汇编语言的准备知识-给初次接触汇编者3 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/138/2021_2022__E6_B1_87_E7_BC_96_E8_AF_AD_E8_c98_138271.htm “ 汇编语言 ” 作为一门语言，对应于高级语言的编译器，我们需要一个 “ 汇编器 ” 来把汇编语言原文件汇编成机器可执行的代码。高级的汇编器如MASM, TASM等等为我们写汇编程序提供了很多类似于高级语言的特征，比如结构化、抽象等。在这样的环境中编写的汇编程序，有很大一部分是面向汇编器的伪指令，已经类同于高级语言。现在的汇编环境已经如此高级，即使全部用汇编语言来编写windows的应用程序也是可行的，但这不是汇编语言的长处。汇编语言的长处在于编写高效且需要对机器硬件精确控制的程序。而且我想这里的人学习汇编的目的多半是为了在破解时看懂反汇编代码，很少有人真的要拿汇编语言编程序吧？（汗.....） 好了，言归正传。大多数汇编语言书都是面向汇编语言编程的，我的帖是面向机器和反汇编的，希望能起到相辅相成的作用。有了前面两篇的基础，汇编语言书上对大多数指令的介绍应该能够看懂、理解了。这里再讲一讲一些常见而操作比较复杂的指令。我这里讲的都是机器的硬指令，不针对任何汇编器。 无条件转移指令jmp: 这种跳转指令有三种方式：短(short)，近(near)和远(far)。短是指要跳至的目标地址与当前地址前后相差不超过128字节。近是指跳转的目标地址与当前地址在用一个段内，即CS的值不变，只改变EIP的值。远指跳到另一个代码段去执行，CS/EIP都要改变。短和近在编码上有所不同，在汇编指令中一般很少显式指定，只要写 jmp 目标地址，几乎任何

汇编器都会根据目标地址的距离采用适当的编码。远转移在32位系统中很少见到，原因前面已经讲过，由于有足够的线性空间，一个程序很少需要两个代码段，就连用到的系统模块也被映射到同一个地址空间。 jmp的操作数自然是目标地址，这个指令支持直接寻址和间接寻址。间接寻址又可分为寄存器间接寻址和内存间接寻址。举例如下(32位系统): jmp 8E347D60 .直接寻址段内跳转 jmp EBX .寄存器间接寻址：只能段内跳转 jmp dword ptr [EBX] .内存间接寻址，段内跳转 jmp dword ptr [00903DEC] .同上 jmp fward ptr [00903DF0] .内存间接寻址，段间跳转 解释： 在32位系统中，完整目标地址由16位段选择子和32位偏移量组成。因为寄存器的宽度是32位，因此寄存器间接寻址只能给出32位偏移量，所以只能是段内近转移。在内存间接寻址时，指令后面是方括号内的有效地址，在这个地址上存放跳转的目标地址。比如，在[00903DEC]处有如下数据：7C 82 59 00 A7 01 85 65 9F 01 内存字节是连续存放的，如何确定取多少作为目标地址呢？dword ptr 指明该有效地址指明的是双字，所以取 0059827C 作段内跳转。反之，fward ptr 指明后面的有效地址是指向48位完全地址，所以取19F:658501A7 做远跳转。 注意：在保护模式下，如果段间转移涉及优先级的变化，则有一系列复杂的保护 检查，现在可不加理会。将来等各位功力提升以后可以自己去学习。 条件转移指令jxx:只能作段内转移，且只支持直接寻址。

========== 调用指令CALL: Call的寻址方式与jmp基本相同，但为了从子程序返回，该指令在跳转以前会把紧接着它的下一条指令的地

址压进堆栈。如果是段内调用（目标地址是32位偏移量），则压入的也只是一个偏移量。如果是段间调用（目标地址是48位全地址），则也压入下一条指令的完全地址。同样，如果段间转移涉及优先级的变化，则有一系列复杂的保护检查。 与之对应retn/retf指令则从子程序返回。它从堆栈上取得返回地址（是call指令压进去的）并跳到该地址执行。retn取32位偏移量作段内返回，retf取48位全地址作段间返回。retn/f 还可以跟一个立即数作为操作数，该数实际上是从堆栈上传给子程序的参数的个数（以字计）返回后自动把堆栈指针esp加上指定的数*2，从而丢弃堆栈中的参数。这里具体的细节留待下一篇讲述。 虽然call和ret设计为一起工作，但它们之间没有必然的联系。就是说，如果你直接用push指令向堆栈中压入一个数，然后执行ret，他同样会把你压入的数作为返回地址，而跳到那里去执行。这种非正常的流程转移可以被用作反跟踪手段。