MOV（传送）

指令写法：MOV  target，source

功能描述：将源操作数source的值复制到target中去，source值不变

注意事项：1）target不能是CS（代码段寄存器），我的理解是代码段不可写，只可读，所以相应这地方也不能对CS执行复制操作。2）target和source不能同时为内存数、段寄存器（CS\DS\ES\SS\FS\GS）3)不能将立即数传送给段寄存器4）target和source必须类型匹配，比如，要么都是字节，要么都是字或者都是双字等。4）由于立即数没有明确的类型，所以将立即数传送到target时，系统会自动将立即数零扩展到与target数的位数相同，再进行传送。有时，需要用BYTE PTR 、WORD PTR、 DWORD PTR明确指出立即数的位数

写法示例：MOV  dl,01H;MOV  eax,[bp]; eax =ss:[bp] 双字传送。

2、     XCHG(交换)

指令写法：XCHG object1，object2

功能描述：交换object1与object2的值

注意事项：1）不能直接交换两个内存数的值 2）类型必须匹配3）两个操作数任何一个都不能是段寄存器【看来段寄存器的写入的限制非常的严格，MOV指令也不能对段寄存器进行写入】，4）必须是通用寄存器（ax、bx、cx、dx、si、di）或内存数

写法示例：XCHG  ax，[bx][si]; XCHG ax,bx;

3、     LEA(装入有效地址)

指令写法：LEZ reg16，mem

功能描述：将有效地址MEM的值装入到16位的通用寄存器中。

写法示例：假定bx=5678H，EAX=1,EDX=2

                              Lea si,2[bx]                    ;si=567AH

                              Lea di,2[eax][edx]       ;di=5

注意，这里装入的是有效地址，并不是实际的内存中的数值，如果要想取内存中该地址对应的数值，还需要加上段地址才行，而段地址有可能保存在DS中，也有可能保存在SS或者CS中哦:>不知道我的理解可正确。。。。

4、     LDS\LES\LGS\LSS（注意，与LEA不同的是，这里是装入的值，而不是有效地址）

这几个指令，名称不同，作用差不多。

写法：LDS reg16，mem32

功能描述：reg16等于mem32的低字，而DS对应于mem32的高字（当为LES时，这里就是ES对应于mem32的高字）

用来给一个段寄存器和一个16位通用寄存器同时复制。

注意事项：第一个操作数必须是16位通用寄存器

在接着往下说之前，先熟悉下堆栈的概念。堆栈，位于内存的堆栈段中，是内存的一部分，具有“先进后出”的特点，堆栈只有一个入口，即当前栈顶，当堆栈为空时，栈顶和栈底指向同一内存地址，在WINDOWS中，可以把堆栈理解成一个倒着的啤酒瓶，上面的地址大，下面的地址小，当从瓶口往啤酒瓶塞啤酒时（进栈），栈顶就会往瓶口下移动，也就是往低地址方向移动，同理，出栈时，正好相反，把啤酒给倒出来，栈顶向高地址方向移动。这就是所谓的堆栈 ，哼哼，很Easy吧。

在汇编语言中，堆栈操作的最小单位是字，也就是说，只能以字或双字为单位，同时，SS：SP指向栈顶（SS为堆栈段寄存器，SP为堆栈指针，二者一相加，就构成了堆栈栈顶的内存地址）。

5、     PUSH（进栈）

写法：PUSH reg16（32）/seg/mem16（32）/imm

功能描述：将通用寄存器/段寄存器/内存数/立即数的值压入栈中，即：

SP=SP-2 SS:[SP]=16位数值（当将32位数值压入栈中时，SP=SP-4，SS:[SP]=32为数值）

6、     POP（出栈）

写法：POP reg16（32）/seg/mem16（32）【不能出栈到CS中】

功能描述：将堆栈口的16（32）位数据推出到通用寄存器/段寄存器/内存中，即：

寄存器/段寄存器/内存= SS:[SP]   SP=SP+2（当将32位数值出栈时，SP=SP+4）（注意，不能出栈给立即数哦，常量不可变嘛）

7、     PUSHA、PUSHAD、POPA、POPAD

作用：将所有16/32位通用寄存器进栈/出栈

如：PUSHA ;将AX、CX、DX、BX、原SP、BP、SI、DI依次进栈。POPA出栈顺序正好相反，但要注意的是，弹出到SP的值被丢弃，SP通过增加16位来恢复（当然嘛，不然栈顶地址就被修改了，就会出息不对齐的情况，就有可能乱套了）

