1．字符串操作指令 stosb 及 lodsb

(1) stosb, stosw, stosd

stosb:

stosb 的操作过程如下:

es:[di] = AL

di++; DF=1 时为 di--

rep stosb: 循环 CX 次 stosb

rep stosb 的操作过程:

```
again:
if(cx == 0) goto done;
ES:[DI] = AL
DI++; 当 DF=1 时, 为 DI--
CX--
goto again;
done:
```

例: 把从地址 1000:10A0 开始共 100h 个字节的内存单元全部填 0

```
mov ax, 1000h
mov es, ax; ES=1000h
mov di, 10A0h                    mov di,10A0h
mov cx, 100h     mov cx, 80h     mov cx,40h
cld               cld            cld
xor al, al       xor ax, ax      xor eax, eax
rep stosb        rep stosw       rep stosd
```

(2) lodsb

lodsb 的操作过程:

AL=DS:[SI]

SI++;当 DF=1 时，为 SI--

例: 设 DS:SI→ "##AB#12#XY"
且 ES:DI 指向一个空的数组，CX=11
通过编程过滤#最后使得 ES:DI →"AB12XY"

```
   cld
again:
   lodsb; AL=DS:[SI], SI++
        ; mov al, ds:[si]
        ; inc si
   cmp al, '#'
   je   next
   stosb; ES:[DI]=AL, DI++
        ; mov es:[di], al
        ; inc di
next:
   dec cx
   jnz again
```

2．控制转移指令

(1) jmp 的 3 种类型

```
①jmp short target      ; 短跳
②jmp near ptr target   ; 近跳
③jmp far ptr target    ; 远跳
```

一般情况下，编译器会自动度量跳跃的距离，因此我们在写源程序的时候不需要加上 short、near ptr、far ptr 等类型修饰。即上述三种写法一律可以简化为 jmp target。

(2)短跳指令

①短跳指令的格式

jmp 偏移地址或标号

以下条件跳转指令也都属于短跳：**jc jnc jo jno js jns jz jnz ja jb jae jbe jg jge jl jle jp jnp**

②短跳指令的机器码

地址	机器码	汇编指令
1D3E:0090	...	
1D3E:0100	EB06	jmp 108h

提问：jmp 90h 机器码为？
EB8E
关键要点：90h-102h=FF8E（16位）=8E(8位)

短跳指令的机器码由 2 字节构成：
第 1 个字节=EB
第 2 个字节=
Δ=目标地址-下条指令的偏移地址=108h-102h=06h

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

例：自我移动的代码 http://10.71.45.100/bhh/movcode.asm
例：修改 printf 让它做加法运算https://www.cc98.org/topic/4584504

③ 短跳太远跳不过去的解决办法

```
cmp ax, bx
je equal; jump out of range
jne not_equal
jmp equal; 近跳
not_equal:
...; 假定这里省略指令的机器码总长度超过 7Fh 字节
equal:
...
```

(3)近跳指令

①近跳指令的 3 种格式

```
jmp 偏移地址或标号; 如 jmp 1000h
jmp 16 位寄存器   ; 如 jmp bx
jmp 16 位变量     ; 如 jmp word ptr [addr]
```

②近跳指令的机器码

```
   地址          机器码        汇编指令
1D3E:0100   E9FD1E       jmp  2000h
```

