Brennan's Guide to Inline Assembly

文档版本 1.1.2.2

这篇文档是关于DJGPP内置汇编语言的介绍。DJGPP基于GCC，因此它使用AT&T、Unix语法 并且有一些独特的汇编语言方法。

下面描述了Intel语法和AT&T语法的区别

1. 语法

* **寄存器命名**

寄存器命名被加上%前缀。如下例，为了引用eax寄存器：

AT&T: %eax

Intel: eax

* **源/目的 顺序**

在AT&T语法中，源操作数总是在左边，目的操作数总是在右边(Intel语法是相反的)。因此如果我们加载ebx的值到eax中：

AT&T: movl %eax,%ebx

Intel: mov ebx, eax

* **常量值/立即数 格式**

在AT&T语法中常量值/立即数 以”$”开头，如果我们从读取静态地址变量\_booga的值到eax中，那么语法如下：

AT&T: movl $\_booga, %eax

Intel: mov eax, \_booga

如果我们读取0xd00d的值到ebx：

AT&T: movl $0xd00d, %ebx

Intel: mov ebx, 0xd00dh

* **操作符大小**

你必须在指令添加后缀，后缀的值为b、w或者l，以此明确目的寄存器的宽度是byte、word或者longword。如果你忽略了，那么GAS（GNU汇编器）会试图猜测位数。你一定不希望它猜测然后猜错了。所以不要忘记在指令后面指定位宽。

AT&T: movw %ax, %bx

Intel: mov bx,ax

* **引用内存**

DJGPP使用386保护模式，所以你可以忘记所有的真实模式寻址，包括在那一个寄存器上有默认的段的限制，这样寄存器可以是基址指针或者索引指针(SI、DI)。

下面是典型的32位地址格式：

AT&T：immed32(basepointer,indexpointer,indexscale)

Intel：[basepointer + indexpointer\*indexscale + immed32]

你可以认为计算地址的公式如下：

immed32 + basepoint + indexpointer\*indexscale

你不必使用所有的这些域，但是至少必须使用immed32,basepointer之中的一个，并在指令操作符的后面一定要加上位宽后缀。如下例：

* + 寻址一个特定的C变量：

AT&T: \_booga

Intel: [\_booga]

注意：下划线是你怎样从汇编器获得静态（全局）C 变量。只有当用于全局变量时有效，否则需要使用扩展的asm来将变量提前加载到寄存器中。

* + 寻址寄存器指向的地址

AT&T: (%eax)

Intel: [eax]

* + 寻址基于寄存器的值的地址偏移后的地址

AT&T：\_variable(%eax)

Intel：[eax+\_variable]

* + 寻址一个整数数组中的值（以4间隔）

AT&T：\_array(,%eax,4)

Intel：[eax\*4 + array]

* + 可以对立即数做一些简单的数学运算：

AT&T:\_struct\_pointer+8

Intel: 同上

* + 在八个字符的数组中寻址特定的字符：假定eax存了期望的字符，ebx存了字符的偏移量

AT&T: \_array(%ebx,%eax,8)

Intel: [ebx+eax\*8+\_array]