POPAD PUSHAD一样，只不过是32位的罢了。

8、     PUSHF、PUSHFD、POPF、POPFD

功能描述：标志寄存器FLAGS（EFLAGS）进栈或出栈

如：PUSHF ；FLAGS进栈  POPF； 栈顶字出栈到FLAGS

总结下，POP 和PUSH通常可以用来交换两个寄存器的值，也可以用来保护寄存器的值，如下：

交换ax与cx的值：push ax；push cx；pop ax； pop cx；

保护寄存器：push ax；push cx；….中间有很多执行的代码…pop cx;pop ax;

9、LAHF\SAHF（标志寄存器传送指令）

写法：lahf；

作用：AH=FLAGS的低8位

写法：sahf；

作用：FLAGS的低8位=AH

10、符号扩展和零扩展指令

CBW；AL符号扩展为AX

CWD；AX符号扩展为32位数DX:AX

CWDE;AX符号扩展为EAX；

CDQ：EAX符号扩展为64位数EDX:EAX

MOVSX（符号扩展指令的一般形式）

写法：MOVSX reg16\32，reg8\reg16\mem8\mem16

作用：用来将8位符号扩展到16位，或者16位符号扩展到32位

MOVZX（零扩展指令）

写法：MOVZX reg16\32，reg8\reg16\mem8\mem16

零扩展，就是高位补0进行扩展。通常用在将数据复制到一个不同的寄存器中，如AL零扩展为EBX。相同寄存器的零扩展，可以使用MOV 高位， 0来实现。

11、BSWAP（字节交换）

写法：bswap reg32

作用：将reg32的第0与第3个字节，第1与第2个字节进行交换。

示例：设EAX=12345678h

执行bswap eax；后，eax=78563412H

12、XLAT（换码）

写法：XLAT；

作用：AL=DS:[bx+AL]

将DS:BX所指内存中的由AL指定位移处的一个字节赋值给AL。（貌似这是一个方便偷懒的指令哦。。），原来它的主要用途是查表。注意可以给它提供操作数，用来指定使用哪个段地址，如：

XLAT ES：table；使用ES来作为段地址，table不起作用。

XLAT table ；使用table所在段对应的段寄存器作为段地址。

------------------------------------------------数据传送指令结束------------------------------------------------------

----------------------------------算术指令开始-----------------------------------------------

13、ADD（加法）

写法：ADD reg/mem reg/mem/imm

作用：将后面的操作数加到前面的操作数中

注意：两个操作数必须类型匹配，并且不能同时是内存操作数

ADC （带进位加法）

写法：ADC reg/mem, reg/mem/imm ；

作用：dest=dest+src+cf

当CF=0时 ADD与ADC的作用是相同的。

示例：实现64位数EDX:EAX与ECX:EBX的加法：

Add EAX,EBX；

ADC EDX,ECX;

14、INC（自加一）

写法：INC reg/mem；

作用：dest=dest+1；

15、XADD（交换加）

写法：XADD reg/mem， reg

作用：先将两个数交换，然将二者之和送给第一个数

16、SUB（减法）

写法：SUB reg/mem， reg/mem/imm；

作用：dest=dest-src；

SBB（带借位减法）

写法：SBB reg/mem， reg/mem/imm

作用：dest=dest-src-cf；

注意：两个操作数必须类型匹配，且不能同时是内存数

17、DEC（自减1）

写法：DEC reg/mem；

作用：dest=dest-1；

18、CMP（比较）

写法：CMP reg/mem， reg/mem/imm

作用：dest-src

注意：这里并不将结果存入dest中，而仅仅是执行相减的运算，达到依据运算结果去影响EFLAG标志位的效果

19、NEG（求补）

写法：NEG reg/mem

作用：求补就是求相反数，即：dest=0-dest；

20、CMPXCHG(比较交换)

