

# 汇编语言程序设计

## 复习

# 汇编语言的特点

- ◇ 汇编语言是一种以处理器指令系统为基础的低级程序设计语言，它采用助记符表达指令操作码，采用标识符号表示指令操作数
- ◇ 利用汇编语言编写程序的主要优点是可以直接、有效地控制计算机硬件，因而容易创建代码序列短小、运行快速的可执行程序
- ◇ 在有些应用领域，汇编语言的作用是不容置疑和无可替代的
- ◇ 汇编程序设计的过程是与其他高级语言程序设计大致相同

## 第3章：汇编语言程序设计

# 教学重点

---

- ◇ 汇编语言源程序格式与开发
- ◇ 常量、变量和标号
- ◇ 汇编语言程序设计方法
  - ◆ 顺序程序设计
  - ◆ 分支程序设计
  - ◆ 循环程序设计
  - ◆ 子程序设计
- ◇ 基本指令

# 3.1 汇编语言源程序格式

```
stack segment stack      ;堆栈段
    DW 40h dup(0)
```

```
stack ends
```

```
data segment             ;数据段
    ;<变量定义>
```

```
data ends
```

```
code segment             ;代码段
    assume cs:code, ds:data, ss:stack
```

```
start: mov ax,data       ;ds指向数据段
        mov ds,ax
        ;<执行代码>
```

```
code ends
```

```
end start
```

为每段分配不同的段寄存器。

Why? 段寄存器不能接受立即数。

- ◇ 完整的汇编语言源程序由段组成
- ◇ 一个汇编语言源程序可以包含若干个代码段、数据段、附加段或堆栈段，段与段之间的顺序可随意排列
- ◇ 需独立运行的程序必须包含一个代码段，并指示程序执行的起始点，一个程序只有一个起始点
- ◇ 所有的可执行性语句必须位于某一个代码段内，说明性语句可根据需要位于任一段内
- ◇ 通常，程序还需要一个堆栈段

## end汇编结束

- ◇ 汇编结束表示汇编程序到此结束将源程序翻译成目标模块代码的过程
- ◇ 源程序的最后必须有一条**END**伪指令

**END [标号]**

- ◇ 可选的“标号”参数指定程序开始执行点，连接程序据此设置**CS**和**IP**值（上页采用了**start**标识符）

# 硬指令与伪指令

- ◇ 硬指令 (**Instruction**) ——使**CPU**产生动作、并在程序执行时才处理的指令  
与具体的处理器有关、与汇编程序无关。
- ◇ 伪指令 (**Directive**) ——不产生**CPU**动作、在程序执行前由汇编程序处理的说明性指令  
伪指令与具体的处理器类型无关，但与汇编程序有关。不同版本的汇编程序支持不同的伪指令。

# 标识符与保留字

- ◆ 指令语句和伪指令语句的符号统称标识符
- ◆ 标识符一般最多由31个字母、数字及规定的特殊符号（如 \_、\$、?、@）组成，不能以数字开头。默认情况下，汇编程序不区别标识符中的字母大小写
- ◆ 一个源程序中，每个标识符的定义是唯一的，还不能是汇编语言采用的保留字

# 保留字

保留字 (**Reserved Word**) 是汇编程序已经利用的标识符 (也称为关键字)，主要有：

- ◇ 硬指令助记符——例如：**MOV、ADD**
- ◇ 伪指令助记符——例如：**DB、DW**
- ◇ 操作符——例如：**OFFSET、PTR**
- ◇ 寄存器名——例如：**AX、CS**
- ◇ 预定义符号——例如：**@data**



汇编语言大小写不敏感



# 操作数

◇ 处理器指令的操作数可以是立即数、寄存器和存储单元

## ◆ 立即数寻址方式

立即数存储在哪里？

**MOV AX, 0102H ; AX←0102H**

## ◆ 寄存器寻址方式

**MOV AX, BX ; AX←BX**

## ◆ 存储器寻址方式

- 1、直接寻址方式
- 2、寄存器间接寻址方式
- 3、寄存器相对寻址方式
- 4、基址变址寻址方式
- 5、相对基址变址寻址方式

## 1、直接寻址方式

默认的段地址在DS段寄存器

**MOV AX, [2000H] ; AX ← DS:[2000H]**

## 2、寄存器间接寻址方式

有效地址存放在基址寄存器BX  
或变址寄存器SI、DI中

**MOV AX, [BX] ; AX ← DS:[BX]**

## 3、寄存器相对寻址方式

有效地址 = BX/BP/SI/DI + 8/16位位移量

**MOV AX, [SI+06H] ; AX ← DS:[SI+06H]**

**MOV AX, 06H[SI] ; AX ← DS:[SI+06H]**

## 4、基址变址寻址方式

有效地址 = BX/BP + SI/DI

**MOV AX, [BX+SI] ; AX ← DS:[BX+SI]**

**MOV AX, [BX][SI] ; AX ← DS:[BX+SI]**

## 5、相对基址变址寻址方式

有效地址 = BX/BP + SI/DI + 8/16位位移量

**MOV AX, [BX+DI+6] ; AX ← DS:[BX+DI+6]**

**MOV AX, 6[BX+DI]**

**MOV AX, 6[BX][DI]**

# 注释

- ◇ 语句中由分号 “;” 开始的部分为注释内容，用以增加源程序的可读性
- ◇ 必要时，一个语句行也可以由分号开始作为阶段性注释
- ◇ 汇编程序在翻译源程序时将跳过该部分，不对它们做任何处理

# 汇编语言程序的开发过程（附录B）

文本编辑器

编辑

源程序：文件名.asm

错误

汇编程序

汇编

目标模块：文件名.obj

错误

连接程序

连接

可执行文件：文件名.exe

错误

调试程序

调试

应用程序

错误

## 3.2 常量和变量

- ◇ 汇编语言的数据可以简单分为常量和变量
- ◇ 常量可以作为硬指令的立即数或伪指令的参数，变量主要作为存储器操作数

## 3.2.1 常量

常量表示一个固定的数值，它又分成多种形式

1. 常数
2. 字符串
3. 符合常量
4. 数值表达式

# 1. 常数

- ◇ 指由10、16、2和8进制形式表达的数值，各种进制的数据以后缀字母区分，默认不加后缀字母的是十进制数

十进制	由0 ~ 9数字组成，以字母D (d) 结尾（缺省情况可以省略）	100, 255D
十六进制	由0 ~ 9、A ~ F数字组成，以字母H (h) 结尾，以字母开头的常数需要加一个前导0	64H, 0FFH 0B800H
二进制	由0和1两个数字组成，以字母B (b) 结尾	01100100B

## 2. 字符串

- ◇ 字符串常量是用单引号或双引号括起来的单个字符或多个字符
- ◇ 其数值是每个字符对应的**ASCII**码值
- ◇ 例如：
  - ‘d’ (等于**64H**)
  - ‘AB’ (等于**4142H**)
  - ‘Hello, Assembly !’



### 3. 符号常量

◇ 符号常量使用标识符表达一个数值

◇ 符号定义伪指令有“等价**EQU**”和“等号**=**”：

符号名 **EQU** 数值表达式

符号名 **EQU** <字符串>

符号名 **=** 数值表达式

◇ **EQU**用于数值等价时不能重复定义符号名，但“**=**”允许有重复赋值。例如：

**X = 7** ; 等效于: **X equ 7**

**X = X+5** ; “**X EQU X+5**”是错误的

## 4. 数值表达式

- ◇ 数值表达式一般是指由运算符连接的各种常量所构成的表达式
- ◇ 汇编程序在汇编过程中计算表达式，最终得到一个确定的数值，所以也是常量
- ◇ 表达式的数值在程序运行前的汇编阶段计算，所以组成表达式的各部分必须在汇编时就能确定
- ◇ 我们经常使用的是加减乘除（+ - \* /）

例如：

**mov ax,3\*4+5** ； 等价于：**mov ax,17**

### 3.2.2 变量

- ◇ 变量实质上是指主存单元的数据，虽然内存单元地址不变，但其中存放的数据可以改变
- ◇ 变量需要事先定义才能使用
- ◇ 变量定义（**Define**）伪指令为变量申请固定长度为单位的存储空间，并可以同时将相应的存储单元初始化
- ◇ 定义后的变量可以利用变量名等方法引用其中的数据，即变量的数值

