# 第1章 汇编语言基础知识

**http://bbs.pediy.com  看雪学院(国内讨论破解)**

**[http://forum.exetools.com](http://forum.exetools.com/) 讨论汇编、破解的网站**

**http://www.tuts4you.com 国外讨论破解**

**http://www.woodmann.com/crackz 国外老的破解教程**

**http://www.movsd.com 国外32位汇编网站**

## 1.1 什么是汇编语言

### 1.1.1 计算机语言的地位和作用

### 1.1.2计算机语言的分类

**假如要编程计算1+1=?，则可用BASIC、C、ASM、机器语言分别写出如下程序：**

**①  BASIC程序**

**x = 1**

**x = x + 1**

**②  C语言程序**

**int  x;**

**x = 1;**

**x ++;**

**③  汇编语言(Assembly Language)程序**

**mov  ax, 1**

**add  ax, 1**

**④  机器语言程序**

** 1011 1000**

IMG_256**0000 0001**

**0000 0000**

** 0000 0101**

**0000 0001**

**0000 0000**

**汇编语言是对机器语言的符号化**

**Intel CPU电脑中，所以长度超过1个字节的值**

**都按照低字节在前，高字节在后的顺序存放到内存中。**

**例如有一个16位数=1011 0110 1111 0000**

**高8位      低8位**

**现在要把它放到内存地址1000开始的单元中：**

**地址    值**

**1000  1111 0000**

**1001  1011 0110**

**Little-Endian表示法**

### 1.1.3 汇编语言   (港台译作组合语言)

英文名称：**Assembly Language    assemble组合**

汇编语言是最接近机器语言的低级语言。

## 1.2 为什么要学习汇编语言

### 1.2.1 汇编语言的特点

汇编语言主要有三大特点：**控制强**、**代码短**、**速度快**。

**32位汇编的编程步骤:**

**主页下载masm32，解包并安装，假定安装在c:\masm32;**

**主页下载sum.rar并解包，里面包含一个汇编源程序sum.asm，把此源程序复制到c:\masm32;**

**双击c:\masm32\qeditor.exe打开32位汇编的集成环境;**

**编译步骤:**

**file->open->sum.asm**

**project -> build all**

**运行: 双击编译出来的exe文件或project->run program**

**若要对sum.asm进行修改，则可以在qeditor集成环境下进行，也可以用第三方程序编辑器如editplus(主页下载)进行。**

**16位汇编编程步骤：**

**主页下载dosbox86.rar，解包，设解压到c:\，则会自动生成c:\dosbox86文件夹；**

**用editplus编辑并保存源程序h.asm，把此源程序复制到dosbox86\masm文件夹内;**

**双击dosbox86\dosbox86.exe**

**file->open->h.asm**

**assemble->assemble 汇编 把h.asm编译成h.obj**

**assemble->link 连接 把h.obj连接成h.exe**

**run->run 运行**

**debug->user screen 查看结果**

**debug->debug 跟踪，F8走一步**

**附: h.asm源程序**

**data segment**

**hello db "Hello,world!", 0Dh, 0Ah, '$'**

**data ends**

**code segment**

**assume cs:code, ds:data**

**main:**

**mov ax, data**

**mov ds, ax**

**mov ah, 9**

**mov dx, offset hello**

**int 21h**

**mov ah, 4Ch**

**int 21h**

**code ends**

**end main**

**16位汇编源程序的手工编译步骤:**

**先选择菜单File->exit退出集成环境**

**接下去输入编译命令:**

**masm h;      编译  macro assembler宏汇编器**

**link h;      连接**

**h             运行**

**td h         调试**

**如果需要在调试时看到源代码，则按以下步骤编译:**

**tasm /zi /zd h       tasm: turbo assembler**

**tlink /v h            Borland公司开发**

**h**

**td h                   td: turbo debugger**

**在退出集成环境后，输入命令**

**intview**

**可以查中断大全(例如输入2109表示查int 21h的09h功能，2102表示int 21h的2号功能)**

**若要在dosbox命令窗口中重新回到集成环境，可以输入以下**

**命令：**

**ae**

### 1.2.2 汇编语言的应用

**操作系统DOS、设备驱动程序、查错程序(debug、SoftICE)、**

**混合语言编程、嵌入式系统**

**病毒与反病毒**

**加密与解密**

数据恢复

逆向工程：

**OllyICE   主页可以下载  调试32位Windows程序**

**跟踪修改password.exe的过程：**

**F8跟踪一步**

**按F2设一个断点(break point)**

**F7跟踪进入call调用内部**

**call之前通常用push来传递函数的参数**

**Ctrl+G 输入地址可以在代码窗/数据窗中定位到该地址**

**strcmp(a,b) 若a==b,则函数值=0，否则函数值≠0**

**无条件跳转指令jmp (jump)**

**可以把jnz 401043 改成**

**jmp 401034**

**修改EXE文件的工具QuickView 主页下载**

**开始->运行->command进DOS方式**

**假定QV安装在D:\QV, password.exe也拷贝在**

**此文件夹内，输入以下命令进入QV文件夹**

**D:**

**CD \QV**

**QV password.exe  用工具打开该EXE文件**

**先按2次回车切换到汇编指令方式下，再按F2切换到**

**32位汇编模式。按F7搜索特征机器码75 0F 68 54**

**光标定位到1032地址，再按Tab键跳到汇编代码处,**

**输入jmp 1034。按Alt+F9保存。按Esc退出。**

## 1.3 怎样学好汇编语言

"**The only way to learn a new programming language is by writing programs in it." --- Brian W. Kernighan & Dennis M. Ritchie**

**"The best way to become a better programmer is to write programs." --- James Sinnamon**

## 1.4 第一个汇编语言程序

**data segment**

**hi db "Hello,world!", 0Dh, 0Ah, "$"**

**data ends**

**code segment**

**assume cs:code, ds:data**

**main:**

**mov ax, data**

**mov ds, ax**

**mov ah, 9**

**mov dx, offset hi**

**int 21h**

**mov ah, 4Ch**

**int 21h**

**code ends**

**end main**

# 第2章                      数据的表示方式和运算

## 2.1  数制

## 2.1.1  二进制数(The Binary Numbering System)

**每个位共有0-1两个数字的变化，运算时“逢二进一”，“借一当二”。二进制数在汇编语言中表示：加后缀B或b，例如：**

**10110110B    10110000b**

**mov ax, 1010B  相当于**

**mov ax, 0Ah  相当于**

**mov ax, 10**

### 2.1.2  十六进制数(The Hexadecimal Numbering System)

十六进制数在汇编语言中表示：**加后缀H或者h**，例如：

**2564h    1B7Ch    4Ch**

**如果16进制数以字母开头，则前面还需要加0**，例如：

**0BC7Dh   0Dh      0Ah**

## 2.2  二进制数据的组织

小端表示法 little-endian

Short int x=0x1234；

设&x=1000;

+1000(有加号表示地址） 0x34；低字节在先

+1001 0x12；高字节在后

Long int x=0x12345678；

设&x=1000;

+1000(有加号表示地址） 0x78；低字节在先

+1001 0x56；高字节在后

+1002 0x34；

+1003 0x12；

基本概念：**bit, byte, word, double word**

①  **bit：位**

②  **byte：字节**

**1 字节 = 8 位**

**字节是计算机存储和处理信息的基本单位**。

**byte 相当于C语言中的**

**unsigned char 范围[0,255]即[00h,0FFh]**

③  **word：字=2字节=16位   相当于C语言中的**

**unsigned short int 范围[0,65535]**

**即[0000h, 0FFFFh]**

③  **double word=2字=4字节=32位：双字**

**相当于C语言中的unsigned long int**

**Db 12h； unsigned char a=0x12；**

**Dw 1234h； unsigned short int b=0x1234；**

**Dd 12345678h ；unsigned long int c=0x12345678；**

## 2.3  符号数（Signed Numbers）的表示

①  **正数的表示：最高位=0**

**8位正数**

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| **0**0000000 | **0**1111111 | 二进制 |
| 00h | 7Fh | 十六进制 |
| 0 | +127 | 十进制 |

**16位正数**

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| **0**0000000000000000 | **0**111111111111111 | 二进制 |
| 0000h | 7FFFh | 十六进制 |
| 0 | **+32767** | 十进制 |