写法：CMPXCHG reg/mem， reg；

作用：AL/AX/EAX-oprd1，如果等于0，则oprd1=oprd2，否则，AL/AX/EAX=oprd1；

即：比较AL/AX/EAX与第一个操作数，如果相等，则置ZF=1,并复制第二个操作数给第一个操作数；否则，置ZF=0，并复制第一个操作数给AL/AX/EAX。

说明：CMPXCHG主要为实现原子操作提供支持

CMPXCHG8B（8字节比较交换指令）

写法：CMPXCHG8B MEM64;

功能：将EDX:EAX中的64位数与内存的64位数进行比较，如果相等，则置ZF=1，并存储ECX:EBX到mem64指定的内存地址；否则，置ZF=0，并设置EDX:EAX为mem64的8字节内容

21、MUL（无符号乘法）

写法：MUL reg/mem；

作用：当操作数为8位时，AX=AL\*src；

当操作数为16位时，DX:AX=AX\*src；

当操作数为32位时，EDX:EAX=EAX\*src；

22、IMUL(带符号位乘法)

写法：IMUL reg/mem；（作用同上）

IMUL reg16，reg16/mem16，imm16；

IMUL reg32，reg32/mem32，imm32；

IMUL reg16，imm16/reg16/imm16；

IMUL reg32，reg32/mem32/imm32；

注意：没有两个操作数均为8位的多操作数乘法。

对于同一个二进制数，采用MUL和IMUL执行的结果可能不同，设AL=0FF，BL=1，分别执行下面的指令，会得到不同的结果：

Mul bl； AX=0FFH(255);

Imul bl； AX=0FFFFH（-1）（高一半为低一半的扩展）

23、DIV（无符号除法 ）/IDIV(带符号数除法)

写法：DIV reg/mem；/IDIC reg/mem

作用：如果操作数是8位，AX%SRC，结果商在AL、余数在AH中；

如果操作数是16位，DX:AX%SRC，结果商在AX,余数在DX中；

如果操作数是32位，EDX:EAX%SRC，结果商在EAX,余数在EDX中；

注意：不能直接实现8位数除8位数、16位数除16位数、32除32，若需要这样，则必须先把除数符号扩展或零扩展到16、32、64位，然后用除法指令。

对于IDIV，余数和被除数符号相同，如：-5 IDIV 2 = 商 -2，余数：-1；

在下列情况下，会使CPU产生中断：一：除数为0 ；二：由于商太大，导致EAX\AX或AL不能容纳，从而产生了溢出。

-----------------BCD码调整指令（十进制调整指令）待补充------------------------------------------------

24、关于BCD码：BCD码就是一种十进制数的二进制编码表示，分为压缩BCD码和非压缩BCD码，压缩BCD码用4个二进制位表示一个十进制位，即用0000B~1001B表示十进制0~9，如0110 0100 0010 1001B表示6429

用8位二进制来表示一个十进制叫非压缩BCD码，其中，低四位与压缩BCD码相同，高四位无意义。

压缩BCD码调整指令包括DAA(加法的压缩BCD码调整)和DAS（减法的压缩BCD码调整）

写法：

DAA;

作用：调整AL中的和为压缩BCD码。

功能：使用DAA指令时，通常先执行ADD/ADC指令，将两个压缩BCD码相加，结果存放在AL中，然后使用该指令将AL调整为压缩BCD码格式。

DAA的调整算法：

IF(AL低4位>9 或 AF=1)

THEN

AL=AL+6;

AF=1;

ENDIF

IF( AL高4位>9或CF=1)

THEN

AL=AL+60H;

CF=1;

ENDIF

说明：CF反映压缩BCD码相加的进位。

DAS;

作用：调整AL中的差为压缩BCD码。

功能：使用DAS指令时，通常先执行SUB/SBB指令，将两个压缩BCD码相减，结果存放在AL中，然后使用该指令将AL调整为压缩BCD码格式。

DAS的调整算法：

IF(AL低4位>9 或 AF=1)

THEN

AL=AL-6;

AF=1;

ENDIF

IF( AL高4位>9或CF=1)

THEN

AL=AL-60H;

CF=1;

ENDIF

说明：CF反映压缩BCD码相减的借位。

特别注意，如果使用DAA或DAS指令，则参加加法或减法运算的操作数应该是压缩BCD码，如果将任意两个二进制数相加或相减，然后调整，则得不到正确的结果。

关键是调整的规则，其中AF标志位就是专门为BCD码调整设计的，当低四位有向高四位进位或借位时，值为1。而CF就是最高位有进位或者借位时，为1.