# 1. 变量的定义

◇ 变量定义的汇编语言格式为：

变量名 伪指令 初值表

◇ 变量名为用户自定义标识符。变量名也可以没有

◇ 初值表是用逗号分隔的参数,主要由常量、数值表达式或“?”组成。其中“?”表示未赋初值

◇ 多个存储单元如果初值相同，可以用复制操作符**DUP**进行定义：

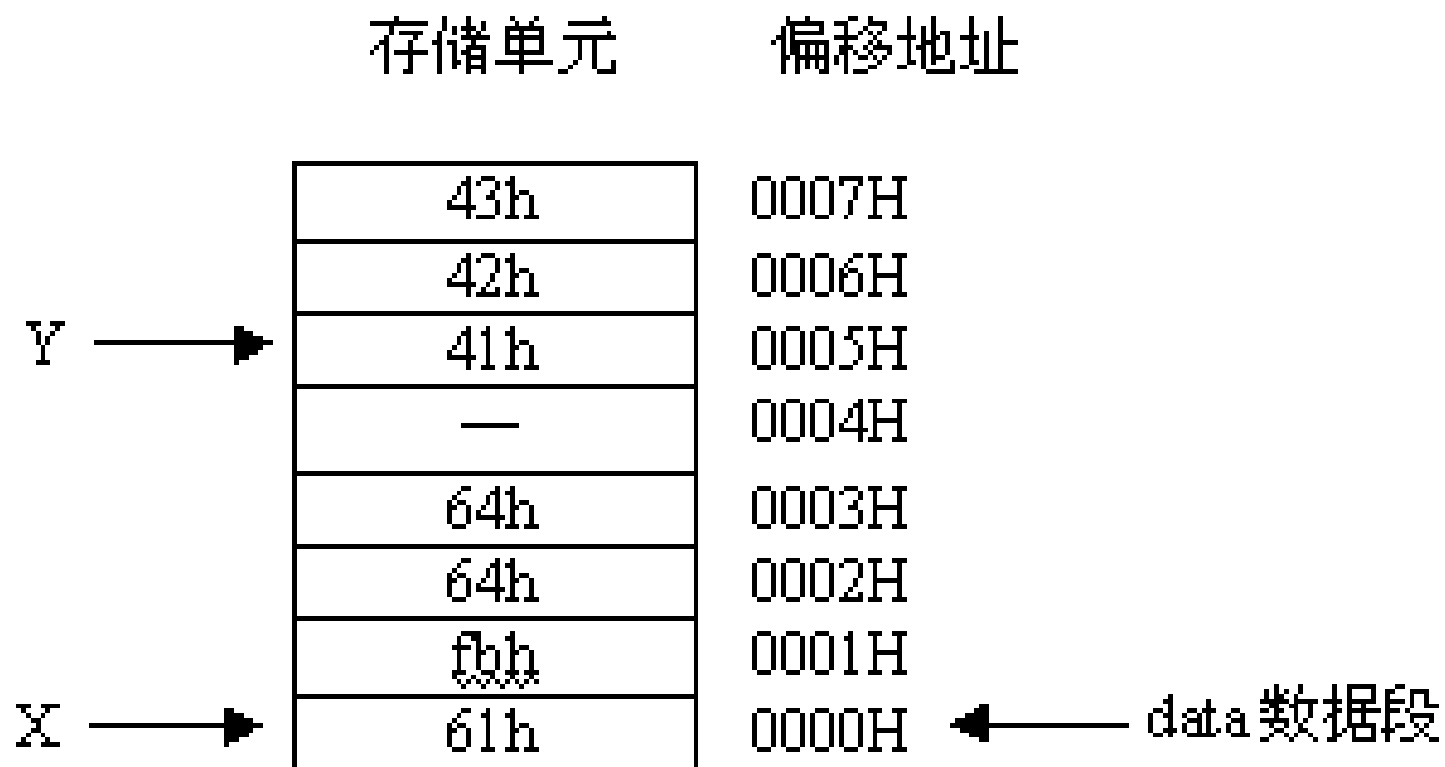
重复次数 **DUP**(重复参数)

◇ 变量定义伪指令有**DB**、**DW**、**DD**等

； 数据段

```
X      db 'a',-5  
      db 2 dup(100),?  
Y      db 'ABC'
```

## 字节变量定义实例



## 2. 变量的应用

◇ 变量具有存储单元的逻辑地址

◇ 程序代码中

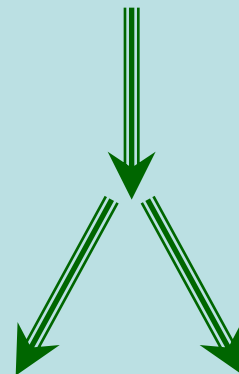
- ◆ 通过变量名引用其指向的首个数据
- ◆ 通过变量名加减位移量存取以首个数据为地址的前后数据

### 3.3 顺序程序设计

- ◇ 没有分支、循环等转移指令的程序，会按指令书写的前后顺利依次执行，这就是顺序程序
- ◇ 顺序结构是最基本的程序结构
- ◇ 完全采用顺序结构编写的程序并不多见

## 3.4 分支程序设计

- ◇ 分支程序根据条件是真或假决定执行与否
- ◇ 判断的条件是各种指令，如**CMP**、**TEST**等执行后形成的状态标志
- ◇ 转移指令**Jcc**和**JMP**可以实现分支控制
- ◇ 分支结构有
  - ◆ 单分支结构
  - ◆ 双分支结构
  - ◆ 多分支结构