**32位正数**

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| 00000000 | 7FFFFFFFh | 十六进制 |
| 0 | 2147483647 | 十进制 |

②  **负数的表示**：**二进制补码(two’s complement)**

**8位负数: -128 ~ -1**

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| **1**0000000 | **1**1111111 | 二进制 |
| 80h | 0FFh | 十六进制 |
| -128 | -1 | 十进制 |

**16位负数**

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| **1**0000000000000000 | **1**111111111111111 | 二进制 |
| 8000h | 0FFFFh | 十六进制 |
| -32768 | -1 | 十进制 |

**32位负数**

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| 80000000h | 0FFFFFFFFh | 十六进制 |
| -2147483648 | -1 | 十进制 |

③  符号数的表示

**8位符号数**(在C语言中对应的数据类型为**char**)

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| **1**0000000 | **0**1111111 | 二进制 |
| 80h | 7Fh | 十六进制 |
| -128 | +127 | 十进制 |

**16位符号数**(在C语言中对应的数据类型为**short int**)

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| **1**0000000000000000 | **0**111111111111111 | 二进制 |
| 8000h | 7FFFh | 十六进制 |
| -32768 | +32767 | 十进制 |

**32位符号数**(在C语言中对应的数据类型为**long int**)

|  |  |  |
| --- | --- | --- |
| 最小值 | 最大值 | 进制 |
| 80000000h | 7FFFFFFFh | 十六进制 |
| -2147483648 | +2147483647 | 十进制 |

**零扩充及符号扩充问题：**

**非符号数扩充时，前面补0；**

**unsigned char a=255;   8位非符号数**

**short int b; 16位符号数**

**b=a;      b=0000 0000 1111 1111=255**

**被扩充的数必须是非符号数，才可以使用零扩充，**

**与扩充以后的数的类型无关。**

**被扩充的数必须是符号数，才可以使用符号扩充，**

**与扩充以后的数的类型无关。**

**符号数扩充时，前面补原数的最高位。**

**char a=-1;    8位符号数 1111 1111**

**short int b; 16位符号数**

**b = a;          b=1111 1111  1111 1111=-1**

**a=127;  0111 1111**

**b=a;    0000 0000 0111 1111=127**

|  |
| --- |
|  |
|  | |  | | --- | | IMG_257 | |

**其中偏置指数并非真正指数，**

**真正指数 = 偏置指数-127 = 133-127 = 6**

**尾数前面有一个隐含的起始位1及小数点，**

**实际尾数 = 1. 1111110 11000000 00000000**

**所以，根据指数值6把小数点右移6位得到：**

**1 111111.0 11000000 00000000 = 127.375**

## 2.5  符号扩展（Sign Extention）与零扩展（Zero Extention）

**当把8位非符号数转化成16位数时需要进行零扩充；**

**当把16位非符号数转化成32位数时也需要进行零**

**扩充。**

零扩充的规则是：扩充的各位一律为0。

符号扩充的规则是：把被转化数的最高位填充到扩充的各个位中。

## 2.6  字符的表示

(1) 在汇编语言中，字符与它的ASCII码是等价的。

**字符在本质上是一个8位数**。例如：**'A'与65等价**。

(2) 数字0-9与字符’0’-’9’的相互转化：

**数字+**'0'**=字符   3 + '0'   '3'**

**字符-**'0'**=数字   '8' – '0'  8**

 (3) 大小写字母的相互转化

**大写字母+20h=小写字母**

**小写字母-20h=大写字母**

**mov al, 'A'**

**add al, 20h ; AL='a'**

## 2.7  二进制数据的运算

### 2.7.1  算术运算

加、减、乘、除、求余

### 2.7.2  逻辑运算

**与、或、异或、非**

**1 and 1 = 1      1 or 1 =1    1 xor 1 = 0**

**0 and 1 = 0      0 or 1= 1    0 xor 1 = 1**

**1 and 0 = 0      1 or 0 = 1   1 xor 0 = 1**

**0 and 0 = 0      0 or 0 = 0   0 xor 0 = 0**

**① 与运算可以使某些位变0；**

**设a是一个8位数，要使该数的最低位与最高位都变0，**

**其它位不变：**

**?011 011?**

**0111 1110  and)**

**0011 0110**

**② 或运算可以使某些位变1；**

**?011 011?**

**1000 0001  or)**

**1011 0111**

**③ 异或运算可以使某些位反转；**

**1011 0110 最高位简称msb（sigificant）**

**1000 0001  xor) 最低位简称 lsb（不想变得位写为0）**

**0011 0111**

④ **异或运算还有以下规律：**

**假定X xor Y = Z，则有：**

**X xor Z = Y，并且 Y xor Z = X。**

**加密：X=“Who are you?”**

**Y=”123123123.....” XOR)**

**Z=”f.......”**

**（Z XOR Y可得到X）**

**两个二进制做异或时，可把异或当作加法或减法；**

**DES算法中用到XOR运算；**

### Regtest：若软件弹出的不是messagebox，而是一个标准的window，则可以在Destory window这个函数设断点；（也可在creatwindow，creatwindowex，showwindow这些函数设断点。）

在OD中ctrl+g,输入。。。，再按F2设断点

在OD中按：可以给某个地址命名一个标号。

在OD中给某条指令加注释，可按；键

### 2.7.3  移位运算

**左移： <<    shl: shift left**

**右移： >>    shr: shift right**

**Rol ax ，1循环左移**

**Ror ax，1循环右移**

左移、右移、循环左移、循环右移等。

# 第3章 8086/8088微型计算机系统组成

## 3.1  微型计算机系统概述

### 3.1.1  微型计算机硬件组成

### 3.1.2  微型计算机软件组成

### 3.2  8086/8088 CPU功能结构

### 3.2.1  8086/8088 CPU编程结构

**8086共有14个寄存器(register)，每个寄存器都是**

**16位:**

**AX, BX, CX, DX, （通用寄存器：算术，逻辑，移位运算**

**SP, BP, SI, DI,（用来保存偏移地址）**

**Stack pointer;base pointer;source index;destination idex;**

**CS, DS, ES, SS,        IP,        FL**

**段寄存器(用来存放段地址)  指令指针   标志寄存器**

**Code segment;data segment;extra segment;stack segment**

**远指针是指段地址、偏移地址合在一起构成的地址；**

**近指针是指仅有偏移地址构成的地址；**

**段长度=64K=2^16字节**

**IP：Instruction Pointer**

**FL: flag**

**IP=将要执行的那条指令的偏移地址**

**cs决定了将要执行的指令的段地址，ip决定了将要执行的指令的偏移地址。所以cs:ip（逻辑地址）指向将要执行的命令；ds决定了变量的段地址。**

**P.S.:**

**ds及cs在exe刚载入内存时指向程序段前缀（program segment prefix），psp共占100h字节，里面存放与当前exe相关的信息，如命令行参数。**

**80386以上的CPU有超过14个寄存器，除了段寄存器是**

**16位，其余寄存器都是32位:  e:extended**

**ax的高八位叫ah；低八位叫al；**

**EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI**

**CS, DS, ES, SS, FS, GS, EIP, EFL**

**(段寄存器，是16位)**

**P.S.:**

**32位cpu中，eax,ebx,ecx,edx,esi,edi,ebp,esp(都是32位寄存器）均可表示指针。eax中的e是extended，意思是对ax的扩充。eax的高16位没有名字，低16位是ax。**

**32位的间接寻址形式中增加寄存器乘以一个常数的形式，例如：**

**mov eax,dword ptr ds:[ebx+esi\*4]（16位中无乘法运算）**

**注意所乘系数只能是2，4，8。**

**32位cpu中段地址与16位有质的区别：**

**其中32位的段寄存器仍旧沿用16位的段寄存器名，**

**而且宽度也是16位，但增加了2个段寄存器：fs,gs。**

**例如ds=0010h,esi=12345678h则ds:esi对应的物理地址需要通过查表（GDT表）再计算得到。**

**gdt:global descriptor table 全局描述符表**

**gdt表其实是一个数组，数组中每个元素的宽度为8字节。**

**ds加上gdt首地址指向表中的第二个元素，提到其中的第2，3，4，7字节，逆过来，此值既是ds对应的段首地址**

**于是ds:esi对应的物理地址=12345678h+12345678h**

**64为cpu中，rax,rbx,rcx,rdx,rsi,rdi,rbp,rsp（都是64为寄存器)均可表示指针。**

### 3.2.2  8086/8088 的内部寄存器

  键盘输入的两种方法：

（1）getchar()方式：

Mov ah,1;

