学习汇编语言，最关键的就在于汇编指令集的掌握以及计算机工作方式的理解，以下是80X86 汇编过程中经常用到的一些汇编指令。
从功能分类上来说，一共可分为
一、　数据传送指令：MOV、XCHG、LEA、LDS、LES、PUSH、POP、PUSHF、POPF、CBW、CWD、CWDE。
二、　算术指令：ADD、ADC、INC、SUB、SBB、DEC、CMP、MUL、DIV、DAA、DAS、AAA、AAS。
三、　逻辑指令：AND、OR、XOR、NOT、TEST、SHL、SAL、SHR、SAR、RCL、RCR、ROL、ROR。
四、　控制转移指令：JMP、Jcc、JCXZ、LOOP、LOOPZ、LOOPNZ、LOOPNE、CALL、RET、INT。
五、　串操作指令：MOVS、LODS、STOS、CMPS、SCAS。
六、　标志处理指令：CLC、STC、CLD、STD。
七、　32 位 CPU 新增指令（后续补充并完善）
除上述的一些指令外，还有许多 32 位 80X86CPU 新增指令，这些指令有时会简化程序设计，不过由于我也是刚刚学习汇编，这些都是从书上看到的，所以很多还不是十分了解，我写这些的目的仅仅是想让自己能更好的去记住这些指令的作用和用法，同事也希望和我一样刚入门的朋友能够多了解一些，并没有其他目的，所有的示例也并没有经过实际的代码测试，所以希望各位朋友，不管你喜欢不喜欢，反对不反对，请文明发言，谢谢！
----------数据传送指令开始----------
1、　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。
表一：测试单个标志位的 SETcc 指令：
SETcc 指令
描述
置 1 条件
SETC,SETB,SETNAE
有进位时置 1
CF=1
SETNC,SETNB,SETAE
无进位时置 1
CF=0
SETZ,SETE
为 0（相等）时置 1
ZF=1
SETNA,SETNE
非 0（不等）时置 1
ZF=0
SETS
为负时置 1
SF=1
SETNS
为正时置 1
SF=0

SET0
溢出时置 1
OF=1
SETNO
不溢出时置 1
OF=0
SETP,SETPE
‘1’的个数为偶数时置 1
PF=1
SETNP,SETPO
‘1’的个数为奇数时置 1
PF=0
表二：用于带符号数比较的 SETcc 指令，这些指令常用在 CMP 指令之后，以判断带符号数的大小：
SETcc 指令
描述
置 1 条件
SETG,SETNLE
大于（不小于等于）时置 1
SF=OF 且 ZF=0
SETGE,SETNL
大于等于（不小于）时置 1
SF=OF
SETL,SETNGE
小于（不大于等于）时置 1
SF≠OF
SETLE,SETNG
小于等于（不大于）时置 1
SF≠OF 或 ZF=1
表三：用于无符号数比较的 SETcc 指令，常用在 CMP 指令之后，用来判断无符号数的大小：
SETcc 指令
描述
置 1 条件
SETA,SETNBE
大于（不小于等于）时置 1
CF=0 且 ZF=0
SETAE,SETNB,SETNC
大于等于（不小于）时置 1
CF=0
SETB,SETNAE,SETC
小于（不大于等于）时置 1
CF=1
SETBE,SETNA
小于等于（不大于）时置 1

CF=1 或 ZF=1
----------位操作指令结束----------

----------控制转移指令开始----------
31、JMP(无条件转移指令)
执行代码的跳转，分为两种，一：段内转移，即要跳过去的代码地址和当前地址在同一段，这时只要修改 IP（专用寄存器--指令指针）即可；二：段间转移：即要跳过去的代码地址和当前代码地址不在同一段内，需要同时修改 CS 和 IP 的值。
写法：
1、JMP label;若 label 与该指令位于同一代码段内，IP=label 的偏移地址，否则 CS:IP=label 的分段地址，简单的说，就是跳到 label 的地址去。
2、JMP reg16/mem16；段内转移，偏移地址=reg16/[mem16]
3、JMP mem32；段间间接转移，段地址 CS=mem32 高字，偏移地址 IP=mem32 低字。
说明：当操作数是内存操作数时，若内存操作数是双字类型，则产生段间转移，若内存操作数是字类型，则产生段内间接转移。当不能确定类型时，编译器将报错。
32、Jcc（条件转移指令）
写法：Jcc label；
作用：若条件成立，则 IP=label 的偏移地址，否则，CPU 将忽略该条件转移，继续执行下一条指令。
条件转移有以下几种形式：
表一：测试单个标志位的 Jcc 指令：
Jcc 指令
描述
转移条件
JC,JB,JNAE
有进位时转移
CF=1
JNC,JNB,JAE
无进位时转移
CF=0
JZ,JE
为零（相等）时转移
ZF=1
JNZ,JNE
非零（不等）时转移
ZF=0
JS
为负时转移
SF=1
JNS
为正时转移
SF=0

