# C语言：ASM汇编内嵌语法

## 1.基本内联汇编

**基本内联汇编的格式是**

\_\_asm\_\_ \_\_volatile\_\_("Instruction List");

**关键字**

“\_\_asm\_\_” 表示后面的代码为内嵌汇编，“asm”是“\_\_asm\_\_”的别名。

“\_\_volatile\_\_” 表示编译器不要优化代码，后面的指令保留原样，“volatile”是它的别名。

**内嵌汇编语法如下**

\_\_asm\_\_(

汇编语句模板

: 输出部分

: 输入部分

: 破坏描述部分)

### 2.3.1 汇编语句模板

汇编语句模板由汇编语句序列组成，语句之间使用“;”、“\n”或“\n\t”分开。

指令中的操作数可以使用占位符引用C语言变量，操作数占位符最多10个，名称如下：%0，%1…，%9。

指令中使用占位符表示的操作数，总被视为long型（4，个字节），但对其施加的操作 根据指令可以是字或者字节，当把操作数当作字或者字节使用时，默认为低字或者低字节。

对字节操作可以显式的指明是低字节还是次字节。方法是在%和序号之间插入一个字母， “b”代表低字节，“h”代表高字节，例如：%h1。

### 2.3.2 输出部分

输出部分描述输出操作数，不同的操作数描述符之间用逗号格开，每个操作数描述符由限定字符串和 C语言变量组成。每个输出操作数的限定字符串必须包含“=”表示他是一个输出操作数。

例：

\_\_asm\_\_ \_\_volatile\_\_("pushfl ; popl %0 ; cli":"=g" (x) )

描述符字符串表示对该变量的限制条件，这样GCC就可以根据这些条件决定如何分配寄存器，如何产生必要的代码处理指令操作数与C表达式或C变量之间的联系。

### 2.3.3 输入部分

输入部分描述输入操作数，不同的操作数描述符之间使用逗号格开，每个操作数描述符由

限定字符串和C语言表达式或者C语言变量组成。

Static \_\_inline\_\_ void \_\_set\_bit(int nr, volatile void \* addr)

{

\_\_asm\_\_( "btsl%1,%0" : "=m"(ADDR) : "Ir"(nr));

}

此例功能是将(\*addr)的第nr位设为1。

第一个占位符%0与C语言变量ADDR 对应，

第二个占位符%1与C语言变量nr对应。

因此上面的汇编语句代码与下面的伪代码等价： btsl nr, ADDR，

该指令的两个操作数不能全是内存变量，因此将nr的限定字符串指定为“Ir”， 将nr与立即数或者寄存器相关联，这样两个操作数中只有ADDR为内存变量。

### 2.3.4 限制字符

限制字符有很多种，有些是与特定体系结构相关，此处仅列出常用的限定字符和i386 中可能用到的一些常用的限定符。它们的作用是指示编译器如何处理其后的C语言变量与指令操作数之间的关系，例如是将变量放在寄存器中还是放在内存中等，

下表列出了常用的限定字母。

“a”将输入变量放入eax

“b”将输入变量放入ebx

“c”将输入变量放入ecx

“d”将输入变量放入edx

“s”将输入变量放入esi

“d”将输入变量放入edi

“q”将输入变量放入eax，ebx ，ecx ，edx中的一个

“r”将输入变量放入通用寄存器，也就是eax ，ebx，ecx，edx，esi，edi中的一个

“A”把eax和edx，合成一个64位的寄存器(uselong longs)

“m”内存变量

“o”操作数为内存变量，但是其寻址方式是偏移量类型，也即是基址寻址，或者是基址加变址寻址

“V”操作数为内存变量，但寻址方式不是偏移量类型

“,” 操作数为内存变量，但寻址方式为自动增量

“p”操作数是一个合法的内存地址（指针）

寄存器或内存

“g” 将输入变量放入eax，ebx，ecx ，edx中的一个或者作为内存变量

“X”操作数可以是任何类型

立即数

“I” 0-31 之间的立即数（用于32位移位指令）

“J” 0-63 之间的立即数（用于64 位移位指令）

“N” 0-255 ，之间的立即数（用于out 指令）

“i” 立即数

“n” 立即数，有些系统不支持除字以外的立即数，这些系统应该使用“n”而不是“i”

匹配

“0”，“1 ，”... “9 ”

表示用它限制的操作数与某个指定的操作数匹配，也即该操作数就是指定的那个操作数，

例如用“0 ”去描述“％1”操作数，那么“%1”引用的其实就是“%0”操作数，注意作为

限定符字母的0－9 ，与指令中的“％0”－“％9”的区别，前者描述操作数，后者代表操作数。

操作数类型

“=” 操作数在指令中是只写的（输出操作数）

“+” 操作数在指令中是读写类型的（输入输出操作数）

浮点数

“f”

浮点寄存器

“t”第一个浮点寄存器

“u”第二个浮点寄存器

“G”标准的80387

浮点常数

% 该操作数可以和下一个操作数交换位置

例如addl的两个操作数可以交换顺序（当然两个操作数都不能是立即数）

# 部分注释，从该字符到其后的逗号之间所有字母被忽略

\* 表示如果选用寄存器，则其后的字母被忽略

现在继续看上面的例子，

"=m" (ADDR)表示ADDR为内存变量（“m”），而且是输出变量（“=”）；

"Ir" (nr)表示nr，为 0－31之间的立即数（“I”）或者一个寄存器操作数（“r”）。

## 2．.c文件生成.s文件的Gcc编译命令

gcc -O -S main.c

选项-O表示优化编译，我们还可以指定优化等级，比如-O2表示优化等级为2；

选项-S表示将C/C++源文件编译为汇编文件，文件名和C/C++文件一样，只不过扩展名由.c变为.s

## 3. 内嵌汇编 %0，%1

\_\_asm\_\_ \_\_violate\_\_(

"movl %1,%0" //指令模版

: "=r" (result) //”=r”，”m” : 限制符，”=”表示输出

: "m" (input) // result, input：C语言表达式

);

“movl %1,%0”是指令模板；

“%0”和“%1”代表指令的操作数，称为占位符，内嵌汇编靠它们将c语言表达式与指令操作数相对应。

指令模板后面用小括号括起来的是C语言表达式，本例中只有两个：“result”和“input”，他们按照出现的顺序分别与指令操作数“%0”，“%1，”对应；

注意对应顺序：第一个C表达式对应“%0”；第二个表达式对应“%1”，依次类推，操作数至多有10个，分别用“%0”，“%1”….“%9，”表示。

在每个操作数前面有一个用引号括起来的字符串，字符串的内容是对该操作数的限制或者说要求。

“result”前面的限制字符串是“=r”，

“=”表示“result”是输出操作数，

“r”表示需要将“result”与某个通用寄存器相关联，

先将操作数的值读入寄存器，然后在指令中使用相应寄存器，而不是“result”本身，

指令执行完后需要将寄存器中的值存入变量“result”，

从表面上看好像是指令直接对“result”进行操作，实际上GCC做了隐式处理，这样我们可以少写一些指令。

“input” 前面的限制字符串是“r”，

“r”表示该表达式需要先放入某个寄存器，然后在指令中使用该寄存器参加运算。