GCC 内联汇编

首先，让我们来共同了解一下 GCC 内联汇编的一般格式:

```
asm(
    代码列表
    :输出运算符列表
    :输入运算符列表
    :被更改资源列表
);
```

在 C 代码中嵌入汇编需要使用 asm 关键字，在 asm 的修饰下，代码列表、输出运算符列表、输入运算符列表和被更改的资源列表这 4 个部分被 3 个 ":"分隔。这种格式虽然简单，但并不足以说明问题，我们还是通过例子来解释它吧。

```
void test(void)
{
    asm(
    "mov r1,#1\n"
    :
    :
    :"r1"
    );
}
```

在上面的代码中，函数 test 中内嵌了一条汇编指令实现将立即数 1 赋值给寄存器 r1 的操作。由于没有任何形式的输出和输入，因此输出和输入列表的位置上什么都没有填写。但是，我们发现在更改资源列表中，出现了寄存器 r1 的身影，这表示我们通知了编译器，在汇编代码执行过程中 r1 寄存器会被修改。

```
void    test(void)
{
    ....
    asm(
    "stmfd sp!,{r1}\n"
    "mov r1,#1\n"
    "ldmfd sp!,{r1}\n"
    );
    ....
}
```

这段代码内联汇编既无输出也无输入，也没有资源被更改，只留下了汇编代码部分。

C 与内联汇编的交互

```
void test(void)
{
    int tmp = 5;
    asm(
    "mov     r4 ,   %0 \n"
    "add     %0 , r4 , #1 \n"
    :"r"(tmp)
    :"r"(tmp)
    :"r4"
    );
    Printf( "tmp = %d \n ",tmp);
}
```

代码中有一条 mov 指令，该指令将%0 赋值给 r4。这里，符号%0 代表出现在输入运算符列表和输出运算符列表的第一个值。如果%1 存在的话，那么它就代表出现在列表中的第二个值，依次类推。所以，该段代码中，%0 代表的就是"r"(tmp)这个表达式的值了。

那么这个新的表达式又该怎样解释呢？原来 ,在"r"(tmp)这个表达式中，tmp 代表的正是 C 语言向内联汇编输入的变量，操作符"r"则代表 tmp 的值会通过某一个寄存器来传递。在 gcc 中与之类似的操作符还包括"m","I"等等，其含义如下表

操作符	
操作符	含义
r	通用寄存器 R0～R15
m	一个有效的内存地址
I	数据处理中的立即数
X	被修饰的操作符只能做为输出

当 C 语言需要利用内联汇编输出结果时，可以使用输出运算符列表来实现，其格式如下

```
void test(void)
{
    int tmp;
    asm(
    "mov %0,#1\n"
      :"=r"(tmp)
      :
    );
}
```

原本应出现在输入运算符列表中的运算符，现在出现在了输出运算符列表中，同时变量 tmp 将会存储内联汇编的输出结果。这里有一点可能已经引起大家的注意了，代码中操作符 r 的前面多了一个"="。这个等号被称为约束修饰符，其作用是对内联汇编的操作符进行修饰。

CC 的修饰符	
修饰符	说明

无	被修饰的操作符是只读的
=	被修饰的操作符只写
+	被修饰的操作符具有可读可写的属性
&	被修饰的操作符只能作为输出

当一个操作符没有修饰符对其进行修饰时，代表这个操作符是只读的，在上述代码中，我们需要将内联汇编的结果输出出来，那么至少要保证该操作符是可写的。因此,"="或者"+"也就必不可少了。

这里只是介绍了 C 内联汇编的常用用法，更复杂的语法读者可以通过相关文档自己学习。