非压缩BCD码调整指令，包括AAA,AAS,AAM,AAD。

写法：AAA ;

作用：调整AL中的和为非压缩BCD码；调整后，AL高4位等于0，AH=AH+产生的CF

功能：使用AAA指令时，通常先执行ADD/ADC指令，以AL为目的操作数，将两个非压缩BCD码（与高位无关）相加，然后使用AAA将AL调整为非压缩BCD码格式，且高4位等于0，同时，将调整产生的进位加到AH中。

AAA调整算法：

IF(AL低4位>9 或者 AF=1)

THEN

AL=AL+6;

AH=AH+1;

AF=1;

CF=1;

ELSE

AF=0;CF=0;

ENDIF

AL=AL AND OFH;;AL高4位清0

写法：AAS ;

作用：调整AL中的差为非压缩BCD码；调整后，AL高4位等于0，AH=AH-产生的CF

功能：使用AAS指令时，通常先执行SUB/SBB指令，以AL为目的操作数，将两个非压缩BCD码（与高位无关）相减，然后使用AAS将AL调整为非压缩BCD码格式，且高4位等于0，同时，将调整产生的借位从AH中减去。

AAA调整算法：

IF(AL低4位>9 或者 AF=1)

THEN

AL=AL-6；

AH=AH-1；

AF=1;

CF=1;

ELSE

AF=0;CF=0;

ENDIF

AL=AL AND OFH;;AL高4位清0

写法：AAM；

作用：AH=AX DIV 10, AL=AX MOD 10;

功能：使用AAM时，通常先执行MUL/IMUL指令，将两个一字节非压缩BCD码（高四位必须为0）相乘，结果存入AX.然后使用AAM指令将AX（AH=0）调整为两字节压缩BUC码格式。

写法：AAD;

作用：AL=AH\*10+AL,AH=0;

功能：使用AAD时，通常先执行该指令，将AX中的两字节非压缩BCD码（AH与AL的高4位必须为0）调整为相应的二进制表示，然后使用DIV/IDIV指令，除以一个一字节的非压缩BCD码（高四位必须为0），可得到非压缩BCD码的除法结果。

特别注意，参加非压缩BCD码乘法或除法的操作数高4位必须为0。

-----------------------------算术指令结束-----------------------------------------------------------------------------

-----------------------------------------位操作指令开始-----------------------------------------------------

25、AND\OR\XOR\NOT\TEST

写法：

AND reg/mem,reg/mem/imm;

OR reg/mem,reg/mem/imm;

XOR reg/mem,reg/mem/imm;

NOT reg/mem;

TEST reg/mem,reg/mem/imm;

作用：AND\TEST\OR\XOR，两个操作数必须类型匹配，而且不能同时是内存操作数。

XOR通常用来将寄存器清0，如 XOR AX,AX;

TEST与AND的关系类似于CMP与SUB。TEST的典型用法是检查某位是否为1，如：

TEST DX,109H；

若 DX的第0，3，8位至少有一位为1，则 ZF=0，否则ZF=1；

26、移位指令

SHL(逻辑左移)

写法：SHL REG\mem，1\CL ;

作用：将dest的各个二进制位向左移动1（CL）位，并将DEST的最高位移出到CF，最低位移入0。

SAL（算术左移）

写法：SAL REG\mem，1\CL ;

作用：将dest的各个二进制位向左移动1（CL）位，并将DEST的最高位移出到CF，最低位移入0（同SHL）。

SHR（逻辑右移）

写法：SHR REG\mem，1\CL ;

作用：将dest的各个二进制位向左移动1（CL）位，并将DEST的最低位移出到CF，最高位移入0。

SAR(算术右移)

写法：SAR REG\mem，1\CL ;

作用：将dest的各个二进制位向左移动1（CL）位，并将DEST的最低位移出到CF，最高位不变。

SHLD(双精度左移)

