2021/01/04_16位汇编_第7课_算术运算类指令以及控制转移指令

笔记本: 16位汇编
创建时间: 2021/1/4 星期一 10:46
作者: ileemi
标签: 控制转移指令, 算术运算类指令

算术运算类指令

> 四则运算是计算机经常进行的一种操作。算术运算指令实现二进制（和十进制）数据的四则运算。

请注意算术运算类指令对标志的影响：

- 掌握：ADD/ADC/INC、SUB/SBB/DEC/ NEG/CMP
- 熟悉：MUL/IMUL、DIV/IDIV
- 理解：CBW/CWD、DAA/DAS、 AAA/ AAS/AAM/AAD

加法指令 ADD

影响的标志寄存器有：AF、CF、OF、PF、SF、ZF

> ADD指令将源与目的操作数相加，结果送到目的操作数。ADD指令按状态标志的定义相应设置。

带进位加法指令 ADC

> ADC指令将源与目的操作数相加，再加上进位CF标志，结果送到目的操作数（ADC指令按状态标志的定义相应设置）。ADC指令主要与ADD配合，实现多精度加法运算。

多用于两个32位数值相加：

> 示例1：
> mov ax, 0FFFFH
> sub bx, bx
> mov cx, 1
> sub dx, dx
> add ax, cx
> adc bx, dx

> 示例2：
> mov ax, 4652h ;ax=4652h
> add ax, 0f0f0h ;ax=3742h，CF=1
> mov dx, 0234h ;dx=0234h
> adc dx, 0f0f0h ;dx=f325h，CF=0
> dx ax = 0234H 4652H + F0F0H F0F0H = F325H 3742H

减法指令 SUB

> sub指令将目的操作数减去源操作数，结果送到目的操作数。sub指令按照定义相应设置状态标志（sub指令按照定义相应设置状态标志）。

带借位减法指令 SBB

sbb指令将目的操作数减去源操作数，再减去借位CF（进位），再将结果送到目的操作数种（sbb指令按照定义相应设置状态标志）。sbb指令主要与sub配合，实现多精度减法运算。

> SBB reg, imm/reg/mem ； reg <--（reg - (imm/reg/mem) - CF）
> SBB mem ,imm/reg ； mem <-- （mem - imm/reg - CF）

多用于32位数值的减法：

```
mov ax, 0FFFFH
sub bx, bx
mov cx, 1
sub dx, dx
add ax, cx
adc bx, dx ;带借位加法
sub ax, cx
sbb bx, dx ;带借位减法
```

减量指令 DEC

> dec指令对操作数减1（减量），dec指令不影响进位CF标志，按定义设置其他状态标志。

dec bx ;bx = bx - 1

增量指令 INC

> inc指令对操作数加1（加量），inc指令不影响进位CF标志，按定义设置其他状态标志（INC指令和DEC指令都是单操作数指令，主要用于对计数器和地址指针的调整）。

inc bx ;bx = bx +1

求补指令 NEG

> neg指令对操作数执行求补运算：用零减去操作数，然后结果返回到操作数。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 reg/mem ;reg/mem <--（0 - reg/mem）

```
mov ax, 1
neg ax ;0-ax
```

比较指令 CMP

> cmp指令执行的功能与SUB指令，但结果不回送目的操作数。cmp指令将目的操作数减去源操作数，按照定义相应设置状态标志（只做判断，影响标志位，但不影响目的操作数）。

执行比较指令之后，可通过标志位（CF、SF），判断两个数值的大小、是否相等，以及有无符号。

CPU 中数值的运算，数值都是以无符号（正）进行运算的。一个有符号数的运算也可以用无符号运算来计算（结果用补码来表示）。

语法：

> cmp reg,imm/reg/mem ;reg - imm/reg/mem
> cmp mem,imm/reg ;mem - imm/reg

有符号无符号数值相加

0xFFFF（正）+ 1 = CF = 1 0000（无符号，1：进位，CF标志用来判断）
0xFFFF（负）+ 1 = CF = 1 0000（有符号，1：进位，CF标志用来判断）

```
mov ax, 0FFFFH
mov bx, 0FFFFH
mov cx, 1
sub dx, dx
add ax, cx
adc bx, dx ;带借位加法
sub ax, cx
sbb bx, dx ;带借位减法
```