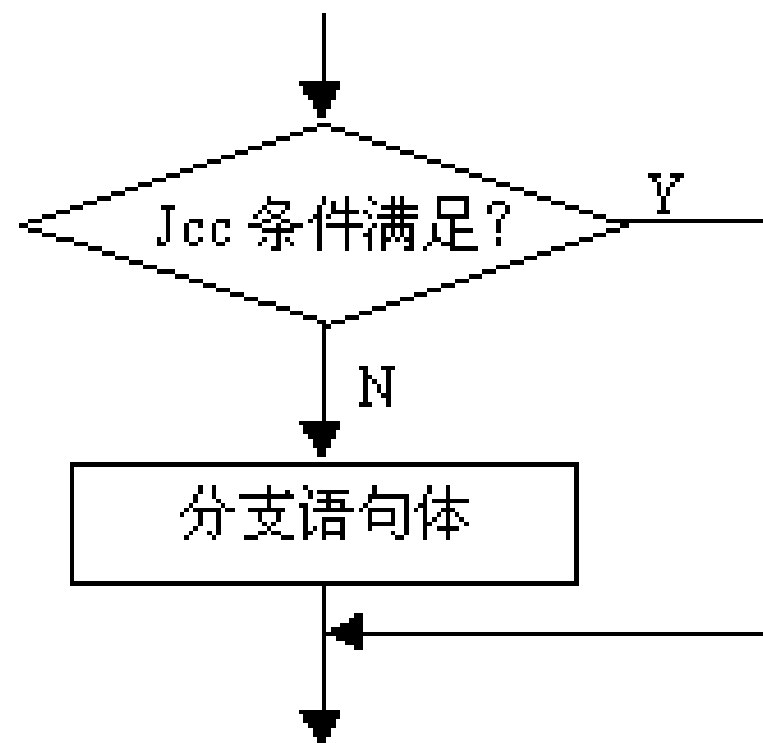




# 单分支结构

- ◇ 条件成立跳转，否则顺序执行分支语句体
- ◇ 注意选择正确的条件转移指令和转移目标地址

实例：求绝对值

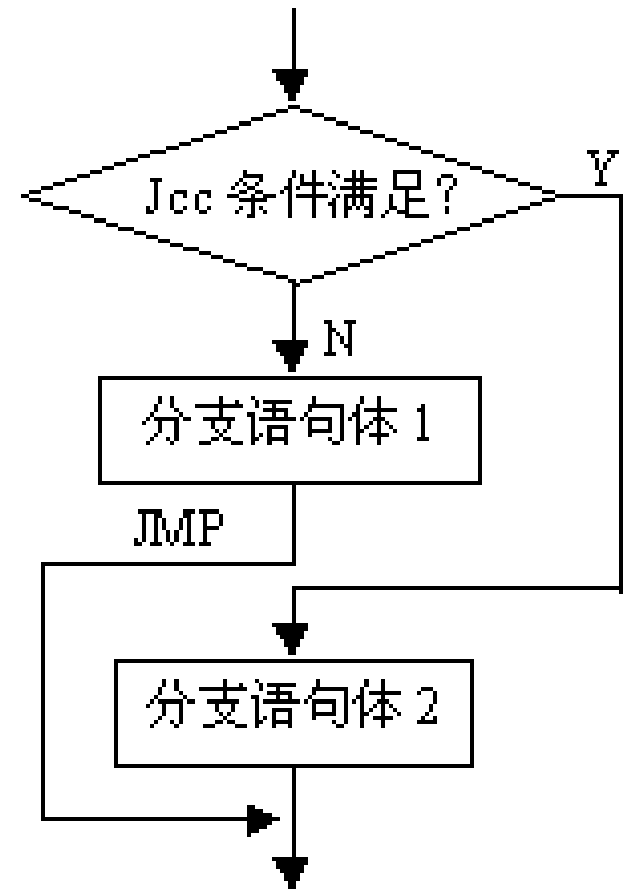


(a) 单分支结构

## 双分支结构

条件成立跳转执行第**2**个分支语句体，否则顺序执行第**1**个分支语句体

注意第**1**个分支体后一定要有一个**JMP**指令跳到第**2**个分支体后

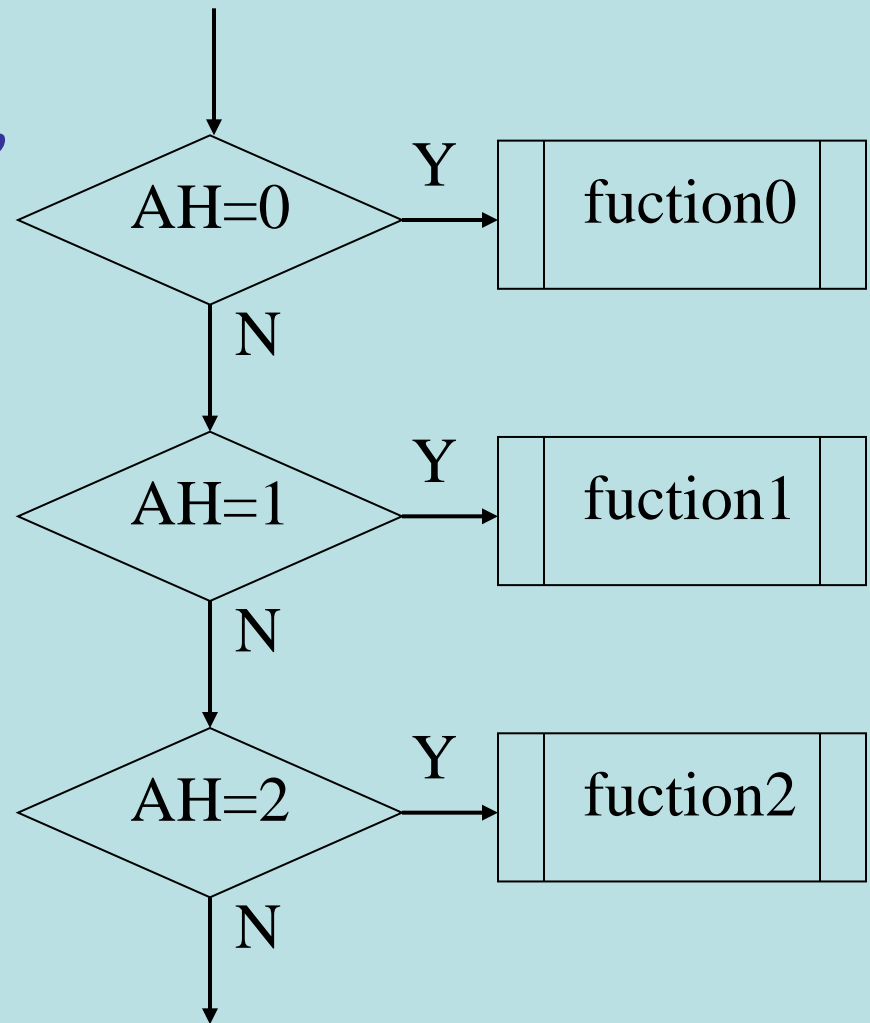


(b) 双分支结构

# 多分支结构

多分支结构是多个条件  
对应各自的分支语句体，  
哪个条件成立就转入相  
应分支体执行

```
or ah,ah    ; =cmp ah,0  
jz function0  
dec ah      ; =cmp ah,1  
jz function0  
dec ah      ; =cmp ah,2  
jz function0
```



## 3.5 循环程序设计

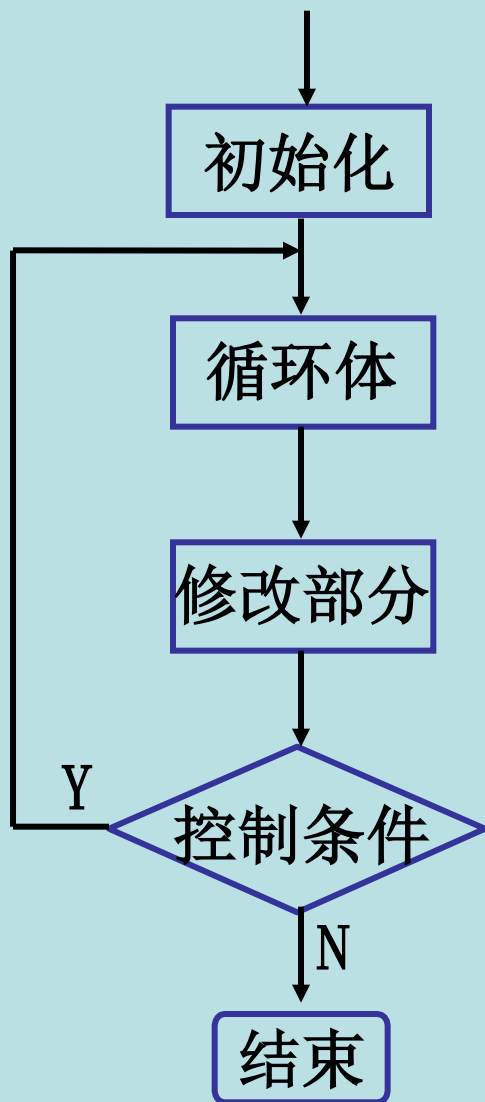
- ◇ 循环程序结构是满足一定条件的情况下，重复执行某段程序
- ◇ 循环结构的程序通常有3个部分：
  - ◆ 循环初始部分——为开始循环准备必要的条件，如循环次数、循环体需要的数值等
  - ◆ 循环体部分——指重复执行的程序部分，其中包括对循环条件等的修改程序段
  - ◆ 循环控制部分——判断循环条件是否成立，决定是否继续循环

关键是什么?

# 循环控制

- ◇ 循环结构程序的设计关键是循环控制部分
- ◇ 循环控制可以在进入循环之前进行，也可以在循环体后进行，于是形成两种结构：
  - ◆ “先判断、后循环”结构
  - ◆ “先循环、后判断”结构
- ◇ 循环结束的控制可以用循环次数，还可以用特定条件等，于是又有：
  - ◆ 计数控制循环
  - ◆ 条件控制循环

# 先循环后判断的循环结构



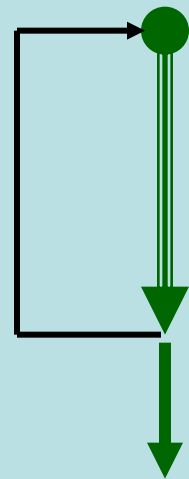
循环的初始状态

循环的工作部分  
及修改部分

{ 计数控制循环  
条件控制循环

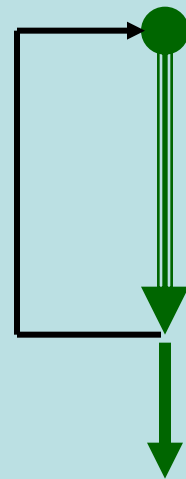
### 3.5.1 计数控制循环

- ◇ 计数控制循环利用循环次数作为控制条件
- ◇ 易于采用循环指令 **LOOP** 和 **JCXZ** 实现
  - ◆ 初始化：将循环次数或最大循环次数置入 **CX**
  - ◆ 循环体
  - ◆ 循环控制：用 **LOOP** 指令对 **CX** 减1、并判断是否为0



### 3.5.2 条件控制循环

- ◇ 条件控制循环需要利用特定条件判断循环是否结束
- ◇ 条件控制循环用条件转移指令判断循环条件
- ◇ 转移指令可以指定目的标号来改变程序的运行顺序，如果目的标号指向一个重复执行的语句体的开始或结束，便构成了循环控制结构