Int 21th;AL=所敲字符的ASCII码

输入时屏幕有回显（echo）

1. bioskey（）方式

Rom由post+bios构成；

Post：Power On Self Test

Bios:Basic input output system bios其实是一个函数集，里面包含键盘输入，屏幕输出，硬盘读写等函数。

Mov ah,1

Int 16h;检测键盘缓冲区是否为空

Jnz keybuf\_is\_not\_empty

Call refresh\_screen

Jmp next

Keybuf\_is\_not\_empty:

Mov ah,0

Int 16h;读取键盘缓冲中的键到AX中

\_;判断按了什么键

Jmp next

ZF是寄存器中FL中的一个位，zf=1时表示当前计算结果为0，zf=0表示当前结果非0；例如：sub ax,ax; FL中ZF=1；

Jz iszero;条件满足

## 字符的颜色：

0=黑色，1=蓝色，2=绿色，3=青，4=红，5=洋红，6=棕，7=白，

8=灰， 9=亮蓝， A B C D E F

背景色的最高位不是用来控制亮度，而是表示闪烁。

## 屏幕上如何画点：

先把显卡切换到图形模式，这里介绍最简单的图形模式，

编号13h（320\*200\*256色）,设置过程如下：

Mov ax,0013h；AH=00H,AL=13H

Int 10h;视频控制函数集

Mov ax,0A000h;