乘法指令 MUL、IMUL

> 乘法指令区分有符号（imul）、无符号（mul），两者在电路上的设计是有区别的。乘法指令的源操作数显式给出，隐含使用另一个操作数ax和dx（**乘法指令利用OF和CF判断乘积的高一半是否具有有效数值**）。

- 字节量相乘：al与r8/m8相乘，得到16位的结果，存入ax中。
- 字量相乘：ax与r16/m16相乘，得到32位的结果，其高字存入dx，低字存入ax。

语法：

> mul r8/m8 ;无符号字节乘法（ax <-- al * r8/m8）
> mul r16/m16 ;无符号字乘法（dx.ax <-- ax * r16/m16）
> imul r8/m8 ;有符号字节乘法（ax <-- al * r8/m8）
> imul r16/m16 ;有符号字乘法（dx.ax <-- ax * r16/m16）

imul：有符号字节乘法（ax（FFFF） * bx（0003） = ax（FFFD））

乘法指令对标志的影响

乘法指令如下影响OF和CF标志：

- mul指令：若乘积的高一半（ah或dx）为0，则OF=CF=0；否则OF=CF=1
- imul指令：若乘积的高一半是低一半的符号扩展，则OF=CF=0；否则均为1
- 乘法指令对其他状态标志没有定义

对标志没有定义：指令执行后这些标志是任意的、不可预测（就是谁也不知道是0还是1）。

对标志没有影响：指令执行不改变标志状态。

使用示例：

```
mov al, 0b4h   ;al = b4h = 180
mov bl, 11h    ;bl = 11h = 17
mul bl         ;ax = Obf4h = 3060
               ;OF = CF = 1，ax高8位不为0

mov al, 0b4h   ;al = b4h = -76
mov bl, 11h    ;bl = 11h = 17
imul bl        ;ax = faf4h =-1292
               ;OF = CF = 1，ax高8位含有效数字
```

除法指令 DIV、IDIV

> 除法指令分无符号（div）和有符号除法指令（idiv）。除法指令的除数显式给出，隐含使用另一个操作数ax和dx作为被除数（除法指令对标志没有定义，除法指令会产生结果溢出）。CPU 不支持小数运算（可用正数代替）。

- 字节量除法：ax除以r8/m8，8位商存入al，8位余数存入ah
- 字量除法：dx.ax除以r16/m16，16位商存入ax，16位余数存入dx

16bit / 8bit = 8bit（运算结果（商）不能大于8位）
32bit / 16bit = 16bit

```
div r8/m8     ;无符号字节除法   al <-- ax / (r8或m8)的商, Ah <-- ax / (r8或m8)的余数
div r16/m16   ;无符号字除法     ax <--dx.ax / (r16或m16)的商, <-- dx.ax / (r16或m16)的余数

idiv r8/m8    ;有符号字节除法   al <-- ax / (r8或m8)的商, Ah <-- ax / (r8或m8)的余数
idiv r16/m16  ;有符号字除法     ax <-- dx.ax / (r16或m16)的商, dx <-- dx.ax / (r16或m16)的余数
```

除法错中断

当被除数远大于除数时，所得的商就有可能超出它所能表达的范围。如果存放商的寄存器al/ax不能表达，便产生溢出，8086CPU中就产生编号为0的内部中断（除法错中断）。

- 对div指令，除数为0，或者在字节除时商超过8位，或者在字除时商超过16位，则发生除法溢出;
- 对idiv指令，除数为0，或者在字节除时商不在-128～127范围内，或者在字除时商不在-32768～32767范围内，则发生除法溢出。

符号扩展指令

符号扩展是指用一个操作数的符号位（即最高位）形成另一个操作数，后一个操作数的各位是全0（正数）或全1（负数）。符号扩展不改变数据大小（符号扩展指令常用于获得倍长的数据）。

- 对于数据64H（表示数据100，0110 0100B），其最高位为0，符号扩展后高8位都是0，成为0064H（仍表示数据100）；
- 对于数据FF00H（表示有符号数 - 256，1111 1111 0000 0000B），其最高位为1，符号扩展后高16位都是1，成为FFFFFF00H（仍表示有符号数 - 256）。