JO
溢出时转移
OF=1
JON
不溢出时转移
OF=0
JP,JPE
‘1’的个数为偶数时转移
PF=1
JNP,JPO
‘1’的个数为奇数时转移
PF=0
表二：用于带符号数比较的 Jcc 指令（常用在 CMP 指令之后，以判断带符号数的大小）
Jcc 指令
描述
转移条件
JG,JNLE
大于（不小于等于）时转移
SF=OF 且 ZF=0
JGE,JNL
大于等于（不小于）时转移
SF=OF
JL,LNGE
小于（不大于等于）时转移
SF<>OF
JLE,LNG
小于等于（不大于）时转移
SF<>OF 或 ZF=1
表三：用于无符号数比较的 Jcc 指令（常用在 CNO 指令之后，以判断无符号数的大小）
Jcc 指令
描述
转移条件
JA,JNBE
大于（不小于等于）时转移
CF=0 且 ZF=0
JAE,JNB,JNC
大于等于（不小于）时转移
CF=0
JB,LNAE,JC
小于（不大于等于）时转移
CF=1
JBE,LNA
小于等于（不大于）时转移
ZF=1 或 CF=1

33、JCXZ/JECXZ（Jump if CX/ECX is zero）
写法：JCXZ label；（若 CX=0，则转移到 label）
JECXZ label；（若 ECX=0，则转移到 label）
说明：label 相对位移量必须在-126~127 之间

34、循环指令
LOOP label；
作用:CX=CX-1;若 CX<>0,则转移到 label；
LOOPZ/LOOPE label；
作用：CX=CX-1;若 CX<>0 且 ZF=1，则转移到 label；
LOOPNZ/LOOPNE label；
作用：CX=CX-1;若 CX<>0 且 ZF=0，则转移到 label；
说明：label 相对位移量必须在-128~127 之间
35、过程调用和返回指令
CALL（过程调用）
写法：CALL label；
作用：若 label 与该指令在同一代码段，则为段内直接调用，IP 进栈，IP=label 的偏移地址，如果是不在同一代码段，则为段间间接调用，CS:IP 进栈，CS:IP=label 的分段地址
写法：CALL reg16/mem16；
作用：段内间接调用，IP 进栈，IP=reg16/【mem16】
写法：CALL mem32；
作用：段间间接调用，CS:IP 进栈，CS 等于 mem32 高字，ip 等于 mem32 低字。
该指令与 JMP 指令的区别就是保存了 CS:IP 的值，这样在调用指令结束后，可以返回回来而已。
RET（过程返回）
写法：RET; 近返回或远返回
RETN； 近返回；
RETF; 远返回
RET imm16； 近返回或远返回，并调整堆栈，SP=SP+imm16；
RETN imm16；近返回，并调整堆栈，SP=SP+imm16；
RETF imm16；远返回，并调整堆栈，SP=SP+imm16；
作用：RET/RETN/RETF：返回地址出栈，从而使调用返回，其中，远返回是 POP 一个双字到 CS:IP，而近返回是 POP 一个字到 IP
RET/RETN/RETF imm16：在返回后，CPU 立即将 imm16 加到堆栈指针 SP。这种机制用来在返回前将参数从栈中移除。
说明：CALL 与 RET 必须配合使用，并且确保返回时栈顶正好是返回地址，不然就会出错。
36、IN T（中断指令）
写法：INT n；（n 为中断号，取值为 0~255）
通常，程序内部的跳转，用 JMP 或 CALL，并且 JMP 和 CALL 得参数是要跳转的过程的入口指令地址，而 INT 则是调用系统提供的中断服务程序，并且参数是中断号，然后由 CPU 根据中断号去计算中断服务程序的入口地址，MS DOS 使用中断号 21H 作为系统调用，一般 INT 中断的步骤如下：
（1）由 AH 给出中断号
（2）根据相应功能的要求，设置入口参数

（3）INT 21H
（4）分析和使用出口参数
比如如下代码实现程序的退出并返回 DOS：
Mov ah，4ch；-----给出中断号
Int 21h； -----开始中断
说明：除了直接以 AL 或 AX 返回出口参数外，INT 21H 还是用 AL 或 AX 作为返回码，对于功能号 0~2eh，由 AL 返回 0（表示成功）或 1（表示失败）；其余功能号则由 CF 返回 0 或者 1，并由 AX 返回错误码。
----------控制转移指令结束----------
----------标志处理指令开始----------
37、标志处理指令
CLC ; CF=0
STC ; CF=1
CMC ; CF=NOT CF
CLD ; DF=0
STD ; DF=1
CLI ; IF=0(应慎用)
STI ; IF=1
----------标志处理指令结束----------
----------串操作指令开始----------
到这为止，所涉及的指令都是处理一个操作数，如果要处理连续内存单元的一批数据，通常需借助于循环。而串操作指令就可以用来处理内存中的数据串，并在助记符后面加上 B、W、D 分别表示操作类型为字节、字或双字
38、MOVS（串传送）
写法：
MOVSB/MOVSW/MOVSD
功能：
ES:[DI]=DS:[SI]
If(DF=0)
Then
SI=SI+size;
DI=DI+size;
Else
SI=SI-size;
DI=DI-size;
Endif
其中，size 等于 1（B）、2（W）、4（D）.
作用：将 DS:SI 所指源串的一个字节/字/双字复制到 ES:DI 所指的内存单元，然后，若 DF=0，则 SI 和 DI 增加 1、2、4，否则减少 1、2、4.
现在有点明白为什么 SI 为源变址寄存器，而 DI 为目标变址寄存器了，而 DS 为数据段寄存