Mov ds,ax

Mov di,0

Mov al,4;红色

Mov ds:[di],al;屏幕（0，0）处会出现一个红点

Mov al,2;绿色

Mov ds:[di+1],al;屏幕（1，0)处会出现一个绿点

{对于给定的（x,y)，他对应的显卡偏移地址=y\*320+x}

要：

MOV ax,data

mov ds,ax  
不能对段寄存器直接给值（立即数）。  
同理，对于其它段寄存器一样不可直接操作〔给值（立即数〕）。

1．通用寄存器

（1）**数据寄存器AX、BX、CX、DX。**

**AH是AX的高8位，AL是低8位**

数据寄存器主要用于算术、逻辑运算。

（2）**变址寄存器SI、DI**。

（3）地址指针寄存器**SP、BP**。

**8086中SP有专门用途，不能放在[]中表示地址，**

**于是增加BX表示地址。因此，8086中只有**

**BX、BP、SI、DI可以放在[]中表示地址。**

**80386中取消了限制，除了S结尾的段寄存器、**

**EIP、EFL外都可以放到[]中表示地址。**

**地址          值**

**+1000:1000  34h ; 1234h的偏移地址是1000h**

**+1000:1001  12h**

**+1000:1002  56h**

**+1000:1003  78h**

**mov ax, 1000h**

**mov ds, ax ; DS=1000h**

**mov si, 1000h**

**mov al, byte ptr ds:[si]; AL=34h**

**mov al, byte ptr ds:[1000h] ; AL=34h**

**mov ax, word ptr ds:[1000h]; AX=1234h**

**ptr: pointer 指针**

**在程序中，段地址必须用段寄存器表示，不能用常数**

**mov ax, word ptr 1000h:[1000h] 错误**

变址寄存器与指针寄存器主要用来表示偏移地址，但也可以用于算术、逻辑运算。

(优先级：CS>DS>SS>ES）

2．**段寄存器CS、SS、DS、ES**

#### CS: code segment

**DS: data segment**

**SS: stack segment 堆栈段**

**ES: extra segment 附加段**

**段地址:[偏移地址] 用来表示该地址对应内存单元中的值**

**例如：设DS=1000h，BX=1234h**

**mov ah, ds:[1234h]; 直接寻址**

**mov ah, ds:[bx]    ; 间接寻址**

3．**控制寄存器IP、FL**

**IP: instruction pointer 指令指针**

**该寄存器与CS配合起来指向将要执行的那条指令。**

**其中CS表示段地址，IP表示偏移地址。**

**CS:IP  下一条指令**

**不能通过mov指令来改变CS及IP的值。**

**FL**称为**标志寄存器（Flags）**，**它是16位的寄存器，**

**但只用其中9位，这9位包括6个状态**

**标志和3个控制标志，如下所示：**

**O D I T S Z A P C**

**CF: 进位标志(carry flag)**

**mov al, 0FFh       1111 1111**

**add al, 2        1 0000 0010 +)**

**此时,CF=1        0000 0001**

**1111 1111**

**0000 0000 +）**

**此时CF=0 1111 1111**

**0000 0001**

**0000 0011 -)**

**0000 0100   此时CF=0**

**Mov　ａｌ，１　　００００　０００１**

**Ｓｕｂ　ａｌ，２　００００　００１０　－）**

**此时ｃｆ＝１**

**例如**

**Ｍｏｖ　ａｌ，０ＦＦｈ**

**Ａｄｄ　ａｌ，２**

**Ｊｃ　ｈａｓ＿ｃａｒｒｙ**

**．．．**

**Ｈａｓ＿ｃａｒｒｙ**

**．．．**

**移位运算中**

**如**

**Ｍｏｖ　ａｌ，８１ｈ；**

**ｓｈｌ　ａｌ，１**

**１０００　０００１＜＜１**

**（ｃｆ＝１）**

**１０００　０００１＜＜２**

**（Ｃｆ＝０）**

**右移类似**

**clc ; 强制使CF=0   CLear Carry flag**

**stc ; 使CＦ=1   SeT Carry flag**

**OF（Overflow Flag）溢出标志**：**两个正数相加结果是负数，**

**两个负数相加结果是正数就溢出，一正一负相加不会溢出**

**mov al, 7Fh ; 7Fh=+127**

**add al, 1   ; AL=80h =-128 此时OF=1, CF=0,**

**mov al,  80h**

**add al, 0FFh; AL=7Fh, OF=1, CF=1**

**1000 0000 =-128        80h+0FFh**

**1111 1111 =-1    +)**

**0111 1111 =+127    此时，OF=1, CF=1**

**CF=1表示非符号数溢出；**

**mov ah, 0FFh ; AH=255**

**add ah, 1    ; AH=0, CF=1, OF=0**

**符号数相加没有发生溢出：0FFh+02h=01h,CF=1,OF=0;**

**OF=1表示符号数溢出。**

**符号数运算结果错误时，就是溢出。**

**mov ah, 0FFh  ; 非符号数0FFh=255**

**add ah, 1     ; AH=0, CF=1**

**非符号数溢出使CF=1**

**符号数溢出使OF=1**

**JO**

**JNO**

**SF（Sign Flag）符号标志：**

**记录运算结果的符号，**

**结果为负时置1；否则置0。**

**SF=运行结果的最高位(符号数)**

**mov al, 1**

**add al, 2 ; AL=3=0000 0011, 此时SF=0**

**mov al, 0FFh**

**sub al, 1    ; AL=0FEh=1111 1110, SF=1**

**JS**

**JNS**

**ZF（Zero Flag）零标志:**

**运算结果为零时，ZF=1；否则，ZF=0。**

**sub ax, ax ; AX=0, ZF=1**

**add ax, 1  ; AX=1, ZF=0**

**MOV AH,1**

**INT 16H;(BIOSKEY(1))**

**INZ BUF\_HAS\_KEY**

**NO\_KEY\_IN\_BUF**

**....**

**BUF\_HAS\_KEY**

**mov指令不影响任何标志位**

**Mov ax,1**

**Sub ax,2;zf=0**

**Mov ax,0;zf=0**

**Jnz notzero;zf=0,跳转条件成立**

**CF  OF  ZF  SF  与这些标志位相关的有对应的**

**跳转指令：jc   jnc 跟cf有关（进位）**

**jo   jno  跟of有关（溢出）**

**jz  jnz 跟zf有关**

**js  jns 根sf有关**

**mov ax, 1**

**sub ax, 2 ; CF=1, SF=1, ZF=0, OF=0**

**jo  overflow ; jump if overflow 不成立**

**jc  less   ; jc: jump if CF==1  条件成立**

**js  negative ; js: jump if SF==1 条件成立**

**jz  zero   ; jz: jump if zero 条件不成立**

**jnz notzero; jnz: jump if not zero 成立**

**ZF:Je==jz、jne==jnz**

**Cmp ax,bx;（ax原值不变，内部有减法运算）**

**Je equal**

**Sub ax, bx；(ax原值会变）**

**Jz zero**

**mov指令不会改变任何标志位**

**mov ax, 2**

**sub ax, 1; AX=1, ZF=0(表示结果非零)**

**mov ax, 0; AX=0 不影响ZF，所以ZF=0**

**jz   zero ; 条件不成立**

**AF（Auxiliary Carry Flag）辅助进位标志**：

**记录运算时第3位（半个字节）产生的进位值或借位值。**

**第3位有进位或借位时，AF=1；否则，AF=0。**

**低4位向高4位产生进位或借位**

**mov al, 23h**

**add al, 1Fh ; AL=42h, AF=1**

**0010  0011**

**000111111 +)**

**0010**

**PF（Parity Flag）奇偶标志：**

**当结果操作数（一个字节数据或字数据）的**

**低8位中“1”的个数为偶数时，PF=1，否则，PF=0。**

**mov al, 2**

**add al, 1; AL=3=0000 0011, PF=1**

**Mov ax,1234h**

**Add ax,5678h;ax=68ACh**

**(只看低八位1的个数）**

**Jp==jpe（jump if parity even)**

**Jnp==jpo(jump if parity odd)**

**clc指令可视cf=0 stc可使cf=1**

**DF（Direction Flag）方向标志：控制字符串的操作方向**。

**当DF=0时为正方向(低地址到高地址)，当DF=1是反方向。**

**cld指令使DF=0， std指令使DF=1**

**若源数据首地址>目标数据首地址，则复制时要按正方向**

**(从低地址到高地址)；**

**若源数据首地址<目标数据首地址，则复制时要按反方向**

**(从高地址到低地址)；**

**标志位DF控制字符串操作的方向。当DF=0时为正方向，DF=1时为反方向。用指令cld使DF=0，用指令std使DF=1。**

**cld: clear direction**

**std: set direction**

**IF（Interrupt Flag）中断标志：当IF=1时，允许中断；否则，关闭中断。**

**cli指令使IF=0表示关/禁止中断；**

**sti指令使IF=1表示开/允许中断。**

**跟硬件中断有关，由硬件触发，**

**（int 21h 软件中断）**

**（硬件可自动添加中断，如：**

**int 09h 键盘中断**

**Int 08h 时钟中断）**

**如想禁止中断可在代码段前加上：**

**Cli（禁止中断）**

**....**

**sti**

**TF（Trace/Trap  Flag）跟踪/陷阱标志：**

**用于程序调试时进行单步(single-step)方式工作。**

## 3.3  8086/8088的内存组织

### 3.3.1  内存单元的地址和内容

### 3.3.2  内存地址的分段

**8086一共有20位地址线，最多只能表示2的20次方个地址。**

**8086的寄存器只有16位，无法存放20位物理地址。**

**段长度=64K     段地址=段起始地址的高16位**

**段起始地址的16进制值的最右边一位必须是0。**

**段地址:偏移地址称为逻辑地址。**

**偏移地址：某个内存单元地址与段起始地址的相对距离**

|  |
| --- |
|  |
|  | |  | | --- | | IMG_258 | |

**1234:0058 = 1200:0398**

**2233F+1-12340=22340-12340=10000h=64K**

**1234:0058 = 1236:0038 = 1233:0068**

**1234:0058=12340+0058=12360+0038**

### 3.3.3  逻辑地址与物理地址

    逻辑地址转化为物理地址：

物理地址 = 段起始地址 + 偏移地址

= 段地址\*10h + 偏移地址

### 3.3.4 数据在存储器中的存放

**对于长度超过1个字节的数据如字、双字，它们在内存中**

**按低字节在前、高字节在后的规律存放。**

### 3.3.5  外部设备

CPU   **端口(port)**   I/O设备

60h端口与键盘相关

In al,60h;可以读到当前的键码

Out 61h,al;把al的值发送到61h端口

70h 71h是与内部时钟相关的2个端口编号

70h是cmos的地址端口

71h是cmos的数据端口

Mov al,0

Out 70h,al