## 3.6 子程序设计

- ◇ 把功能相对独立的程序段单独编写和调试，作为一个相对独立的模块供程序使用，就形成子程序
- ◇ 子程序可以实现源程序的模块化，可简化源程序结构，可以提高编程效率

### 3.6.1 过程定义和子程序编写

- ◇ 汇编语言中，子程序要用一对过程伪指令**PROC**和**ENDP**声明，格式如下：

过程名    **PROC** [**NEAR**|**FAR**]  
                        .....            ; 过程体

过程名    **ENDP**

- ◇ 可选的参数指定过程的调用属性。没有指定过程属性，则采用默认属性
  - ◆ **NEAR**属性（段内近调用）的过程只能被相同代码段的其他程序调用
  - ◆ **FAR**属性（段间远调用）的过程可以被相同或不同代码段的程序调用

## 子程序编写注意事项

- (1)子程序要利用过程定义伪指令声明
- (2)子程序最后利用**RET**指令返回主程序，主程序执行**CALL**指令调用子程序
- (3)子程序中**对堆栈的压入和弹出操作要成对使用，保持堆栈的平衡**
- (4)子程序**开始应该保护使用到的寄存器内容，子程序返回前相应进行恢复**
- (5)子程序应安排在代码段的主程序之外，最好放在主程序执行终止后的位置（返回**DOS**后、汇编结束**END**伪指令前,例如**end start**），也可以放在主程序开始执行之前的位置

## 子程序编写注意事项（续）

(6)子程序允许嵌套和递归

(7)子程序可以与主程序共用一个数据段，也可以使用不同的数据段（注意修改**DS**），还可以在子程序最后设置数据区（利用**CS**寻址）

(8)子程序的编写可以很灵活，例如具有多个出口（多个**RET**指令）和入口，但一定要保证堆栈操作的正确性

(9)处理好子程序与主程序间的参数传递问题

(10)提供必要的子程序说明信息

# 参数传递

## ◇ 主程序与子程序间一个主要问题是参数传递

- ◆ 入口参数（输入参数）：主程序调用子程序时，提供给子程序的参数
- ◆ 出口参数（输出参数）：子程序执行结束返回给主程序的参数

## ◇ 参数的具体内容

- ◆ 传数值：传送数据本身
- ◆ 传地址：传送数据的主存地址

## ◇ 常用的参数传递方法

- ◆ 寄存器
- ◆ 共享变量
- ◆ 堆栈

## 3.7 基本指令

- ◇ 数据传送类指令
- ◇ 算术运算类指令
- ◇ 位操作类指令
- ◇ 控制转移类指令
- ◇ 处理器控制类指令

### 3.7.1 数据传送类指令

- ◇ 数据传送是计算机中最基本、最重要的一种操作，传送指令也是最常使用的一类指令
- ◇ 传送指令把数据从一个位置传送到另一个位置
- ◇ 除标志寄存器传送指令外，均不影响标志位
- ◇ 重点掌握

MOV XCHG XLAT PUSH POP LEA

IN OUT

# 1. 传送指令MOV (move)

◇ 把一个字节或字的操作数从源地址传送到目的地址

MOV reg/mem, imm

MOV reg/mem/seg, reg

MOV reg/seg, mem

MOV reg/mem, seg

； 段寄存器送寄存器或主存



# MOV指令——立即数传送

**mov cl,4** ;  $cl \leftarrow 4$ , 字节传送

**mov dx,0ffh** ;  $dx \leftarrow 00ffh$ , 字传送

**mov si,200h** ;  $si \leftarrow 0200h$ , 字传送

**mov bvar,0ah** ; 字节传送

; 假设bvar是一个字节变量, 定义如下: `bvar db 0`

**mov wvar,0bh** ; 字传送

; 假设wvar是一个字变量, 定义如下: `wvar dw 0`

以字母开头的常数要有前导0

 明确指令是字节操作还是字操作

# MOV指令——寄存器传送

`mov ah,al` ;  $ah \leftarrow al$ , 字节传送

`mov bvar,ch` ;  $bvar \leftarrow ch$ , 字节传送

`mov ax,bx` ;  $ax \leftarrow bx$ , 字传送

`mov ds,ax` ;  $ds \leftarrow ax$ , 字传送

`mov [bx],al` ;  $[bx] \leftarrow al$ , 字节传送



寄存器具有明确的字节和字类型

## MOV指令——存储器传送

<code>mov al,[bx]</code>	<code>; al←ds:[bx]</code>
<code>mov dx,[bp]</code>	<code>; dx←ss:[bp+0]</code>
<code>mov dx,[bp+4]</code>	<code>; dx←ss:[bp+4]</code>
<code>mov es,[si]</code>	<code>; es←ds:[si]</code>



不存在存储器向存储器的传送指令

◇注意：双操作数指令中只能有一个内存操作数出现，若要实现两个内存单元的数据传送，则需使用两条**MOV**指令，并通过通用寄存器中转才能实现。

例：将字节内存单元**VA1**中的数据传送至字节内存单元**VA2**中保存。

```
MOV AL,VA1
```

```
MOV VA2,AL
```

# MOV指令——段寄存器传送

**MOV AX,1100H**

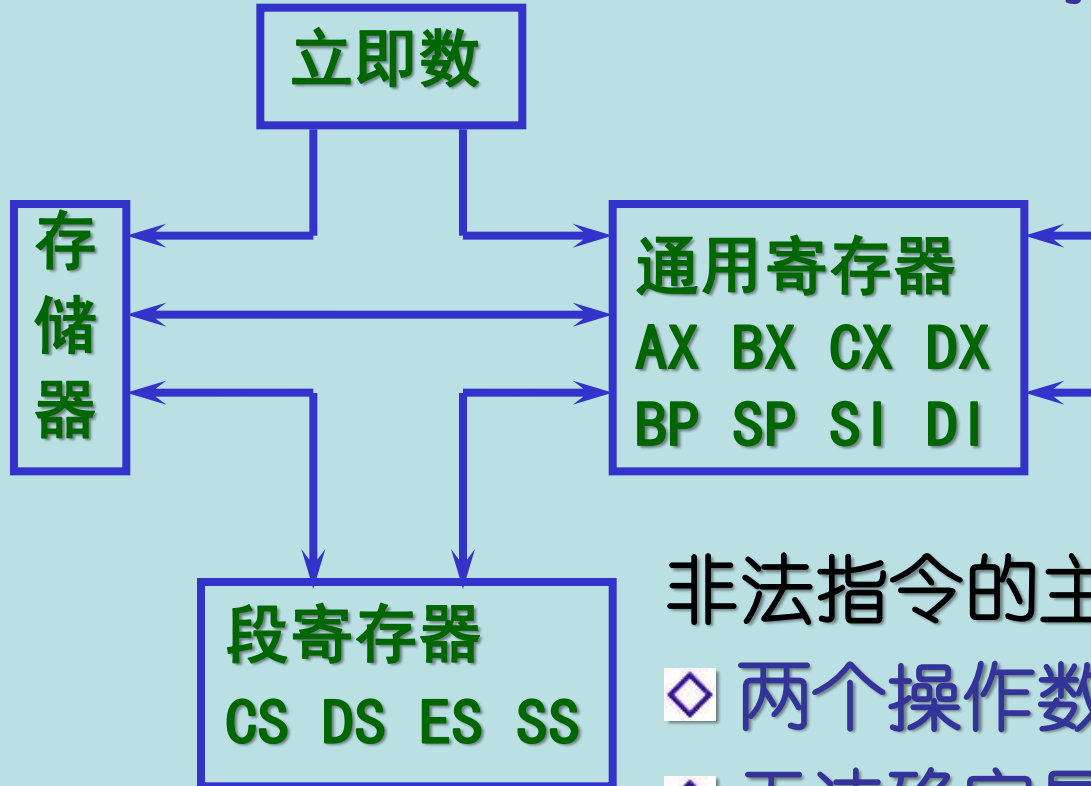
**MOV DS,AX**

**MOV ES,AX**

● 对段寄存器的操作不灵活，不能对段寄存器直接传送立即数，需通过通用寄存器中转，也不能在两个段寄存器直接传递数据。

# MOV指令传送功能图解

MOV指令也并非任意传送!



