

1. 换码指令: XLAT (translate) 也称查表指令

在 `xlat` 执行前必须让 `ds:bx` 指向表, `al` 必须赋值为数组的下标; 执行 `xlat` 后, `AL=ds:[bx+AL]`

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

```
char i;
```

```
i = 10;
```

```
i = t[i]; 最后 i='A'
```

设 `ds`=数组 `t` 的段地址

```
mov bx, offset t; BX=表的首地址
```

```
mov al, 10; AL 为下标
```

```
xlat; 结果 AL='A'
```

`xlat` 指令要求 `DS:BX` 指向数组, `AL`=数组下标。

执行指令后, `AL`=数组元素

例子: <http://10.71.45.100/bhh/xlat.asm>

例子: http://10.71.45.100/bhh/xlat_sub.asm

2. 算术指令

(1) 加法指令: ADD, INC, ADC

```
inc: increment
```

```
mov ax, 3
```

```
inc ax; AX=AX+1=4
```

`inc` 指令不影响 `CF` 标志位

inc 不影响 CF 位, add 指令会影响 CF:

```
again:          again:
add ax, cx      add ax, cx
jc done         inc cx
add cx,1        jnc again
jmp again       done:
done:
```

adc: add with carry 带进位加

计算 12345678h + 5678FFFFh

```
mov dx, 1234h
```

```
mov ax, 5678h
```

```
add ax, 0FFFFh; CF=1
```

```
adc dx, 5678h; DX=DX+5678h+CF
```

把 x 和 y 相加 (x、y 均为由 100 字节构成且用小端表示的大数), 结果保存到 z 中:

```
x db 100 dup(88h)
```

```
y db 100 dup(99h)
```

```
z db 101 dup(0)
```

设 ds 已经赋值为上述数组的段地址

```
mov cx, 100
```

```
mov si, offset x
```

```
mov di, offset y
```

```
mov bx, offset z
```

```

clc
next:
mov al, [si]
adc al, [di]
mov [bx], al
inc si
inc di
inc bx
dec cx
jnz next
adc z[100], 0; 或 adc byte ptr [bx], 0

```

(2) 减法指令: SUB, SBB, DEC, NEG, CMP

dec: decrement 自减, dec 指令不影响 CF

```

mov ax, 3
dec ax; AX=AX-1=2

```

neg:negate 求相反数, 会影响 CF, ZF, SF 等标志位。

```

mov ax, 1
neg ax; AX=-1=0FFFFh, 相当于做减法 0-ax

```

```

mov ax, 0FFFEh
neg ax; AX=2, CF=1, SF=0, ZF=0

```

$\text{neg ax} \equiv (\text{not ax}) + 1$

$$-x \equiv \sim x + 1$$

sbb: subtract with borrow 带借位减

例如求 56781234h-1111FFFFh 的差

```
mov ax, 1234h
```

```
sub ax, 0FFFFh; CF=1
```

```
mov dx, 5678h
```

```
sbb dx, 1111h; DX=5678h-1111h-CF
```

cmp: cmp 与 sub 的区别是抛弃两数之差，仅保留标志位状态

```
mov ax, 3
```

```
mov bx, 3
```

```
cmp ax, bx; 内部是做了减法 ax-bx, 但
```

；是抛弃两数之差，只影响标志位。

```
je they_are_equal; 当 ZF=1 时则跳
```

```
jz they_are_equal; 当 ZF=1 时则跳
```

因此 je≡jz

① ja, jb, jae, jbe 都是非符号数比较相关的跳转指令

jb: CF=1, 故 jb≡jc

ja: CF=0 且 ZF=0

② jg, jl, jge, jle 是符号数比较相关的跳转指令

jg: SF==OF

```

jl: SF!=OF
mov ax, 3
mov bx, 2
cmp ax, bx; AX-BX=1, SF=0, OF=0 → AX > BX
mov ah, 7Fh
mov bh, 80h
cmp ah, bh; AH-BH=0FFh, SF=1, OF=1 → AX>BX


---


mov ax, 2
mov bx, 3
cmp ax, bx; AX-BX=FFFFh, SF=1, OF=0 → AX<BX
mov ah, 80h
mov bh, 7Fh
cmp ah, bh; AH-BH=1, SF=0, OF=1 → AX<BX

```

(3) 乘法指令: MUL(非符号数乘法), IMUL(符号数乘法)

mul 是非符号数的乘法指令。

imul 是符号数的乘法指令。

386 以上 CPU 对 imul 功能进行了扩充:

① `imul eax, ebx, 1234h`
 寄存器 寄存器或 只能是常数
 变量

含义: `eax = ebx * 1234h`

② `imul eax, ebx`; 含义: `EAX=EAX*EBX`
 寄存器 寄存器或变量

`imul eax, dword ptr [esi]`

```
imul eax, dword ptr [ebx+2], 1234h
```

含义: $\text{eax} = \text{dword ptr [ebx+2]} * 1234h$

(4) 除法指令: DIV, IDIV

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

(5) 小数运算

fadd fsub fmul fdiv 小数的+-* / 运算指令

由浮点处理器负责执行, 用法请参考主页 intel 指令集

小数变量的定义:

pi dd 3.14; 32 位小数, 相当于 float

r dq 3.14159; 64 位小数, 相当于 double

; q:quadruple 4 倍的

s dt 3.14159265; 80 位小数, 相当于 long double

在 C 语言中要输出 long double 的值需要使用 "%Lf" 格式

CPU 内部一共有 8 个小数寄存器, 分别叫做

st(0)、st(1)、...、st(7)

其中 st(0) 简称 st

这 8 个寄存器的宽度均达到 80 位, 相当于 C 语言中的

long double 类型。

VC 里面的 long double 类型已经退化成 double 类型。

例子: <http://10.71.45.100/bhh/float.asm>