In al,71h

**端口地址的分配独立于内存地址的分配，**

**并且端口地址的范围是：[0000h, 0FFFFh]，**

**共65536个端口。对端口操作使用指令in与out实现。**

**code segment**

**assume cs:code**

**main:**

**mov al, 0FEh**

**out 64h, al ; 把AL的值输出给64h号端口**

**next:**

**jmp next**

**code ends**

**end main**

# 第4章  8086/8088的寻址方式和指令系统

## 4.1  寻址方式

### 4.1.1 指令结构

**汇编语言指令（instruction）一般由操作码（opcode）与**

**操作数（operand）两部分构成，即：**

**指令 = 操作码 + 操作数**

**Mov al,12h**

**Inc ax(表示ax=ax+1)**

### 4.1.2  操作数寻址方式（operand-addressing modes）

**1．立即数方式（immediate mode）**

如果操作数为**常数**，则该操作数称为

**立即数（immediate data）**。例如：

**mov ah, 2       ; 2 为立即数**

**add bx, 1234h ; 1234h 为立即数**

**jmp 10ABh      ; 10ABh 为立即数**

2．**寄存器方式（register mode）**

**寄存器方式是以寄存器作为操作数，例如：**

**inc bp       ; bp 是寄存器**

**add ax,bx   ; ax和bx 是寄存器**

**jmp dx       ; dx 是寄存器**

3．**直接寻址（direct memory addressing）**

**无论是直接寻址还是间接寻址，段地址都不可以是常数，只能是某个段寄存器。**

**如果操作数的偏移地址用一个立即数表示，**

**则这种寻址方式称为直接寻址。**例如：

**mov ds:[10ABh], al 等价于**

**mov byte ptr ds:[10ABh], al**

**mov ds:[10ABh], 1  错误！**

**mov byte ptr ds:[10ABh], 1; 正确**

**mov word ptr ds:[10ABh], 1; 正确**

**4．间接寻址（indirect memory addressing）**

**Sp,bp,si,di用来表示偏移地址；**

**Cs,ds,es,ss用来表示段地址；**

**Ss:sp用来指向堆栈顶端**

**通用寄存器中bx拿来替换sp作偏移地址寄存器的作用，但他同时又是一个通用寄存器，可以做算术，逻辑，移位运算。【bx][bp][si][di】就成了最简单的间接寻址方式。**

**注意【sp]是错误的。**

**[bx] [bp]  [si]  [di]**

**[bx+2]  [bp-2]  [si+3]  [di-4]**

**[bx+si]  [bp+si]  [bx+di]  [bp+di]**

**[bx+si+1]  [bp+si-2] ...**

**如果操作数的偏移地址是以间接的形式表示，**

**即操作数的偏移地址中含有基址寄存器（BX，BP）或**

**变址寄存器（SI，DI），则这种寻址方式称为间接寻址。**

**间接寻址的一般形式**是：

**基址寄存器    + 变址寄存器    +       位移量**

**（BX或BP） （SI或DI）** （**8位或16位符号数**）

**mov ax, word ptr ds:[bx+si+2]**

**mov dx, word ptr ds:[bp+di-3]**

**mov ax, word ptr ds:[bx+bp+2]  错误**

**mov ax, word ptr ds:[si+di-3]  错误**

**mov ax, word ptr ds:[bx+2]**

**mov ax, word ptr ds:[si+2]**

**mov ax, word ptr ds:[bx]**

**mov ax, word ptr ds:[si]**

**mov ax, ds:[bx+si+2]**

**在TC中可以把.c编译成.asm：**

**1. 先进入DOS方式**

**2. C:**

**3. cd \tc**

**4. tcc –S 程序名.c**

**结果会生成“程序名.asm”**

**32位汇编语言中还可以有如下的寻址方式：**

**[ebx+esi\*4+8]**

**[ebx+esi\*2+8]**

**[ebx+esi\*8+8]**

**[eax+edx\*4-2]   正确**

**[esp+2]        正确**

**[eax+eax\*4+5] 正确**

**[ebx+esi\*3+8]  错误**

**默认段地址规则：**

**凡是间接地址中含有寄存器BP、EBP、ESP，则**

**默认段地址为SS，否则默认段地址是DS。**

**mov ax, [bx+2]相当于mov ax, ds:[bx+2]**

**mov ax, [bp-2]相当于mov ax, ss:[bp-2]**

**mov eax, [esp-4]相当于mov eax, ss:[esp-4]**

**mov eax, [ebp+8]相当于mov eax, ss:[ebp+8]**

**mov eax,ds:[ebp+8]改变默认段地址为DS**

### 4.1.3  段跨越（segment overriding）

**通过在操作数前添加一个段前缀（segment prefix）如CS:、DS:、ES:、SS:来强制改变操作数的段址，这就是段跨越。**

**当【】中包含有寄存器bp时，该变量的段地址一定是ss；否则段地址一定是ds。例如：**

**Mov ax,[bp+2] 相当于 mov ax,ss:[bp+2]**

**mov ax, ds:[bx+si+2]**

**变量的段地址必须用段寄存器表示，不能用常数。**

**设DS=1000h，不能写成：**

**mov ax, 1000h:[bx+si+2] 错误**

**如果变量的偏移地址中不含有寄存器BP，则**

**默认的段地址是DS；如果偏移地址中BP，则**

**默认的段地址是SS。**

**mov ax, ds:[bx+si+2] 可以简化为**

**mov ax, [bx+si+2]**

**mov ax, ss:[bp+2] 可以简化为**

**mov ax, [bp+2]**

**默认的段地址是可以改变的，例如要把【bp+2】默认的ss改成ds则，mov ax,[bp+2]改写为：mov ax,ds:[bp+2]**

## 4.2  8086/8088的指令系统

### 4.2.1  数据传送指令

1．通用数据传送指令：**MOV**，**PUSH，POP，XCHG**

**mov ax, 1**

**mov byte ptr ds:[bx], 1**

**mov byte ptr ds:[bx], ah**

**mov ax, bh; 错误，左右操作数宽度必须相等**

**xor ax, ax**

**mov al, bh**

**mov ds, 1000h; 错误，因为段寄存器不能赋值为常数**

**正确写法：**

**mov ax, 1000h**

**mov ds, ax**

**Mov ds:[bx+si+2],1或mov [bx+si+2],1错误**

**Mov byte ptr [bx+si+2]，1；正确（常数没有明确的宽度：左右需要有一个宽度明确）**

**mov word ptr ds:[1234h], 1000h**

**mov ds, word ptr ds:[1234h]**

**mov byte ptr ds:[1234h],byte ptr ds:[100h]**

**错误，因为两个操作数不能均为内存变量。**

**mov cs, ax; 错误，因为CS不能通过mov改变。**

**Mov ax,cs 是正确的**

**CS:IP 指向将要执行的下一条指令。**

**同理，IP也不能通过mov改变。**

**mov ax, ip**

**1000：2000 jmp 2006h**

**1000:2002 mov ah,1**

**1000:2004 mov al,2**

**...**

**3000:2006 mov bx,3(ip的值改变为2006，cs变为3000）**

**push 与 pop 指令**

**堆栈按先进后出规则压入弹出数据**

**16位的CPU中，SS:SP指向堆栈顶端**

**设SS=1000, SP=2000, AX=1234, BX=5678**

**设SS=1000,ESP=2000,EAX=12345678h,**

**EBX=11223344h 设AX=1234, BX=5678**

**1000:1FF8              44②**

**1000:1FF9              33**

**1000:1FFA              22**

**1000:1FFB              11**

**1000:1FFC 78②      78①③**

**1000:1FFD 56          56**

**1000:1FFE 34①③    34**

**1000:1FFF 12          12**

**1000:2000   ④        ESP④**

**① push eax ② push ebx**

**③ pop eax ④ pop ebx**

**现在执行指令①push ax再执行②push bx**

**再执行③pop ax(结果AX=5678)，**

**最后执行④pop bx(结果BX=1234)**

**不能对8位值进行push/pop: push AH 错误**

**xchg op1, op2  ；交换（exchange）指令**

**mov ax, 1            ; AX=1**

**mov bx, 2         ; BX=2**

**xchg ax, bx      ; AX=2, BX=1**

**2．输入输出指令**：**IN，OUT   端口(port)**

**in al, 21h  从21h号端口读取一个字节存放到AL中**

**in al, 378h 错误，因为端口地址>=100h，必须存放到DX**

**mov dx, 378h; 当端口地址>=100h时端口地址必须放入dx中，而不能用常数形式表达**

**in al, dx**

**out 378h, al 错**

**mov dx, 378h**

**out dx, al    正确**

**out 21h, al   正确**

**3．地址传送指令：LEA，LDS，LES**

**（1）lea  dest, src**

**lea eax, [ebp-64]**

**EAX=EBP-64**

**LEA ；装入有效地址**（**Load  Effective Address**）

**理解为：取变量的偏移地址**

**lea dx, ds:[1000h] ; DX=1000h**

**Lea dx,abc 等价于**

**Lea dx,offset abc**

**（但lea支持间接寻址）**

**Lea dx,ds:[bx+si+3];{dx=bx+si+3}**

**Lea eax,[eax+4\*eax];eax=eax\*5**

**Mov dx,bx+si+3;错误**

**设BX=1000h, SI=2**

**lea ax, ds:[bx+si+3]; AX=bx+si+3=1005h**

**（2）lds dest, src**

**操作：**

**dest = word ptr [src] 偏移地址**

**ds = word ptr [src+2] 段地址**

**1000:0000 78h**

**1000:0001 56h**

**1000:0002 34h**

**1000:0003 12h**

**12345678h**

**假定要把1000：0000中存放的远指针取出来，存放到ds:bx中，则**

**Mov ax,1000h**

**Mov ds,ax**

**Lds bx,dword ptr ds:[0000h]**

**;ds=1234h,bx=5678h**

**（3）les dest, src**  ；把远指针装到ES:dest

；（Load ES and dest with a far pointer）

操作：

dest = word ptr [src]

es = word ptr [src+2]

