10

## 例4.8.5 --4

万

A3: MOV DL, -1 ; 设置初值  
A4: SUB CX, 10000  
INC DL ; 万位累计, 不影响CF  
JNC A4  
ADD CX, 10000 ; 多减一次, 则补加, 恢复  
OR DL, 30H ; 变为ASCII码  
MOV AH, 2 ; 显万位  
INT 21H }

千

A5: MOV DL, -1  
A6: SUB CX, 1000  
INC DL ; 千位累计, 不影响CF  
JNC A6  
ADD CX, 1000 ; 多减一次, 则补加, 恢复  
OR DL, 30H ; 变为ASCII码  
MOV AH, 2 ; 显千位  
INT 21H }

## 例4.8.5 --5

百

```
A7:  MOV    DL, -1
A8:  SUB     CX, 100
      INC     DL                ; 百位累计, 不影响CF
      JNC     A8
      ADD     CX, 100           ; 多减一次, 则补加, 恢复
      OR      DL, 30H           ; 变为ASCII码
      MOV     AH, 2             ; 显百位
      INT     21H
```

十

```
A9:  MOV     DL, -1
A10: SUB     CX, 10
      INC     DL                ; 十位累计, 不影响CF
      JNC     A10
      ADD     CX, 10           ; 多减一次, 则补加, 恢复
      OR      DL, 30H           ; 变为ASCII码
      MOV     AH, 2             ; 显十位
      INT     21H
```

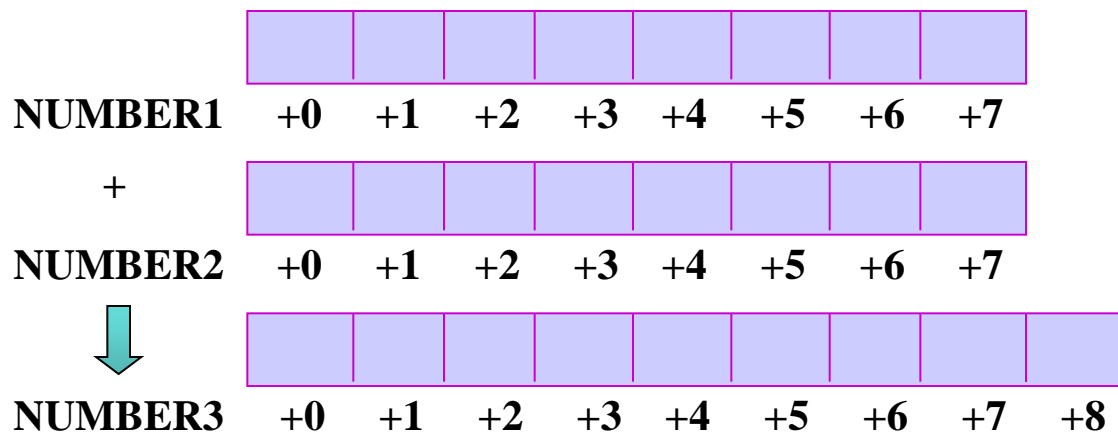
## 例4.8.5 --6

↑

```
A11:  MOV    DL, CL
      OR     DL, 30H
      MOV    AH, 2          ; 显个位
      INT    21H
      -----
      POP    DX
      POP    BX
      POP    AX
      RET                     ; 返主
S2     ENDP
MAIN   ENDP
CODE   ENDS
      END    MAIN
```

## 2) 通过堆栈传送参数或参数表地址

- **特点：**依靠堆栈操作交换参数。需要正确设置返回地址。
- **例4.8.6：**用堆栈传送参数和参数表地址方式，编制程序，键入**8位**的**非压缩BCD码**，**加法**，并**显示**。
- **设计说明：**
  - (1) **S1功能：**接收键盘输入的数字→**数字**及**数字位数**压入堆栈。键入非数字键返回。参数传递采用堆栈传送方式，约束条件：数字位数 $\leq 8$
  - (2) **S2功能：**从堆栈取出加法运算结果的地址→在CRT上显示。参数传递采用堆栈传送方式。
  - (3) 主程序**MAIN功能：**调用S1 →产生加数和被加数→加法运算（如下图），**非压缩BCD码调整**→调用S2 →清0缓冲区。



## 例4.8.6 --1

**DATA SEGMENT**

**NUMBER1 DB 8 DUP(0)** ; 被加数参数表

**NUMBER2 DB 8 DUP(0)** ; 加数参数表

**NUMBER3 DB 9 DUP(0)** ; 和参数表

**DATA ENDS**

**CODE SEGMENT**

**ASSUME CS:CODE, DS:DATA**

**MAIN PROC FAR** ; 主程序定义为FAR属性

**MOV AX, DATA**

**MOV DS, AX**

## 例4.8.6 --2

处理被加数

A1:

|      |             |                      |
|------|-------------|----------------------|
| CALL | S1          | ; 调S1非压缩BCD码接收子程序    |
| POP  | CX          | ; 弹出数据位数             |
| LEA  | BX, NUMBER1 | ; 取被加数参数表地址          |
| POP  | AX          | ; 弹出数据, 依次是个位、十位、... |
| MOV  | [BX], AL    | ; 被加数存入参数表           |
| INC  | BX          | ; 形成下位地址             |
| LOOP | A1          | ; 计数控制               |