非法指令的主要现象：

- ◇ 两个操作数的类型不一致
- ◇ 无法确定是字节量还是字量操作
- ◇ 两个操作数都是存储器
- ◇ 段寄存器的操作有一些限制

# 非法指令—两个操作数类型不一致

- 在绝大多数双操作数指令中，目的操作数和源操作数必须具有一致的数据类型，或者同为字量，或者同为字节量，否则为非法指令

**MOV AL, 050AH** ; 非法指令, 修正:

**; mov ax,050ah**

**MOV SI, DL** ; 非法指令

# 非法指令—无法确定是字节量还是字量操作

- 当无法通过任一个操作数确定是操作类型时，需要利用汇编语言的操作符显式指明

**MOV [BX+SI], 255** ; 非法指令, 修正:

; **mov** byte ptr [bx+si],**255**

; byte ptr 说明是字节操作

; **mov** word ptr [bx+si],**255**

; word ptr 说明是字操作



# 非法指令—两个操作数都是存储器

- **8088**指令系统除串操作指令外，不允许两个操作数都是存储单元（存储器操作数）

**MOV buf2, buf1** ; 非法指令, 修正:

; 假设**buf2**和**buf1**是两个字变量

; **mov ax,buf1**

; **mov buf2,ax**

; 假设**buf2**和**buf1**是两个字节变量

; **mov al,buf1**

; **mov buf2,al**

# 非法指令—段寄存器的操作有一些限制

- **8088**指令系统中，能直接对段寄存器操作的指令只有**MOV**等个别传送指令，并且不灵活

**MOV DS, ES** ; 非法指令, 修正:

; **mov ax,es**

; **mov ds,ax**

**MOV DS, 100H** ; 非法指令, 修正:

; **mov ax,100h**

; **mov ds,ax**

**MOV CS, [SI]** ; 非法指令

; 指令存在, 但不能执行

## 2. 交换指令XCHG (exchange)

◇ 把两个地方的数据进行互换

XCHG reg, reg/mem

; reg  $\leftrightarrow$  reg/mem

- ◇ 寄存器与寄存器之间对换数据
- ◇ 寄存器与存储器之间对换数据
- ◇ 不能在存储器与存储器之间对换数据

### 3. 换码指令XLAT (translate)

- ◇ 将BX指定的缓冲区中、AL指定的位移处的一个字节数据取出赋给AL

XLAT                      ;  $al \leftarrow ds:[bx+al]$

- ◇ 换码指令执行前：

在主存建立一个字节量表格，内含要转换成的目的代码  
表格首地址存放于BX，AL存放相对表格首地址的位移量

- ◇ 换码指令执行后：

将AL寄存器的内容转换为目标代码

换码指令常用于将一种代码转换为另一种代码。如键盘位置码→ASCII码，数字0~9→7段显示码

例 将首地址为**400H**的表格中的**3**号数据（假设为**46H**）取出。

**mov bx,400h ;BX <- 400H**

**mov al,03h ;AL <- 03H**

**xlat ;AL <- 46H**

## 4.进栈指令PUSH

- ◇进栈指令先使堆栈指针**SP**减2，然后把一个字操作数存入堆栈顶部

PUSH r16/m16/seg

;  $SP \leftarrow SP - 2$

;  $SS:[SP] \leftarrow r16/m16/seg$

**push ax**

**push [2000h]**

## 5.出栈指令POP

- ◇ 出栈指令把栈顶的一个字传送至指定的目的操作数，然后堆栈指针**SP**加2

POP r16/m16/seg

; r16/m16/seg ← SS:[SP]

; SP ← SP + 2

**pop ax**

**pop wvar**

# 堆栈操作的特点

- 堆栈操作的单位是字，进栈和出栈只对字量
- 字量数据从栈顶压入和弹出时，都是低地址字节送低字节，高地址字节送高字节
- 堆栈操作遵循先进后出原则，但可用存储器寻址方式随机存取堆栈中的数据（**BP**寄存器采用随机存取方式读写堆栈段中的数据）
- 堆栈段是程序中不可或缺的一个内存区，常用来
  - 临时存放数据
  - 传递参数
  - 保存和恢复寄存器



## 6.标志操作指令

**CLC** ; 复位进位标志:  $CF \leftarrow 0$   
**STC** ; 置位进位标志:  $CF \leftarrow 1$   
**CMC** ; 求反进位标志:  $CF \leftarrow \sim CF$   
**CLD** ; 复位方向标志:  $DF \leftarrow 0$   
**STD** ; 置位方向标志:  $DF \leftarrow 1$   
**CLI** ; 复位中断标志:  $IF \leftarrow 0$   
**STI** ; 置位中断标志:  $IF \leftarrow 1$

## 7.有效地址传送指令LEA (load effective address)

◇ 将存储器操作数的有效地址送至指定的16位通用寄存器

**LEA r16, mem**

； r16 ← mem的有效地址EA

### 例题2.5 有效地址的获取

**mov bx,400h**

**mov si,3ch**

**lea bx,[bx+si+0f62h]**

； **BX ← 400H + 3CH + 0F62H = 139EH**

## 8. 输入输出指令

◇ 接口电路中的寄存器称为I/O端口，当CPU要与外设交换信息或对外设进行某种控制时，便借助访问端口来完成。（一个外设可以有多个I/O端口）

◇ 输入指令

**IN    ACC, PORT**

其中，ACC为AX或AL寄存器，PORT为输入端口地址

## 8. 输入输出指令

### ◇ 输出指令

**OUT PORT,ACC**

其中，**ACC**为**AX**或**AL**寄存器，**PORT**为输入端口地址

◇ **I/O端口的寻址方式---直接（指令中直接给出端口地址， $n=0-255$ ）、间接（用寄存器**DX**的内容来指定端口地址）**

注意：在**IBM-PC**系统中，实际上我们只可能使用间接**I/O**寻址方式去访问外部设备接口，因为低地址端口已经被系统使用了。

◇例：假设35H端口的每一个bit外接一个LED灯，要求点亮第二位的LED灯，其他灯不受影响。

**IN AL,35H**

**OR AL,02H**

**OUT 35H,AL**

注意：MOV指令不能访问端口，IN指令不能访问内存，访问端口的只有IN，OUT两条指令。

◇例：将**AL**中的字节数据输出到**DX**指定的I/O端口。

**IN AL,21H**

**MOV DX,300H**

**OUT DX,AL**

### 3.7.2 算术运算类指令

- ◇ 算术运算类指令用来执行二进制的算术运算：加减乘除。
- ◇ 这类指令会根据运算结果影响状态标志，有时要利用某些标志才能得到正确的结果；使用他们时请留心有关状态标志(除INC和DEC不影响CF标志外，其他按定义影响全部状态标志位)
- ◇ 重点掌握
  - 加法指令：ADD、ADC、INC
  - 减法指令：SUB、SBB、DEC、CMP、NEG

## 1. 加和减指令

### **ADD dest,src**

； 加法： $\text{dest} \leftarrow \text{dest} + \text{src}$

； **ADD**指令使目的的操作数加上源操作数，  
和的结果送到目的操作数

---

### **SUB dest,src**

； 减法： $\text{dest} \leftarrow \text{dest} - \text{src}$

； **SUB**指令使目的的操作数减去源操作数，  
差的结果送到目的操作数



## 2. 带进位加和减指令

### **ADC dest,src**

；加法： $\text{dest} \leftarrow \text{dest} + \text{src} + \text{CF}$

；**ADC**指令除完成**ADD**加法运算外，还要加上进位**CF**，结果送到目的操作数

---

### **SBB dest,src**

；减法： $\text{dest} \leftarrow \text{dest} - \text{src} - \text{CF}$

；**SBB**指令除完成**SUB**减法运算外，还要减去借位**CF**，结果送到目的操作数

### 3. 比较指令CMP (compare)

#### CMP dest,src

； 做减法运算： $\text{dest} - \text{src}$

； **CMP**指令将目的操作数减去源操作数，但差值不回送目的操作数

◇ 比较指令通过减法运算影响状态标志，用于比较两个操作数的大小关系

```
cmp ax,bx  
cmp al,100
```

## 4. 增量和减量指令

# INC reg/mem

；增量（加1）： $\text{reg/mem} \leftarrow \text{reg/mem} + 1$

# DEC reg/mem

；減量（減1）： $\text{reg/mem} \leftarrow \text{reg/mem} - 1$

- ◆ **INC指令和DEC指令是单操作数指令**
- ◆ 与加法和减法指令实现的加1和减1不同的是：**INC和DEC不影响CF标志**

```
inc si          ; si ← si + 1
dec byte ptr [si] ; [si] ← [si] - 1
```

