带符号的加减乘除运算指令

判断数的溢出：有溢出的时候OF会被设置为1

单符号位：最高两位同时有进位或者同时没有进位，则没有溢出

双符号位：最高两位不相同则有溢出。

位的扩展指令

把8位扩展成16位：CBW（convert signed byte to signed word）

隐含操作数在AL中，返回结果在AX中

把16位扩展成32位：CWD（convert signed word to signed double word）

隐含操作数在AX中，返回结果在DX（高16位）和AX（低16位）中

如果原来的最高位是0，则扩充后高位补零，如果原来的最高位是1，则是负数，扩展后高位全部补1。

带符号的除法指令

IDIV 除数

被除数隐含在DX（高16位）和AX（低16位）中

基本与DIV指令一样，除数不能是立即数

1. 字节除以字节

字节存储在AL中，AH不能简单清零，要根据AL的最高位来判断符号

所以直接用CBW就好了

商存储在AL中，余数存储在AH中

1. 字除以字节

被除数存储在AX中

商存储在AL中，余数存储在AH中

1. 字除以字

DX存储为0，AX存储被除数，直接用CWD

商在AX中，余数在DX中

1. 双字除以字

DX存储被除数高16位，AX存储被除数低16位

商在AX中，余数在DX中

运算示例：-30 / 8 = -3 …… -6，得到的余数和被除数的符号位是一样的

带符号的乘法运算

IMUL 乘数

被乘数隐含在AL或者AX中

结果在DX（高16位）和AX（低16位）中

算术移位指令

SAR(shift arithmetic right)

SAR 操作数, 移位次数

最低位移到CF中，最高位不是直接补零，而是根据原来的最高位进行补充，如果原来最高位是1，则补1，如果原来最高位是0，则补零

SAL 操作数, 移位次数

实际上跟SHL是一样的。最高位移出去，移到CF中，最低位空出来，补零。

有符号数的比较

无符号数比较的时候，只需要看CF和ZF就可以判断了，而有符号数不止要看CF和ZF，还要看OF才能判断大小。

都是用CMP

无符号数的比较：

JA 高于（Jump Above）

JB 低于（Jump Below）

有符号数的比较：

JG 大于（Jump Greater）

JGE 大于等于

JL 小于（Jump Less）

JLE 小于等于

JE 等于

字符串操作指令

要关注源操作数所在区域和目标操作数所在区域

源操作数默认放在DS数据段中，偏移值放在SI(source)中，目标操作数默认放在ES附加段中，偏移值放在DI(destination)中。附加段和数据段可以共用同一个物理段（同一个段既当做数据段，也当做附加段来使用），也可以不共用同一个段。

源操作数： DS : SI

目标操作数： ES : DI

操作前要进行段地址的初始化。

预先要对CX进行赋值，表示要传递多少个字节或者字。

MOVSB 按照字节传递，将DS : SI的字符串传到ES : DI中

MOVSW 按照字传递（小端规则），将DS : SI的字符串传到ES : DI中

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | Mnemonic | Destination | Source | Prefix |
| Move string byte | MOVSB | ES:DI | DS:SI | REP |
| Move string word | MOVSW | ES:DI | DS:SI | REP |
| Store string byte | STOSB | ES:DI | AL | REP |
| Store string word | STOSW | ES:DI | AX | REP |
| Load string byte | LODSB | AL | DS:SI | None |
| Load string word | LODSW | AX | DS:SI | None |
| Compare string byte | CMPSB | ES:DI | DS:SI | REPZ/REPNZ |
| Compare string word | CMPSW | ES:DI | DS:SI | REPZ/REPNZ |
| Scan string byte | SCASB | ES:DI | AL | REPZ/REPNZ |
| Scan string word | SCASW | ES:DI | AX | REPZ/REPNZ |

DF标志位，运算方向

CLD：DF标志位清零，地址递增的方向运算

STD：DF标志位置1，地址递减的方向运算

STOSB 按照字节来进行存储。

如果使用了SI或者DI，则SI和DI会自动增加，这条指令是使用了DI。

第七章 模块化编程

地址目的传送指令

LEA 目标操作数, 源操作数

把源操作数传送到目标操作数中。

LEA SI, TABLE 相当于 MOV SI, OFFSET TABLE

这条指令不会改变任何标志位。

LDS 目标操作数, 源操作数

LDS DI, [1200H]

则执行这条指令后DI为1200H内存里面存放的2个单元按照小端规则组合后的值赋值到DI中。如果1200H里面存的是46,1201H里面存的是10，则运行后DI为1046H

LES 目标操作数, 源操作数

LDS DI, [1200H]

如果内存从1200H开始存放的依次是：46 10 38 82

则指令运行完后，DI = 1046H, ES = 8238H

DI与前面LDS是一样的，不同的是在ES在这里被赋值了。

EXTRN位指令

告诉汇编器，当前模块中没有定义的变量是在其他模块中定义的。

EXTRN name : type

EXTRN name1 : type, name2 : type

PUBLIC位指令

当前模块中的变量可能会被其他模块引用，则要用PUBLIC位指令指示出来。

PUBLIC name

BPULIC name1, name2

END位指令

只有一个模块的END后可以有入口，因为程序只能有一个入口。其他模块END后面是没有入口标签的。

把OBJ文件合并到一起：

LINK 1.OBJ+2.OBJ

SEGMENT

标签 SEGMENT [定位] [连接方式] [类名]

类名用单引号括起来，只有三种：’CODE’, ‘DATA’, ‘STACK’

定位：指明段中信息的存储是以何种类型进行分配的

每种类型对应的地址可选项，x代表可以是1或0

PAGE xxxx xxxx xxxx 0000 0000B 存储的地址一定是256的倍数

PARA xxxx xxxx xxxx xxxx 0000B 存储的地址一定是十六的倍数，默认是它

WORD xxxx xxxx xxxx xxxx xxx0B 存储的地址一定是偶数

BYTE xxxx xxxx xxxx xxxx xxxxB 可以存储到任何一个地址

连接方式：一个可执行文件可以有一个或者多个代码段或者数据段

NONE： 默认是它，不连接，与其他段之间没有联系，每个段都有自己的地址，是独立段

PUBLIC：相同类型且命名相同的段合并成一个段：如相同名字的数据段

STACK：所有的堆栈段合并成一个堆栈段

COMMON：相同类型且命名相同的段合并成一个段，拥有相同的段地址

与PUBLIC的区别在内存分配：

如果有3个数据段，大小分别是10K，20K，30K，则PUBLIC会分配60K的数据段，而COMON只分配30K的数据段，任何一个数据段进去则另外两个会出来。

MEMORY：段在载入内存的时候会尽量被放到高内存地址中

子程序调用参数的传递

1. 通过寄存器

如DOS中断、BIOS中断，把参数传递到AH、AL中

1. 通过内存（数据段、堆栈段）

在两个不同的模块之间可以通过公用数据段来传递参数。

1. 通过堆栈段

在调用子程序之前，把要传递的参数先压入栈中，然后调用子程序。

要注意子程序调用会保留现场，CS(FAR调用才有)、IP、BP会被压入栈中，所以要取得前面压入的参数，就要用：

MOV BP, SP ;获取栈顶指针

MOV AX, [BP]+6 ;获取第三个参数

MOV CX, [BP]+8 ;获取第二个参数

MOV DX, [BP]+10 ;获取第一个参数

子程序返回的时候，RET后面要加个6，让它的栈顶指针在子程序调用结束后返回压栈之前的状态。压入3个参数是6个字节，所以是RET 6。