处理加数

A2:

|      |             |                   |
|------|-------------|-------------------|
| CALL | S1          | ; 调S1非压缩BCD码接收子程序 |
| POP  | CX          | ; 取加数位数           |
| LEA  | BX, NUMBER2 | ; 取加数参数表地址        |
| POP  | AX          | ; 取加数个位、十位、百位...  |
| MOV  | [BX], AL    | ; 加数存入参数表         |
| INC  | BX          | ; 形成下位地址          |
| LOOP | A2          | ; 计数控制            |

## 例4.8.6 --3

|              |      |             |                         |
|--------------|------|-------------|-------------------------|
|              | LEA  | SI, NUMBER1 | ; 取被加数参数表地址             |
|              | LEA  | DI, NUMBER2 | ; 取加数参数表地址              |
|              | LEA  | BX, NUMBER3 | ; 取和参数表地址               |
|              | MOV  | CX, 8       | ; 8个字节加法, 8位非压缩BCD码, 计数 |
|              | OR   | CX, CX      | ; 0→CF                  |
| A3:          | MOV  | AL, [SI]    | ; 取被加数                  |
| 8字节BCD<br>加法 | ADC  | AL, [DI]    | ; 加, NUMBER1+ NUMBER2   |
|              | AAA  |             | ; 非压缩BCD码加法调整, 核心指令     |
|              | MOV  | [BX], AL    | ; 存和, 送到NUMBER3         |
|              | INC  | SI          | ; 形成下位地址                |
|              | INC  | DI          | ; 形成下位地址                |
|              | INC  | BX          | ; 形成下位地址                |
|              | LOOP | A3          | ; 计数控制                  |
|              | ADC  | CL, CL      | ; 最高位送CL                |
|              | MOV  | [BX], CL    | ; 存最高位, 送到NUMBER3+8     |



## 例4.8.6 --4

LEA AX, NUMBER3+8 ; 取和参数表最高位地址  
PUSH AX ; 向子程序提供结果最高位地址  
CALL S2 ; 非压缩BCD码显示子程序

清0缓冲区

MOV CX, 16 ; 16个字节

LEA BX, NUMBER1 ; 取被加数参数表地址

XOR AL, AL

A4: MOV [BX], AL ; 清除NUMBER1和NUMBER2中已有数据

INC BX ;

LOOP A4 ;

MOV AH, 4CH ; 返回DOS

INT 21H }

## 例4.8.6 --5

|             |      |         |   |                       |
|-------------|------|---------|---|-----------------------|
| S1          | PROC | NEAR    |   | ; 非压缩BCD码接收子程序        |
|             | POP  | BX      |   | ; 保存返回地址, 段内调用, 只保护IP |
|             | SUB  | CX, CX  |   | ; 键入数据位数计数器清0         |
| A5:         | MOV  | AH, 1   | } |                       |
|             | INT  | 21H     |   |                       |
|             | CMP  | AL, 30H |   |                       |
| 判断<br>0≤X≤9 | JC   | A6      |   | ; 判小于0键, 转A6, 返回      |
|             | CMP  | AL, 3AH |   |                       |
|             | JNC  | A6      |   | ; 判大于9键, 转A6, 返回      |
|             | INC  | CX      |   | ; 键入数据位数计数器加1         |
|             | PUSH | AX      |   | ; 键入的数据压栈             |
|             | JMP  | A5      |   |                       |
| A6:         | PUSH | CX      |   | ; 键入的数据位数压栈, 后入栈, 先弹出 |
|             | PUSH | BX      |   | ; 返回地址压栈, 保持在栈顶位置     |
|             | RET  |         |   | ; 返主                  |
| S1          | ENDP |         |   |                       |

实际上, 该子程序只是将键入的数据入栈, 并未映出是非压缩BCD码。

实际上, 该子程序只是将键入的数据入栈, 并未反映出是非压缩BCD码。

## 例4.8.6 --6

|      |      |          |   |                   |
|------|------|----------|---|-------------------|
| S2   | PROC | NEAR     | ; | 非压缩BCD码显示子程序      |
|      | POP  | AX       | ; | 返回地址IP暂时出栈        |
|      | POP  | BX       | ; | 取加法结果最高位地址        |
|      | PUSH | AX       | ; | 返回地址IP压栈, 恢复      |
|      | MOV  | CX, 9    | ; | 显示9位数据, 计数器初值     |
| A7:  | MOV  | DL, [BX] | ; | 取加法结果最高位、次高位...个位 |
|      | ADD  | DL, 30H  | ; | 形成ASCII码          |
|      | MOV  | AH, 2    | } | ;                 |
|      | INT  | 21H      |   |                   |
|      | DEC  | BX       | ; | 形成下一个显示位的地址       |
|      | LOOP | A7       | ; | 计数控制              |
|      | RET  |          | ; | 返主                |
| S2   | ENDP |          |   |                   |
| MAIN | ENDP |          |   |                   |
| CODE | ENDS |          |   |                   |
|      | END  | MAIN     |   |                   |

---

结 束