近跳指令的第 1 个字节=E9

第 2 个字节=Δ=目标地址-下条指令的偏移地址

=2000h-103h=1EFDh

```
1D3E:0103   B44C         mov  ah, 4Ch
1D3E:0105   CD21         int  21h
...
1D3E:2000   ...
```

```
byte ptr  ; 1 字节
word ptr  ; 2 字节
dword ptr; 4 字节(32 位整数或 float 类型小数)
fword ptr; 6 字节(4 字节偏移地址+2 字节段地址)
qword ptr; 8 字节(64 位整数或 double 类型小数)
tbyte ptr; 10 字节(long double 类型的 80 位小数)
```

short 用来修饰一个短的标号

near ptr 用来修饰一个近的标号

far ptr 用来修饰一个远的标号

(4)远跳指令

①远跳指令的 2 种格式

```
jmp 段地址:偏移地址
jmp dword ptr 32 位变量
```

②远跳指令的机器码

```
jmp 1234h:5678h; 机器码为 0EAh,78h,56h,34h,12h
```

远跳到某个常数地址时,在源程序中不能直接用 jmp 指令，而

应该改用机器码 0EAh 定义，如：

```
db 0EAh
dw 5678h
dw 1234h
```

上述 3 行定义合在一起表示 jmp 1234h:5678h

例：jmp dword ptr 32 位变量 的用法

```
data segment
addr dw 0000h, 0FFFFh
;或写成 addr dd 0FFFF0000h
data ends
code segment
assume cs:code, ds:data
main:
mov ax, data
mov ds, ax
jmp dword ptr [addr]
;相当于 jmp FFFF:0000
code ends
end main
```

例：演示短跳、近跳、远跳 http://10.71.45.100/bhh/jmp.asm

3．循环指令：LOOP

loop dest 的操作过程：

```
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
```

例：求 1+2+3 的和

```
mov ax, 0
mov cx, 3
next:
add ax, cx; ax +3, +2, +1
loop next; cx=2, 1, 0
```

```
            ; dec cx
            ; jnz next
  done:
```

cx=0时，loop次数最多

```
  mov ax, 0
  mov cx, 0
  jcxz done  这条指令可以防止 cx 为 0 时进入循环
next:
  add ax, cx
  loop next; 循环 10000h 次
  done:
```

4. call, ret 指令

(1) 汇编语言中的三种参数传递方式

① 寄存器传递

```
  f:
     add ax, ax; ax=2*ax
     ret         ; 返回时 ax 就是函数值
  main:
     mov ax, 3; ax 就是 f()的参数
     call f
  next:
     mov ah, 4Ch
     int 21h
```

call的原理:
先将call后面的一行代码的偏移地址压入堆栈，然后jmp,
遇到ret后，pop 堆栈进入ip（32位弹如eip），从而使程序知道跳入哪段代码
从而jmp到后面的代码

② 变量传递

```
  f:
```

```
    mov ax, var
    add ax, ax; ax 就是函数值
    ret
main:
    mov var, 3; var 是一个 16 位的变量，用作参数
    call f
```

在汇编语言中，用 db、dw 等关键词定义的变量均为全局变量。在堆栈中定义的变量才是局部变量。

③ 堆栈传递

```
f:
    push bp
    mov bp, sp
    mov ax, [bp+4]; 从堆栈中取得参数
    add ax, ax
    pop bp
    ret
main:
    mov ax, 3
    push ax; 参数压入到堆栈
    call f
    add sp, 2
```

int printf(char *format,...)　　原型
参数从右向左压入堆栈
为printf函数服务，这样知道第一个参数为bp+4
printf(3,"%d")　×
printf("Hello");

(2) C 语言函数调用 y=f(2,3)求两数之和转化成汇编语言

```
f:
   push bp;(4)
   mov bp, sp
   mov ax, [bp+4]
   add ax, [bp+6]
   pop bp; (5)
   ret; (6)
```

```
main:
   mov ax, 3
   push ax; (1)
   mov ax, 2
   push ax; (2)
   call f; (3)
here:
   add sp, 4;(7)
```

上述程序运行过程中的堆栈布局如下：

```
ss:1FF8 old bp <- bp (4)
ss:1FFA here <- (3)(5)
ss:1FFC 02   <- (2)(6)
ss:1FFE 03   <- (1)
ss:2000 ??   <- (7)
```

参数个数可变的函数例子：**myprintf.c**

https://www.cc98.org/topic/4586741