4．**标志寄存器传送指令：PUSHF，POPF**

**pushf**

**pop ax ; AX=FL**

**or ax, 1; 第0位变1，其它位不变**

**push ax**

**popf    ; FL=AX**

**(P.S.一个段的宽度必须是10h的倍数**

**堆栈段的定义方法：**

**Data segment**

**Abc db 1,2,3**

**Xyz dw 2345h,5678h**

**Data ends**

**S segment stack**

**Db 100h dup(0)**

**S ends**

**Code segment**

**Assume cs:code, ds:data,ss:s**

**Main:**

**Mov ax,data**

**Mov ds,ax**

**Push xyz[0]**

**Pop xyz[2]**

**Mov ah,4Ch**

**Int 21h**

**Code ends**

**End main**

### 4.2.2  转换指令

1．**符号扩充指令**：**CBW，CWD**

**CBW: convert byte to word**

**CWD: convert word to double word**

**CDQ: convert double word to**

**quadruple word(32位扩充为64位)**

**mov al, 7Fh           mov al, 0F9h**

**cbw    ; AX=007F      cbw; AX=0FFF9h**

**Dx:ax用来表示一个三十二位数，dx是高16位，ax是低16位。**

**mov ax, 7FFFh**

**cwd            ; DX=0000 AX=7FFFh**

**mov ax, 8000h**

**cwd   ; DX=0FFFFh, AX=8000h**

**mov eax, 0FFFFFFFFh**

**cdq  ; EDX=0FFFFFFFFh, EAX=0FFFFFFFFh**

**高32位           低32位**

**设AL=0F9h是一个非符号数，要把它扩充到AX中**

**mov al,0F9h ; AL=F9h**

**movzx ax, al; AX=00F9h**

**movzx: move by zero extention**

**movsx ax, al; AX=0FFF9h  相当于cbw**

**movsx: move by sign extention，效果等同于cbw**

**mov ax, 0F987h**

**movzx eax, ax; EAX=0000F987h**

2．**换码指令：XLAT  (translate)也称查表指令**

**char t[]="0123456789ABCDEF";**

**char i;**

**i = 10;**

**i = t[i]; 最后i='A'**

**mov bx, offset t**

**mov al, 10**

**xlat; 结果AL='A'**

**实际AL=DS:[BX+AL]**

**Xlat指令要求DS:BX指向数组**

4.2.3  算术指令

1．加法指令：**ADD，INC，ADC**

**mov ax,1234h       mov ax, 1234h**

**mov bx,5678h       add ax, 5678h**

**add ax, bx; AX=AX+BX**

**add word ptr ds:[bx], word ptr ds:[1080h]错误**

**add ax, ds:[bx]    add ds:[bx], ax**

**mov ax, 12**

**inc ax; AX=AX+1=13**

**inc指令不影响CF位，add指令会影响CF位**

**mov al, 0FEh**

**add al, 1    ; AL=0FFh , CF=0**

**inc al        ; AL=00h  , CF=0正确**

**例4.4  求2F365h与5E024h的和，结果存放到寄存器DX及AX中，要求DX存放结果的高16位，AX存放结果的低16位。**

**mov ax, 0F365h ; AX = 2F365h的低16位**

**mov dx, 0002      ; DX = 2F365h的高16位**

**add ax, 0E024h; CF=1**

**adc dx, 5 ; 两数的高16位相加,并加上低16相加时产生的进位 ; DX = 2+5+1 =8**

**; DX:AX = 8D389h**

2．减法指令：**SUB，SBB，DEC，NEG，CMP**

**dec 自减，相当于C语言中的--运算。**

**neg  op ; 求补（negate），求相反数**

**mov ah, 2**

**neg ah    ; AH = -2 = 0FEh**

**mov ah, 0FFh**

**neg ah    ; AH=1**

**注意neg与not的区别：**

**not在C语言中相当于~**

**mov ah, 2;    0000 0010**

**not ah   ; AH=1111 1101B=0FDh**

**cmp指令：比较(compare)**

**sub ax, bx; AX的值会变化=原AX-BX**

**cmp ax, bx; 也会计算ax-bx，但差不**

**存放到AX中，因此AX、BX不变**

**标志寄存器FL中的一些位会发生变化，**

**例如CF、ZF、OF、SF**

**设ah=0FFh,bh=1**

**Cmp ah,bh**

**Jg ah\_is\_greater\_than\_bh;跳转条件不满足**

**Ja ah\_is\_above\_bh;跳转条件满足**

**非符号数大小比较指令：ja,jb,jae,jbe,je,jne**

**符号数大小比较跳转指令：jg,jl,jge,jle,je,jne**

**CF=0且ZF=0时，ja条件满足**

**CF=1式，JB条件满足**

**当SF==OF时，jq条件满足**

**折AH=3,BH=2**

**Cmp ah,bh;SF=0,OF=0 ->jg条件满足**

**设ah=7Fh,bh=80h**

**Cmp ah,bh;SF=1，OF=1 ->jg条件满足**

**Ah=2,bh=3**

**Cmp ah,bh;SF=1,OF=0 ->JL条件满足**

**Ah=80h,bh=7Fh**

**Cmp ah,bh;SF=0,OF=1 ->JL条件满足**

**SF！=OF式，jl条件满足**

**与CF、ZF、SF、OF相关的跳转指令：**

**jc jnc  jz jnz  js jns  jo jno**

**另外，je == jz   ，  jne == jnz**

**je:jump if equal   jne:jump if not equal**

**设AX=1， BX=2,则**

**cmp ax, bx;  CF=1, ZF=0, OF=0, SF=1**

**jc  ax\_is\_less\_than\_bx 条件成立**

**jb ax\_is\_less\_than\_bx 条件成立**

**jc与jb指令完全等价。**

**jz与je指令完全等价。**

**对于非符号数减法或比较：**

**若CF=1  AX < BX    相关指令jb或jc**

**若ZF=1  AX == BX   相关指令je或jz**

**若CF=0且ZF=0  AX > BX  相关指令ja**

**对于符号数减法或比较：**

**若SF==OF  AX > BX 相关指令jg (greater)**

**若ZF=1    AX==BX  相关指令je或jz**

**若SF≠OF  AX < BX 相关指令jl (less)**

**①SF=OF=0 设AX=3,BX=2**

**cmp ax, bx; SF=0, OF=0**

**ax-bx是正数且结果无错AX > BX**

**②SF=OF=1 设AL=7Fh, BL=80h**

**cmp al, bl; SF=1, OF=1**

**al-bl是负数但结果有错AL > BL**

**③SF=0,OF=1 设AL=80h,BL=1**

**cmp al, bl; SF=0, OF=1 AL < BL**

**④SF=1,OF=0 设AL=1, BL=2**

**cmp al, bl; SF=1, OF=0 AL < BL**

**非符号数比较相关的跳转指令：**

**ja  jb  jae  jbe  je  jne**

**符号数比较相关的跳转指令：**

**jg  jl  jge  jle  je  jne**

**3．乘法指令**：**MUL(非符号数乘法)，IMUL(符号数乘法)**

（1）**mul  src**; **非符号数乘法**（unsigned multiplication）

操作：**multiply**

**8位 \* 8位 = 16位**

**16位 \* 16 位 = 32位**

**32位 \* 32 位 = 64位**

**被乘数固定为某个寄存器，**

**乘积也固定存放于某些寄存器中**

IMG_259**AX = AL \* src**  ; **当src为字节（8位数）时**

**DX:AX = AX \* src** ; **当src为字（16位数）时**

**EDX:EAX = EAX \* src  ; 32位乘法**

**注意：src不可以为常数，只能是寄存器或变量**

**mov al, 10h**

**mov bl, 12h**

**mul bl ; AX=0120h**

**mul指令后面不能跟常数，只能跟寄存器或内存变量。**

**mul byte ptr ds:[bx] 是正确的**

**mov ax, 1234h**

**mov bx, 1000h; 0123 4000h**

**mul bx ; DX=0123h    AX=4000h**

**; DX:AX=01234000h**

**mov eax, 12345678h**

**mov ebx, 10000h**

**mul ebx; EDX=00001234 EAX=56780000**

**mov dx, 1234h**

**mov ax, 2h**

**mov bx, 34h**

**mul bx       ; DX=0000h, AX=0068h**

**4．除法指令：DIV，IDIV**

（1）**div  op ;** 非符号数除法（unsigned division）

**(1) 16位AX / 8位  = 8位AL(商)..8位AH(余数)**

**(2) 32位DX:AX / 16位 = 16位AX .. 16位DX**

**(3) 64位EDX:EAX / 32位 = 32位EAX .. 32位EDX**

**mov ax, 0123h**

**mov bl, 10h**

**div bl ;**

**; AL=123h/10h=12h,**

**; AH=123h%10h=03h**

### 4.2.5  逻辑运算和移位指令

1．**逻辑运算指令：AND，OR，XOR，NOT，TEST**