> 不影响标志位。
> cbw：byte --> word（al的符号扩展至ah，如al的最高有效位是0，则ah = 00，al的最高有效位为1，则ah = FFH，al不变）
> cwd：word --> dword（ax的符号扩展至dx，如ax的最高有效位是0，则dx = 00，ax的最高有效位为1，则dx = FFFFH，ax不变）

代码示例：

```
mov al,80h   ;al = 80h
cbw          ;ax = ff80h
add al,255   ;al = 7fh
cbw          ;ax = 007fh
```

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。分成压缩bcd码和非压缩bcd码。

BCD 压缩码

> 二进制编码的十进制数：用4位二进制编码表示一位十进制数，一个字节可以表示两个十进制位，即00～99。

8086支持压缩bcd码和非压缩bcd码的调整运算：

真值	8	64
二进制编码	08H	40H
压缩bcd码	08H	64H
非压缩bcd码	08H	0604H

mov al, 19h ;bcd 压缩码 19 + 1 = 20
inc al ;1a

> 16进制的结果（1a）调整为10进制的结果（20）
> add al, 6

daa ;调整加法，按4位进行调整
das ;调整减法
dam ;调整乘法
dav ;调整除法

非 BCD 压缩码

> 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。

mov ax, 0109h
inc ax ; 1a

aaa ;按8位进行调整，调整加法
aas ;调整减法
aam ;调整乘法
aav ;调整除法

控制转移指令

> 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最常用指令（控制转移类指令通过改变IP 和 CS 数值，实现程序执行顺序的改变）。

无条件转移指令 JMP

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处，从目标地址处开始执行那里的指令。

示例：
jmp LOOP ;操作数LOOP是要转移到的目标地址（目的地址、转移地址）

> JMP 指令分为4种形式：

1. 段内转移、直接寻址
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

目标地址的寻址方式

直接寻址方式：转移地址像立即数一样，直接在指令的机器代码中，就是直接寻址方式（使用标号表达）。

> jmp IF_ELSE ;IF_ELSE -- 标号

间接寻址方式：转移地址在寄存器或主存单元中，就是通过寄存器或存储器的间接寻址方式（用寄存器或存储器操作数表达）。

> jmp word ptr [1000]
> jmp ax

目标地址的范围：段内

- 段内转移（近转移 near）：
  近转移：两个字节表示偏移（存储偏移量表示要跳转到的位置），范围在（-32KB~32KB）不需要更改CS段地址，只要改变IP偏移地址。
  jmp near ptr LOOP1（操作码：E9）
- 段内转移（短转移 short）：转移范围可以用一个字节表达，在段内（-128~127）的范围，可以使用短转移。
  jmp short LOOP1（操作码：EB）

> 往上跳转使用**有符号**（近转移：两个字节，短转移：一个字节）数值表示编译量。
> 示例：EBF7 JMP 0007

jmp LOOP1：伪指令（编译器自动选择一种进行使用），仅 VS 编译器支持。

FF（-1）- F6 = -1 - 9 = -10

目标地址的范围：段间

> 代码段到代码段（跨段）

段间转移（远转移 far）：从当前代码段跳转到另一个代码段，可以在1MB范围，需要更改CS段地址和IP偏移地址。目标地址必须用一个32位数表达，叫做32位远指针，它就是逻辑地址。

前16位：偏移地址
后16位：段地址

jmp far ptr LOOP1

> 实际编程时，汇编程序会根据目标地址的距离，自动处理成短转移、近转移或远转移。程序员可用操作符short、near ptr 或far ptr 强制转移。

条件转移指令 jcc

jcc LOOP
指定的条件cc如果成立，程序转移到由标号LOOP指定的目标地址去执行指令；条件不成立，则程序将顺序执行下一条指令（操作数LOOP是采用短转移，称为相对寻址方式）。

> 指定标志位进行跳转

jz LOOP1：ZF == 1时，跳转
jc LOOP1：CF == 1时，跳转
jnz LOOP1：ZF == 0时，跳转

dec 指令会影响 ZF 标志位。使用 jcc 可以模拟高级语言的 if（jnz） else等。

条件转移指令中的条件cc

> 等于：e
> 有符号：大于（G）、小于（L）
> 无符号：大于（A）、小于（B）

有符号：jge、jle、jl、jg
无符号：jae、jbe、ja、jb

jl == jnge（操作码一致，等价）