## 5. 求补指令NEG (negative)

**NEG reg/mem**

**; reg/mem ← 0 – reg/mem**

- ◇ **NEG**指令对操作数执行求补运算，即用零减去操作数，然后结果返回操作数
- ◇ 求补运算也可以表达成：将操作数按位取反后加1
- ◇ **NEG**指令对标志的影响与用零作减法的**SUB**指令一样
- ◇ **NEG**指令也是一个单操作数指令

## 6.符号扩展指令

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

**CBW** ; **AL**符号扩展成**AX**

**CWD** ; **AX**符号扩展成**DX**

## 7. 乘法和除法指令

### ◇ 乘法指令分无符号和有符号乘法指令

**MUL reg/mem** ; 无符号乘法

**IMUL reg/mem** ; 有符号乘法

### ◇ 除法指令分无符号和有符号除法指令

**DIV reg/mem** ; 无符号除法

**IDIV reg/mem** ; 有符号除法

## 8. 十进制调整指令

- ◇ 十进制数调整指令对二进制运算的结果进行十进制调整，以得到十进制的运算结果，以此实现十进制BCD码运算
- ◇ 8088指令系统支持两种BCD码调整运算
  - **压缩BCD码**就是通常的**8421**码；它用**4**个二进制位表示一个十进制位，一个字节可以表示两个十进制位，即**00~99**
  - **非压缩BCD码**用**8**个二进制位表示一个十进制位，只用低**4**个二进制位表示一个十进制位**0~9**，高**4**位任意，通常默认为**0**

### 3.7.3位操作类指令

- ◇ 位操作类指令以二进制位为基本单位进行数据的操作
- ◇ 当需要对字节或字数据中的各个二进制位操作时，可以考虑采用位操作类指令
- ◇ 注意这些指令对标志位的影响

#### 1. 逻辑运算指令

**AND OR XOR NOT TEST**

#### 2. 移位指令

**SHL SHR SAR**

#### 3. 循环移位指令

**ROL ROR RCL RCR**



## 1. 逻辑运算指令

- ◇ 双操作数逻辑指令 **AND**、**OR**、**XOR** 和 **TEST** 设置 **CF = OF = 0**，根据结果设置 **SF**、**ZF** 和 **PF** 状态，而对 **AF** 未定义；它们的操作数组合与 **ADD**、**SUB** 等一样：

运算指令助记符 **reg, imm/reg/mem**

运算指令助记符 **mem, imm/reg**

- ◇ 单操作数逻辑指令 **NOT** 不影响标志位，操作数与 **INC**、**DEC** 和 **NEG** 一样：

**NOT reg/mem**

## 逻辑与指令AND

对两个操作数执行逻辑与运算，结果送目的操作数

AND dest, src ; dest ← dest  $\wedge$  src

只有相“与”的两位都是1，结果才是1；否则，“与”的结果为0

	01100111
AND	11011100
<hr/>	
	01010100

◇ **取位操作**：将指定的二进制数位从字节或字数据中分离

**AND AL,00000010B**

在程序设计中分离状态端口不同的状态位，再分别加以判断

◇ **清零操作**：将指定的二进制位清零。

**AND AL,11111101B**

用于将控制端口中指定的控制位清零，达到相应的控制目的。

## 逻辑或指令OR

对两个操作数执行逻辑或运算，结果送目的操作数

OR dest, src ; dest ← dest  $\vee$  src

只要相“或”的两位有一位是1，结果就是1；否则，结果为0

	01000110
OR	01011010
<hr/>	
	01011110

## ◇ 置位操作


**OR AL,00000001B**

用于将控制端口中指定的控制位置1，达到相应的控制目的。

# 逻辑异或指令XOR

对两个操作数执行逻辑异或运算，结果送目的操作数

XOR dest, src ;  $\text{dest} \leftarrow \text{dest} \oplus \text{src}$

 只有相“异或”的两位不相同，结果才是1；否则，结果为0

	01000110
XOR	10100101
<hr/>	
	11100011

## ◇ 置位操作

**XOR AL,10000000B**

用于将控制端口中指定的控制位置变反，达到相应的控制目的。

# 测试指令TEST

- ◇ 对两个操作数执行逻辑与运算，结果并不送目的操作数，仅按**AND**指令影响标志

TEST dest, src ; dest  $\wedge$  src

 **AND**与**TEST**指令的关系，  
同**SUB**与**CMP**指令的关系一样



◇ 测试某一个位是0还是1

**TEST AL, 00000010B**

**JNZ label**

如果al第二位为1，设置ZF=0，JNF将会跳转。

◇ 测试寄存器是否为空

**TEST AL,AL**

**JZ label**

如果al为零，设置ZF=1，JZ跳转。

- ◆ **TEST**指令通常可用于测试状态端口中指定状态位，并结合条件转移指令根据当前状态实现分支或循环处理。
- ◆ 也可用于对寄存器、内存单元指定数据位进行测试，根据不同的取值作不同的处理。

# 逻辑非指令NOT

对一个操作数执行逻辑非运算

NOT reg/mem ; reg/mem  $\leftarrow$   $\sim$  reg/mem

● 按位取反，原来是“0”的位变为“1”；原来是“1”的位变为“0”

NOT 01000110  
—————  
10111001

## 2. 移位指令

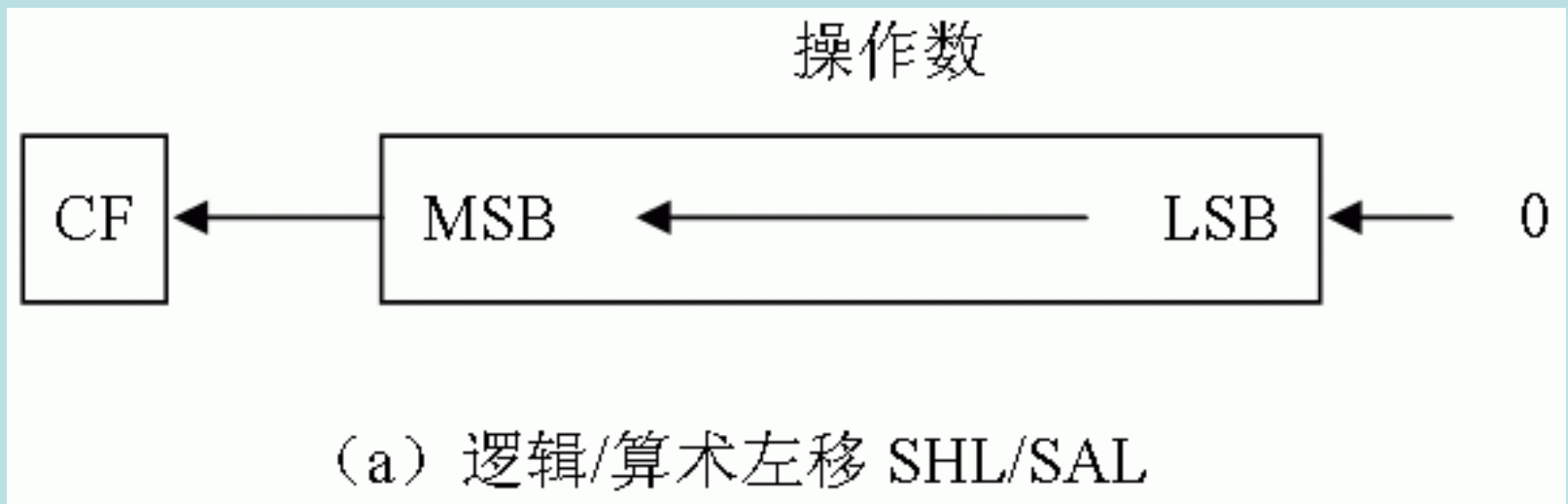
- ◇ 将操作数移动一位或多位，分成逻辑移位和算术移位，分别具有左移或右移操作
- ◇ 移位指令的**第一个操作数**是指定的被移位的操作数，可以是寄存器或存储单元；**后一个操作数**表示移位位数：
  - ◆ 该操作数为**1**，表示移动一位
  - ◆ 该操作数为**CL**，**CL**寄存器值表示移位位数（移位位数大于**1**只能**CL**表示）
- ◇ 按照移入的位设置进位标志**CF**，根据移位后的结果影响**SF**、**ZF**、**PF**