（1）**and  dest, src**    ; 二进制与运算（Boolean AND）

**and == &**

**or  == |**

**xor == ^**

**not == ~**

**mov ah, 12h**

**and ah, 0Fh; AH=02h**

**mov ah, 0FFh**

**not ah ; AH=0**

**xor ax, ax; 相当于 mov ax, 0**

**mov ah, 0Ah ; 1010 and 0001**

**test ah, 1 ; 计算AH and 01，结果不存放到AH**

**; 但结果会影响标志位，如ZF=1**

**jz bit0\_is\_zero 条件成立**

**test ah, 80h; 10000000B**

**jnz bit7\_not\_zero**

2．**移位指令**：**SHL**，**SHR**，**SAL**，**SAR**，

**ROL，ROR，**

**RCL，RCR**

**shl逻辑左移(shift left): 针对非符号数**

**shr逻辑右移(shift right): 针对非符号数**

**sal算术左移(shift arithmetic left):对符号数 等价于shl**

**sar算术右移(shift arithmetic right):对符号数**

**！=shr**

**mov ah, 3; 0000 0011**

**shl ah, 1; AH=0000 0110=6 相当于乘2**

**{**

**在32位环境下，假定要对12345678 9876ABCDh逻辑左移1位**

**Mov eax,9876ABCDh**

**Shl eax,1;CF=1**

**Mov edx,CF=1 12345678h**

**Rcl edx,1**

**}**

**mov ah, 9; 0000 1001**

**shr ah, 1; 0000 0100=4 相当于除以2**

**mov ah, 9**

**shr ah, 2; 在8086中是错误的，但在386以上**

**; CPU中是正确的。**

**mov cl, 2  ; CL=移位次数(当次数>=2时)**

**shr ah, cl; 这是8086中的正确写法**

**0000 1001 右移1位后变成 0000 0100，**

**其中第0位移出后进入CF中**

**mov ah, 5; 0000 0101**

**shr ah, 1; AH=0000 0010, CF=1**

**SAL 等价于 SHL**

**mov ah, 5; 0000 0101**

**sal ah, 1; 0000 1010  右边补0**

**mov ah, 0FFh; 1111 1111=-1**

**sal ah, 1; 1111 1110=-2**

**mov ah, 80h; 1000 0000**

**sal ah, 1; 0000 0000**

**mov al, 0FAh ; 或写成 mov al, -6**

**sar al, 1 ; 1111 1010  1111 1101**

**算术右移时，左边必须补原数的最高位**

**例4.6  把32位数3F65 C932h逻辑左移2位，**

**结果存放到DX:AX中。**

**设DX=3F65，AX=C932**

**mov cl,2**

**mov bx, ax; BX=1100 1001 0011 0010**

**shl ax, cl**

**shl dx, cl; …………00**

**mov cl, 14**

**shr bx, cl; BX=0000 0000 0000 0011**

**or dx, bx**

**mov ax, 0C932h;1100 1001 0011 0010**

**mov dx, 3F65h; 0011 1111 0110 0101**

**mov cx,2**

**again:**

**shl ax, 1; 100 1001 0011 00100,CF=1**

**rcl dx, 1; 011 1111 0110 01011,CF=0**

**dec cx**

**jnz again**

### 4.2.6  字符串操作指令

1．字符串传送指令：MOVSB

（1）**[rep] movsb** ; 以字节为单位

传送字符串（**move string byte**）

**movsb 表示移动1个字符**

**rep movsb 表示移动CX个字符**

**DS:SI指向源字符串 (SI:source index)**

**ES:DI指向目标字符串(DI:destination index)**

**Cx=移动次数**

**DF=0即方向标志设成正方向（用指令cld)**

**SI: source index**

**DI: destination index**

**movsb的操作过程：**

**1. byte ptr ES:[DI] = byte ptr DS:[SI]**

**2. 若DF=0即正方向则SI++，DI++；**

**若DF=1即反方向则SI--,DI--**

 Rep movsb所做的操作如下：

again：

if（cx==0)

Goto done

Byte ptr es:[di] = byte ptr ds:[si]

If(df==0)

{si++;di++)

Else

{si--;di--}

Cx--

Goto again

Done:

2．**字符串比较指令**：**CMPSB**

**cmpsb**

**repe cmpsb**

**(若本次比较相等则继续比较下一个)**

**repne cmpsb**

**(若本次比较不等则继续比较下一个)**

**repe  cmpsb 的操作**

**again:**

**if ( CX == 0 )     ; 若剩余比较次数为0，**

**goto  done   ; 则跳转至done结束字符串比较操作**

**endif            ; 若CX不等于0则继续**

**NULL = byte ptr DS:[SI] - byte ptr ES:[DI]**

**; 比较字节DS:[SI]与ES:[DI]**

**temp = FL       ; 保存比较后的标志位**

**if ( DF == 0 )        ; 若方向位为0（正方向）**

**SI = SI + 1    ; DS:SI指向源字符串的下一个字节**

**DI = DI + 1   ; ES:DI指向目标字符串的下一个字节**

**else             ; 若方向位为1（反方向）**

**SI = SI - 1    ; DS:SI指向源字符串的上一个字节**

**DI = DI – 1   ; ES:DI指向目标字符串的上一个字节**

**endif**

**CX = CX - 1      ; 剩余比较次数减1**

**FL = temp**

**if ( ZF == 1 )  ; 若比较时字节DS:[SI]与ES:[DI]相等，则**

**goto  again ; 跳转至again继续比较**

**endif**

**done:**; 结束字符串比较操作

**repne cmpsb 的操作**

    again:

        if ( CX == 0 )   ; 若剩余比较次数为0，

           goto  done      ; 则跳转至done结束字符串比较操作

        endif         ; 若CX不等于0则继续

        NULL = byte ptr DS:[SI] - byte ptr ES:[DI]

; 比较字节DS:[SI]与ES:[DI]

        temp = FL          ; 保存比较后的标志位

        if ( DF == 0 )       ; 若方向位为0（正方向）

           SI = SI + 1      ; DS:SI指向源字符串的下一个字节

           DI = DI + 1      ; ES:DI指向目标字符串的下一个字节

        else                ; 若方向位为1（反方向）

           SI = SI - 1       ; DS:SI指向源字符串的上一个字节

           DI = DI - 1       ; ES:DI指向目标字符串的上一个字节

        endif

        CX = CX - 1        ; 剩余比较次数减1

        FL = temp

; 恢复刚比较字节DS:[SI]与ES:[DI]后的标志位

        if ( ZF == 0 ) ; 若比较时字节DS:[SI]与ES:[DI]不相等，则

           goto  again     ; 跳 转至again继续比较

        endif       ; 若比较时两者相等，则不再进行下次比较

    done:                  ; 结束字符串比较操作

如果字符串全等： 最后一个元素不相等的情况

Repe cmpsb

Je.....

(不要用cx==0)

Cmpsb cmpsw(16位) cmpsd(32)

3．**字符串扫描指令：scasb**

（1）[repe | repne] scasb ;

Scasb:

Cmp al,es:[di]

Inc di;(当DF=1时为di--)

Repe scasb:

Next:

If(cx==0) goto done;

Cmp al,es:[di]

Inc di

Cx--

Jne done

Goto next

Done:

{P.S. C中p=strchr(s,’x’)在字符串s中寻找‘x’字符，返回地址

P=strstr(s,t)在字符串s中寻找子串t}

**在字符串中查找一个字符（scan string byte）**

操作：  **scan  string  byte**

**设ES:DI以'\0'结束的字符串，要求该字符串长度(字符串的长度不包括0）**

**mov ax, 1000h**

**mov es, ax**

**mov di, 1080h; ES:DI"ABC..."**

**mov cx, 0FFFFh; ES:DI"ABC",0,[]...**

**mov al, 0 ; AL=待查找的字符**

**cld**

**repne scasb; 循环结束时，DI=1088h, CX=FFFF-8**

**inc cx**

**not cx; CX=FFFF-CX=字符串长度(不含0)**

Repe scasw(10h,11h,22h.......)