写法：SHLD REG16/REG32/MEM16/MEM32, REG16/REG32, IMM8/CL;(类型须匹配)

作用：将OPRD1的各二进制左移，并将oprd1的最高位移到CF,oprd2的最高位移到oprd1的最低位，但是，oprd2的值不变。

SHRD（双精度右移）

写法与作用与双精度左移类似。注意移动方向为右移。

以上位移指令对标志位的影响：

若移位后符号位发生了变化，则OF=1，否则OF=0;CF为最后移入位；按一般规则影响ZF与SF。然而，若移位次数为0，则不影响标志位；若移位次数大于1，则OF无定义。

27、循环移位指令

ROL(循环左移)

写法：ROL REG\MEM, 1\CL；或 ROL REG/MEM,IMM8;(类型可不匹配)

作用：将DEST的各二进制位向左移动，并将最高位移出到CF,并同时移入最低位。

ROR(循环右移)

写法：ROR REG\MEM, 1\CL；或 ROR REG/MEM,IMM8;(类型可不匹配)

作用：将DEST的各二进制位向右移动，并将最低位移出到CF,并同时移入最高位。

RCL(带进位循环左移)

写法：RCL REG\MEM, 1\CL；或 RCL REG/MEM,IMM8;(类型可不匹配)

作用：将DEST的各二进制位向左移动，并将最高位移出到CF，原CF移入最低位。

RCR(带进位循环右移)

写法：RCR REG\MEM, 1\CL；或 RCR REG/MEM,IMM8;(类型可不匹配)

作用：将DEST的各二进制位向右移动，并将最低位移出到CF，原CF移入最高位。

28、位测试指令

BT（位测试）

写法：BT REG16/MEM16,REG16/IMM8;或BT REG32/MEM32,REG32/IMM8;

作用：CF=DEST的第index位，dest不变。

BTS（位测试并置位）

写法：BTS REG16/MEM16,REG16/IMM8;或BTS REG32/MEM32,REG32/IMM8;

作用：CF=DEST的第index位，dest的第index位=1；

BTR(位测试并复位)

写法：BTR REG16/MEM16,REG16/IMM8;或BTR REG32/MEM32,REG32/IMM8;

作用：CF=DEST的第index位，dest的第index位=0；

BTC(位测试并复位)

写法：BTC REG16/MEM16,REG16/IMM8;或BTC REG32/MEM32,REG32/IMM8;

作用：CF=DEST的第index位，dest的第index位取反；

说明：若dest为寄存器，则以index除以16（dest为reg16）或32（dest为reg32）的余数作为测试位。当然，index最好不要超出操作数的位数。

若dest为内存操作数，则无论其类型为字或双字，测试位为相对于起始地址的位移，例如,设BX=50，X为字类型的变量，则执行指令BT X,BX；后，CF=X+6单元的第2位，因为50%8=6余2.

BTS、BTC、BTR指令可用于并发程序设计。

29、位扫描指令

BSF(前向位扫描)

写法：BSF reg16/reg32， reg16/reg32/mem16/mem32；（类型须匹配）

作用：dest=src中值为1的最低位编号（从低位向高位搜索）

BSR(后向位扫描)

写法：BSR reg16/reg32， reg16/reg32/mem16/mem32；（类型须匹配）

作用：dest=src中值为1的最高位编号（从高位向低位搜索）

说明：BSF和BSR搜索SRC操作数中首次出现1的位置，BSF从低位向高位搜索，BSR反之。若找到一个1，则置ZF=0，并存储位编号到DEST操作数中。若SRC=0，即没有1出现，则置ZF=1，且dest的值不确定。

比如，有如下二进制数0111 1111 1010 0100

执行bsf后，位编号为2，执行bsr后，位编号为14.

30、条件置位指令

通用写法：SETcc reg8/mem8

作用：若条件cc成立，则dest=1，否则，dest=0；

SETcc有很多种命令形式，这里的cc只是一个描述符，具体的参见下面的三个表，其中，E（Equal）表示相等，G（Greatet）表示带符号大于，L（Less）表示带符号小于，A（Above）表示无符号大于，B（Below）表示无符号小于。