# 逻辑左移指令SHL

**SHL reg/mem,1/CL**

； reg/mem左移1或CL位

； 最低位补0， 最高位进入CF



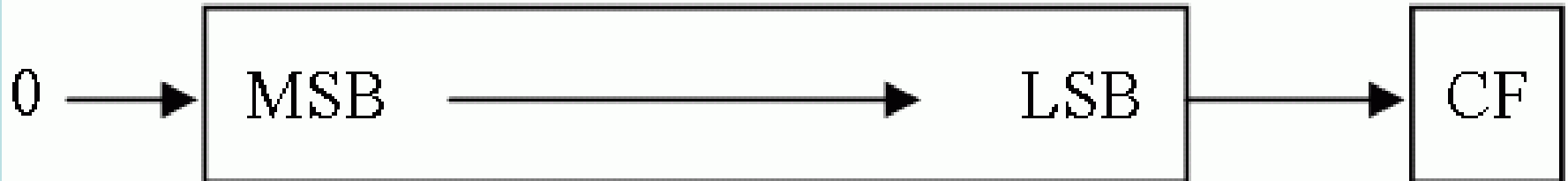
## 逻辑右移指令SHR

**SHR reg/mem,1/CL**

； reg/mem右移1/CL位

； 最高位补0，最低位进入CF

操作数



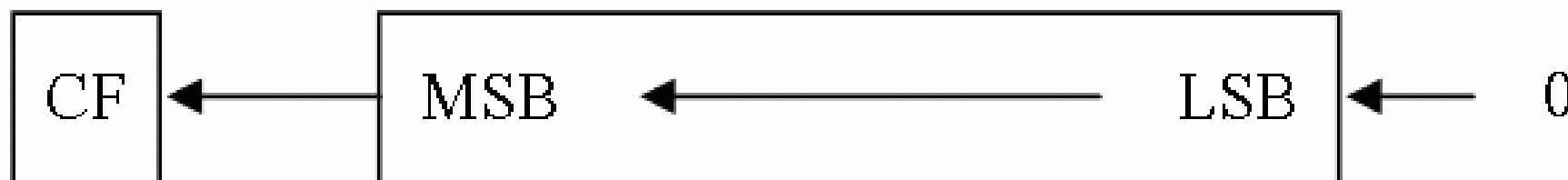
(b) 逻辑右移 SHR

# 算术左移指令SAL

**SAL reg/mem,1/CL**

；与SHL是同一条指令

操作数



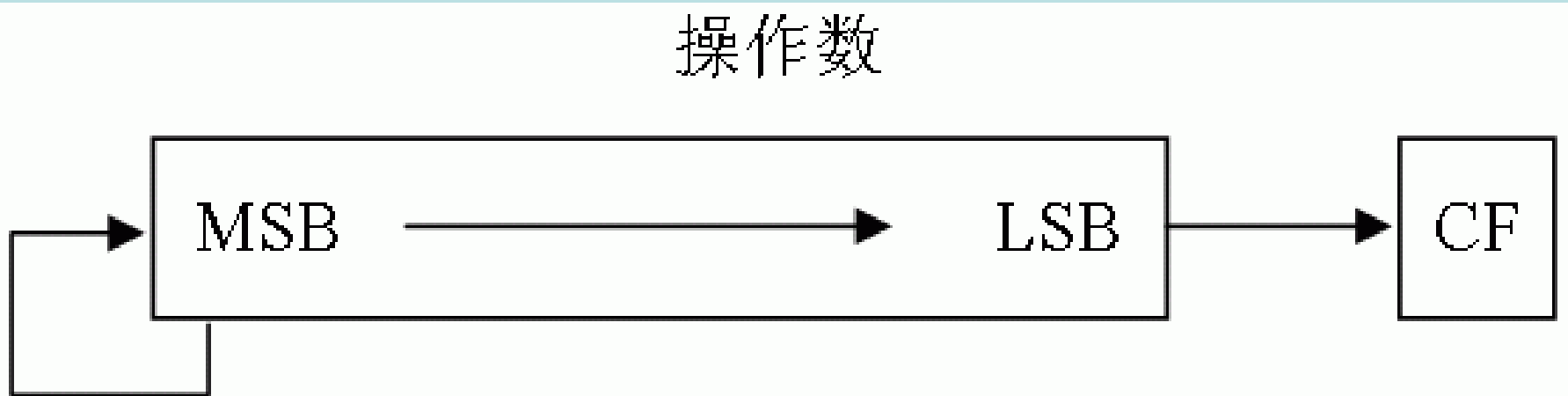
(a) 逻辑/算术左移 SHL/SAL

## 算术右移指令SAR

**SAR reg/mem,1/CL**

； reg/mem右移1/CL位

； 最高位不变，最低位进入CF



(c) 算术右移 SAR



### 3. 循环移位指令

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

**ROL reg/mem,1/CL** ;不带进位循环左移

**ROR reg/mem,1/CL** ;不带进位循环右移

**RCL reg/mem,1/CL** ;带进位循环左移

**RCR reg/mem,1/CL** ;带进位循环右移

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

### 3.7.4 控制转移类指令

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

◇ 重点掌握：

◆ **JMP/Jcc/LOOP/JCXZ CALL/RET**

◆ **INT n/IRET** 常用系统功能调用

● 控制转移类指令通过改变**IP**（和**CS**）值，实现程序执行顺序的改变

# 1. 无条件转移指令

**JMP label** ; 程序转向label标号指定的地址

- ◇ 只要执行无条件转移指令**JMP**，就使程序转到指定的目标地址，从目标地址处开始执行指令
- ◇ 操作数**label**是要转移到的目标地址（目的地址、转移地址）
- ◇ **JMP**指令分成4种类型：
  - (1) 段内转移、相对寻址
  - (2) 段内转移、间接寻址
  - (3) 段间转移、直接寻址
  - (4) 段间转移、间接寻址

# 无条件转移指令JMP (jump)

**JMP label** ;段内转移、相对寻址

**;IP←IP + 位移量**

**JMP r16/m16** ;段内转移、间接寻址

**;IP←r16/m16**

**JMP far ptr label** ;段间转移、直接寻址

**;IP←偏移地址,CS←段地址**

**JMP far ptr mem** ;段间转移, 间接寻址

**;IP←[mem],CS←[mem + 2]**

## 2. 条件转移指令

◇ 条件转移指令**Jcc**根据指定的条件确定程序是否发生转移。其通用格式为：

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

◇ **label**是一个标号、一个8位位移量，表示**Jcc**指令后的那条指令的偏移地址，到目标指令的偏移地址的地址位移

◇ **label**只支持短转移的相对寻址方式

# Jcc指令的分类

◇ Jcc指令不影响标志，但要利用标志（[表2-3](#)）

◇ 根据利用的标志位不同，分成三种情况：

- (1) 判断单个标志位状态
- (2) 比较无符号数高低
- (3) 比较有符号数大小

● Jcc指令实际虽然只有16条，但却有30个助记符

● 采用多个助记符，目的是为了更方便记忆和使用

## ◆ 判断单个标志位状态

### (1) **JZ/JE**和**JNZ/JNE**

利用零标志**ZF**，判断结果是否为零（或相等）

### (2) **JS**和**JNS**

利用符号标志**SF**，判断结果是正是负

### (3) **JO**和**JNO**

利用溢出标志**OF**，判断结果是否产生溢出

### (4) **JP/JPE**和**JNP/JPO**

利用奇偶标志**PF**，判断结果中“1”的个数是偶是奇

### (5) **JC/JB/JNAE**和**JNC/JNB/JAE**

利用进位标志**CF**，判断结果是否进位或借位

## ◆ 比较数值大小：无符号数的高低

◇ 无符号数的大小用高 (**Above**)、低 (**Below**) 表示，需要利用**CF**确定高低、利用**ZF**标志确定相等 (**Equal**)

◇ 两数的高低分成**4**种关系，对应**4**条指令

**JB (JNAE)** : 目的操作数低于 (不高于等于) 源操作数

**JNB (JAE)** : 目的操作数不低于 (高于等于) 源操作数

**JBE (JNA)** : 目的操作数低于等于 (不高于) 源操作数

**JNBE (JA)** : 目的操作数不低于等于 (高于) 源操作数



## ◆ 比较数值大小：有符号数的大小

◇ 判断有符号数的大（**Greater**）、小（**Less**），需要组合**OF**、**SF**标志、并利用**ZF**标志确定相等与否

◇ 两数的大小分成4种关系，分别对应4条指令

**JL (JNGE)**：目的操作数小于（不大于等于）源操作数

**JNL (JGE)**：目的操作数不小于（大于等于）源操作数

**JLE (JNG)**：目的操作数小于等于（不大于）源操作数

**JNLE (JG)**：目的操作数不小于等于（大于）源操作数

## 例2.22 求较大值

```
    cmp ax,bx    ; 比较AX和BX  
    jae next     ; 若 $AX \geq BX$ , 转移  
    xchg ax,bx   ; 若 $AX < BX$ , 交换  
next: mov ax,ax
```