器，ES 为附加段寄存器了。

39、LODS（串载入）
写法：LODSB\LODSW\LODSD
功能：
AL/AX/EAX=DS:[SI];
IF (DF=0) THEN
SI=SI+size;
ELSE
SI=SI-size;
Endif
作用：将 DS:SI 所指源串的值复制到 AL/AX/EAX 中，然后，根据 DF 使 SI 增加或减小 1、2、4
40、STOS（串存储）
写法：
STOSB\STOSW\STOSD
功能：
ES:[DI]=AL/AX/EAX；
IF (DF=0) THEN
DI=DI+size；
ELSE
DI=DI-size；
ENDIF
作用：将 AL/AX/EAX 中的值复制到 ES:[DI]所指的内存单元中去，并根据 DF 标志位的值调整 DI
41、CMPS(串比较)
写法：CMPSB/CMPSW/CMPSD
功能：
DS:[SI]-ES:[DI];
IF (DF=0) THEN
SI=SI+size；DI=DI+size；
ELSE
SI=SI-size；DI=DI-size；
ENDIF
作用：将 DS:SI 所指内存值与 ES:DI 所指内存值进行比较，并根据比较结果设置标志位，然后，对 SI 和 DI 做相应的调整。
42、SCAS(串扫描)
写法：SCASB/SCASW/SCASD
功能：
AL/AX/EAX-ES:[DI];
IF (CF=0) THEN
DI=DI+size；
ELSE
DI=DI-size；

ENDIF
作用：将 AL/AX/EAX 与 ES:DI 所指内存值进行比较，根据比较结果设置标志位，然后根据 DF 调整相应的 DI 的值。
说明：以上串操作的共性：
DS:SI 指向源串，ES:DI 指向目的串
SI 和 DI 自动增加或减少 1、2、4,关键看 DF 及操作类型是 B\W\D
43、重复前缀
重复前缀用来和以上几个串操作指令混合使用
REP(重复)
功能：当 CX<>0 时，重复执行后面的串指令，每执行一次，CX 自动-1，该指令只能用在 MOVS\LODS\STOS 之前
REPZ/REPE(为零/等于时重复)
功能：当 CX<>0 且 ZF=1 时，重复执行后面的指令，每执行一次，CX 自动-1，该指令只能用在 CMPS\ACAS 之前。
REPNZ/REPNE（非零/不等于时重复）
功能：CX<>0 且 ZF=0 时，重复执行后面的指令，每执行一次，CX 自动-1，该指令只能用在 CMPS\ACAS 之前。
说明：REPNE SCAS(B/W/D)适用于在多字节、字、双字数据结构中搜索特定值。
----------串操作指令结束----------
----------CPU 控制指令开始----------
44、NOP（无操作）
写法：NOP；
作用：该指令不做任何事情，只占用 1 个字节，耗费一个指令执行周期。
45、HIT（暂停）
写法：HIT；
作用：HIT 使 CPU 进入暂停状态，这时 CPU 不执行任何操作，直到系统复位或发生外部中断为止，中断使 CPU 继续执行后面的指令（貌似和屏保或待机的功能类似）
46、LOCK（封锁前缀）
功能：LOCK 指令用于多处理器系统，作为某些指令的前缀，可以使 CPU 通过锁住总线等方式，抱着指令作为原子性操作，即：指令执行过程不会被打断操作。
该指令用于以下指令的前缀时，以保证原子性的对内存的“读-修改-写”操作：
1) 加法：ADD\ADC\INC\XADD
2) 减法：SUB\SBB\DEC\NEG
3) 交换：XCHG\CMPXCHG\CMPXCHG8B
4) 逻辑：AND\NOT\OR\XOR
5) 位测试：BTS\BTC\BTR
说明：其他类型指令不能加 LOCK 前缀，另外，XCHG 总是原子性操作，无论前面有没有加 LOCK 前缀。LOCK 前缀典型用于 BTS 指令，以实现多处理器环境中程序的并发执行，如：
LOCK BTS [EBX],AX
LOCK ADD [SI],AL
----------CPU 控制指令结束----------

到这里为止，一些基本的汇编指令都已经学习完了，但是还得好好的去应用，不然还真的记不住这些指令的功能。
总结一下，一般情况下，通用寄存器可以较随便使用，段寄存器和指针寄存器用来指示位置，一般不能随便更改，另外一个就是标志寄存器的各个标志位的意义也非常的重要，很多指令都是根据标志位来执行操作的。
接下来准备学习汇编的编程格式，然后就可以写一些简单的程序并自己进行调试了 ，哈哈哈哈哈。偶非常的期待。