寻找2211h，可循环两次

4．存入字符串指令：**STOSB**

**store string byte**

**stosb把AL的值保存到ES:DI所指向的内存单元中，**

**当DF==0时DI++，当DF==1时DI--**

**rep stosb 循环CX次stosb**

**设要把从地址1000:10A0开始共100h个字节**

**内存单元全部填0**

**mov ax, 1000h**

**mov es, ax; ES=1000h**

**mov di, 10A0h**

**mov cx, 100h          mov cx, 80h**

**cld                     cld**

**xor al, al             xor ax, ax**

**rep stosb**

**Again:**

**If(cx==0) goto done;**

**Es:[di] = al**

**Di++;DF=1时，di-2**

**Cx--**

**Goto again**

**Done:**

**rep stosw rep stosd**

 Ax,+2 eax,+4

5．从字符串取字节或字指令：LODSB

（1）**lodsb**              ; 从DS:SI字符串取字节（load string byte）

**load string byte**

**AL=DS:[SI], SI++**

**DF==0 正方向 AL=DS:[SI],SI++**

**DF==1 反方向**

**一般情况下悟rep前缀**

### 4.2.7  控制转移指令

1．无条件跳转指令：JMP   **相当于C语言中 goto语句**

（1）jmp  dest  ; **短跳（short jump）**

短跳的目标地址idata与该短跳指令的下条指令的偏移地址之间的距离必须落在范围**［-128，+127］**之内。

**目标偏移地址-下条指令偏移地址=跳转距离**

**地址            机器码   汇编指令**

**1D3E:0100     EB06     jmp  0108h**

**1D3E:0102     B402     mov  ah，2**

**1D3E:0104     B241     mov  dl, 41h**

**1D3E:0106     CD21     int  21h**

** 1D3E:0108     B44C     mov  ah，4Ch**

**1D3E:010A     CD21     int  21h**

**短跳指令机器码中第1字节恒为EB，**

**第2字节是目标地址与下条指令偏移地址的差。**

**短跳指令除了jmp外，还包括条件跳转指令：**

**jc  jz  js  jo   jnc  jnz   jne  jno  jns**

（2）jmp  dest  ; **近跳（near jump）**

**近跳的跳转距离用2个字节表示。（段内跳跃）**

**地址        机器码      汇编指令**

**1D3E:0100   E9FD1E      jmp  2000h**

**1D3E:0103   B44C         mov  ah，4Ch**

**1D3E:0105   CD21         int  21h**

**（E9为jump near**

**EB为jump short)**

**Jmp 后面可加数字，寄存器，变量**

**如jmp 1234h**

**Mov bx,178ch**

**Jmp bx**

**Mov word ptr ds:[10F0h],2F08h**

**Jmp word ptr ds:[10F0h]**

**P.S.:jmp bx 与jmp word ptr [bx]的区别**

**Tasm中m#（#为数字，开关设置多次扫描）**

**（3）jmp  dest    ; 远跳（far jump）**

例如：                   **远指针/远地址**

**1000:2000    jmp  1234h:5678h ; 段地址用常数表示**

**; 执行此跳转指令后，CS=1234h，IP=5678h**

**mov  word   ptr  ds:[10F0h], 5678h**

**mov  word   ptr  ds:[10F2h], 1234h**

**jmp  dword  ptr  ds:[10F0h]   1234:5678**

 EA远跳，直接接地址，不接变化量

远跳后面可以跟一个地址也可以跟一个变量

MASM或TASM写源程序表达远跳时，直接写成

Jmp 1234h:2678h编译不成功

应该写成

Db 0EAh;表示远跳指令的机器码

Dw 5678h,1234h;或者dd 12345678h

1. **条件跳转指令**

**(都是短跳）**

**jz  dest**

**jnz dest**

**jc  dest**

**jnc dest**

**js  dest**

**jns dest**

**jo  dest溢出**

**jno dest**

**jp dest奇偶校验**

**jnp dest**

**以下格式与非符号数大小比较有关：**

**ja  dest**

**jae dest**

**jb  dest**

**jbe dest**

**以下格式与符号数大小比较有关：**

**jg  dest**

**jge dest**

**jl  dest**

**jle dest**

**以下格式与非符号数及符号数的相等／不相等比较有关：**

**je  dest**

**jne dest**

**jcxz  dest**    ; 若CX等于零则跳转（jump if CX is zero）

3．循环指令：LOOP

**loop  dest ; 循环**

操作：

**CX = CX - 1   ; 循环次数减1**

**if ( CX !=0 )   ; 若CX不等于0，则**

**goto  dest   ; 跳转至dest**

**endif         ; 若CX等于0，则结束指令执行**

Int f(int n)

{

If(n==1)

Return 1;

Else

Rerurn n+f(n-1);

}

F:

Push bp

Mov bp,sp

Mov ax,[bp+4]

Cmp ax,1

Je done

Dec ax

Push ax

Call f

Add sp,2

Add ax,[bp+4]

Done :

Mov sp,bp

Pop bp

Ret

(call函数时会将下一条指令的地址压入栈中）

4．**子程序调用与返回指令：CALL，RETN，RETF**

**其中retn表示近返回，可简写成ret;**

**retf表示远返回。**

**call既可以表示近调用，也可以表示远调用**

（1）**call  dest**; 近过程调用（near procedure call）

操作：

**push** **下条指令偏移地址**     **; 压入堆栈**

**jmp dest**; 跳转至dest

（2）retn [count]  ; 近返回（near return）

retn指令一般用于从近调用指令call调用的近过程中返回。为了书写方便，我们通常用ret来代替retn。

操作：

    ① ret的操作

**pop  IP**              ; 从堆栈中弹出返回地址到IP

    ② ret  count的操作

        pop  IP               ; 从堆栈中弹出返回地址到IP

        SP = SP + count        ; 从堆栈中弹出count个字节

用堆栈传递参数又有三种方式：  
A.参数从右到左顺序压入堆栈，由调用者清理堆栈

B.参数从左到右顺序压入堆栈，由调用者清理堆栈——pascal

C.参数从右到左顺序压入堆栈，由被调用者清理堆栈———stdcall

P.S.:

混合语言编程

#include<stdio.h>

int main(void)

{

int a=10,b=20,c;

\_asm

{

mov eax,a

add eax,b

mov c,eax

}

printf("%d",c);

return 0;

}

#include<stdio.h>

//declare specification

\_\_declspec(naked) int f(int a,int b)

{

\_\_asm

{

push ebp

mov ebp,esp

mov eax,[ebp+8]

add eax,[ebp+0Ch]

//mov esp,ebp

pop ebp

ret

}

}

int main(void)

{

int a,b=20,c;

c=f(10,20);

printf("%d",c);

return 0;

}

5．**中断和中断返回指令**：int, iret

（1）**int  n**   ; 中断（interrupt）

中断指令一般用于向系统如DOS和BIOS提出中断请求以得到系统的服务。例如调用DOS中断int 21h可以请求DOS为用户程序提供如输入字符、输出字符、打开文件、分配内存等服务。

**1．近调用在跳转到目标地址前，把下条指令的偏移**

**地址压入堆栈；用ret/retn返回**

**2．远调用在跳转到目标地址前，把下条指令的段地址**

**及偏移地址压入堆栈；用retf返回**

**3．中断在跳转到目标地址前，把标志寄存器、**

**下条指令的段地址、偏移地址压入堆栈；**

**用iret返回    iret: interrupt return中断返回**

操作：**int n**

**pushf                     ; 把标志寄存器压入堆栈**

**push  下条指令段地址     ; 把CS压入堆栈**

**push  下条指令偏移地址       ; 把下条指令的偏移地址压入堆**

**TF = 0，IF = 0          ; 清陷阱标志与中断标志**

**jmp  dword  ptr  0000:[n\*4]**

**0000:0000~0000:03FF 这块内存空间称为中断向量表**

**一共占用4\*100h=400h字节**

**因为一共有int 00h~int 0FFh共100h个中断**

**和int指令对应的返回指令为iret，它的操作如下：**

**pop ip**

**pop cs**

**popf**

6．杂类指令：**CLC，STC，CMC，CLD，STD，CLI，STI，NOP，HLT**

（1）**clc**  ; 清进位位（clear carry）

操作：

**CF = 0**

格式：

    clc

（2）**stc**        ; 置进位位（set carry）

操作：

    CF = 1

格式：

    stc

（3）**cmc**        ; 进位位求反（**complement carry**）

操作：**clc  stc  cmc**

    CF = ~ CF   ; 若原来CF=0，则求反后变1，若原来CF=1，则求反后变0

格式：

    cmc

（4）cld         ; 清方向位（clear direction）

操作：**cld  std**

    DF = 0

格式：

    cld

（5）std         ; 置方向位（set direction）

操作：

    DF = 1

格式：

    std

（6）**cli**          ; **禁止中断（clear interrupt）**

此指令用来禁止外部中断如时钟中断、键盘中断等。

操作：**cli  ...  sti**

    IF = 0

格式：

    cli

（7）**sti**        ; **允许中断（set interrupt）**

操作：

    IF = 1

格式：

    sti

（8）**nop**    ; **无操作（no operation）**

**nop = xchg ax, ax  = 机器码90h**

此指令对应的机器码是90h，它占用一个字节的内存空间。执行此指令除了消耗一些时钟周期外，将不会产生任何操作。

（9）**hlt**     ; **停止处理器（halt processor）**

执行此指令将停止处理器的工作，直到系统复位（reset）或发生非屏蔽中断（unmaskable interrupt）或外部中断。