如果**AX**和**BX**存放的是有符号数，  
则条件转移指令应采用**JGE**指令

### 3. 循环指令

◇ 一段代码序列多次重复执行就是循环

◇ 8088设计有针对**CX**计数器的计数循环指令

**LOOP label** ; 循环指令

; 首先**CX**←**CX**−1; 然后判断; 若**CX**≠0, 转移

**JCXZ label** ; 为0循环指令

: 如果**CX**=0, 则转移

◇ **label**操作数采用相对短寻址方式

◇ 还有**LOOPZ/LOOPE**和**LOOPNZ/LOOPNE**两条指令



```
dec cx
jnz label
```

## 4. 子程序指令

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

 转移指令有去无回

 子程序调用需要返回，  
其中利用堆栈保存返回地址

## ◆ 子程序调用指令CALL

### ➤ CALL指令分成4种类型（类似JMP）

**CALL label** ; 段内调用、直接寻址

**CALL r16/m16** ; 段内调用、间接寻址

**CALL far ptr label** ; 段间调用、直接寻址

**CALL far ptr mem** ; 段间调用、间接寻址

### ➤ CALL指令需要保存返回地址：

■ 段内调用——入栈偏移地址IP

$SP \leftarrow SP - 2$ ,  $SS:[SP] \leftarrow IP$

■ 段间调用——入栈偏移地址IP和段地址CS

$SP \leftarrow SP - 2$ ,  $SS:[SP] \leftarrow CS$

$SP \leftarrow SP - 2$ ,  $SS:[SP] \leftarrow IP$

## ◆ 子程序返回指令**RET**

➤ 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

**RET** ; 无参数段间返回

**RET i16** ; 有参数段间返回

➤ 需要弹出**CALL**指令压入堆栈的返回地址

■ 段内返回——出栈偏移地址**IP**

**IP**←**SS:[SP]**, **SP**←**SP+2**

■ 段间返回——出栈偏移地址**IP**和段地址**CS**

**IP**←**SS:[SP]**, **SP**←**SP+2**

**CS**←**SS:[SP]**, **SP**←**SP+2**

## 5. 中断指令和系统功能调用

- ◇ **中断 (Interrupt)** 是又一种改变程序执行顺序的方法
- ◇ **8088CPU支持256个中断**，每个中断用一个编号（中断向量号）区别
- ◇ 中断指令有3条：  
**INT i8      IRET      INTO**
- ◇ 本节主要掌握类似子程序调用指令的中断调用指令**INT i8**，进而掌握系统功能调用

# 中断指令

INT i8

- ； 中断调用指令：产生i8号中断
- ； 主程序使用，其中i8表示中断向量号

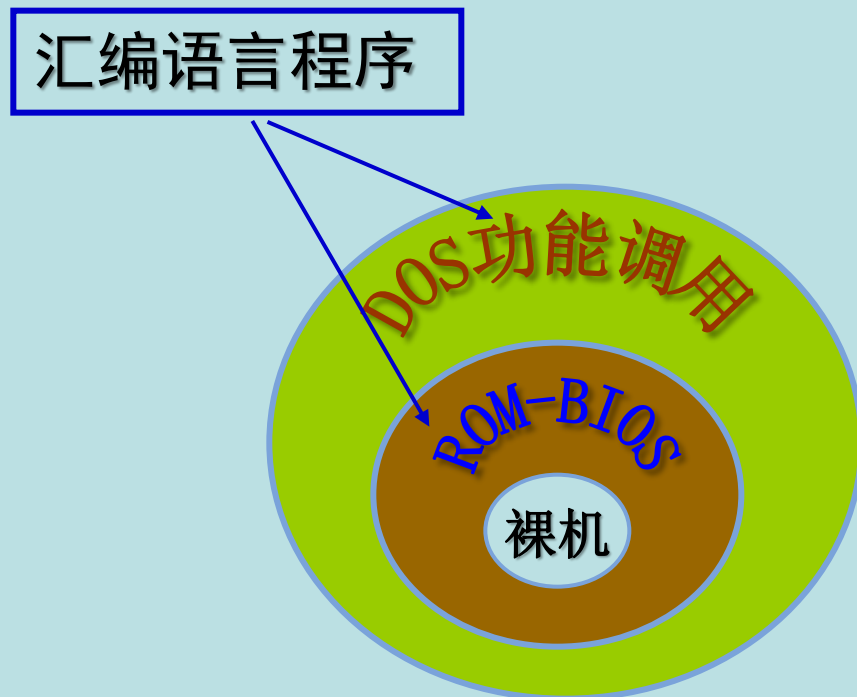
IRET

- ； 中断返回指令：实现中断返回
- ； 中断服务程序使用



# 系统功能调用方法

- ◇ 汇编程序提供给汇编语言程序员的功能非常有限
- ◇ 程序员需要利用 **ROM-BIOS** 和操作系统提供的资源
- ◇ 系统功能调用是程序设计的一个重要方面



# 系统功能调用步骤

◇ 通常按照如下4个步骤进行：

- (1) 在**AH**寄存器中设置系统功能调用号
- (2) 在指定寄存器中设置入口参数
- (3) 用中断调用指令（**INT i8**）执行功能调用
- (4) 根据出口参数分析功能调用执行情况

# DOS输入输出功能调用

◇ **DOS利用21H号中断提供给用户近百个系统功能，主要包括设备管理、目录管理和文件管理三个方面的功能**

◇ **重点掌握：**

- ◆ **输出一个字符，02H号功能调用**
- ◆ **输出一个字符串，09H号功能调用**
- ◆ **程序结束，4CH号功能调用**

## 02H号DOS功能调用

### ◇ 输出一个字符

- (1) 功能调用号: **AH = 02H**
- (2) 入口参数: **DL = 欲显示字符的ASCII码**
- (3) 功能调用: **INT 21H**
- (4) 出口参数: 无

； 显示一个问号 “ ? ”

**mov ah,02h**      ； 设置功能调用号

**mov dl,'?'**      ； 设置入口参数

**int 21h**          ； 功能调用

## 09H号DOS功能调用

### ◇ 输出一个字符串

(1) 功能调用号：**AH = 09H**

(2) 入口参数：

**DS:DX** = 欲显示字符串在内存中的首地址（逻辑地址形式：**DS** = 段地址，**DX** = 偏移地址）

内存中的字符串以**ASCII**码形式保存，最后必须添加一个“\$”结尾（并不显示）

(3) 功能调用：**INT 21H**

(4) 出口参数：无

## 例 提示按任意键继续

； 在数据段定义要显示的字符串

**msgkey db 'Press any key to contiune...','\$'**

； 在代码段编写程序

**mov ah,9**

**mov dx,offset msgkey**

； 设置入口参数：

； **DS** = 数据段地址（假设已经设置好）

； **DX** = 字符串的偏移地址

**int 21h**

# 4CH号DOS功能调用

## ◇ 程序终止

应用程序执行结束，将控制权交还操作系统。

(1) 功能调用号：**AH=4CH**

(2) 入口参数：无

(3) 功能调用：**INT 21H**

(4) 出口参数：**AL=返回码**（通常用**0**表示程序没有错误）

； 程序终止

**mov ax,4c00h** ； 设置功能调用号和返回码

**int 21h** ； 功能调用

# ROM-BIOS输入输出功能调用

- ◇ **ROM-BIOS**也以中断服务程序的形式，向程序员提供系统的基本输入输出程序
- ◇ **ROM-BIOS**功能更加基本，且与操作系统无关
- ◇ 当**DOS**没有启动或不允许使用**DOS**功能调用时，可以使用**ROM-BIOS**功能调用
- ◇ 熟悉输出一个字符：
  - (1) 功能调用号：**AH=0EH**
  - (2) 入口参数：**AL=欲显示字符的ASCII码**  
通常**BX=0**
  - (3) 功能调用：**INT 10H**
  - (4) 出口参数：无



### 3.7.5 处理器控制类指令

◇ 处理器控制类指令用来控制**CPU**的状态，使**CPU**暂停、等待或空操作等

**NOP** ； 空操作指令，等同于 “**xchg ax,ax**”指令

**SEG:** ； 段超越前缀指令：**CS:**， **SS:**， **DS:**， **ES:**

**HLT** ； 暂停指令：**CPU**进入暂停状态

◇ 还有其他指令：

**LOCK**   **ESC**   **WAIT**