本笔记来自网络。

在针对ARM体系结构的编程中，一般很难直接使用C语言产生操作协处理器的相关代码，因此使用汇编语言来实现就成为了唯一的选择。但如果完全通过汇编代码实现，又会过于复杂、难以调试。因此，C语言内嵌汇编的方式倒是一个不错的选择。然而，使用内联汇编的一个主要问题是，内联汇编的语法格式与使用的编译器直接相关，也就是说，使用不同的C编译器内联汇编代码时，它们的写法是各不相同的。下面介绍在ARM体系结构下GCC的内联汇编。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。下面，我们看一个例子：

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

注：换行符和制表符的使用可以使得指令列表看起来变得美观。你第一次看起来可能有点怪异，但是当C编译器编译C语句的是候，它就是按照上面（换行和制表）生成汇编的。

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，在汇编代码执行过程中R1寄

存器会被修改，因此为了通知编译器，在被更改资源列表中，需要写上寄存器R1。

寄存器被修改这种现象发生的频率还是比较高的。例如，在调用某段汇编程序之前，寄存器R1可能已经保存了某个重要数据，当汇编指令被调用之后，R1寄存器被赋予了新的值，原来的值就会被修改，所以，需要将会被修改的寄存器放入到被更改资源列表中，这样编译器会自动帮助我们解决这个问题。也可以说，出现在被更改资源列表中的资源会在调用汇编代码一开始就首先保存起来，然后在汇编代码结束时释放出去。所以，上面的代码与如下代码从语义上来说是等价的。

```
1.  void test(void)
2.  {
3.      ……
4.      asm(
5.          "stmfd sp!,{r1}\n"
6.          "mov r1,#1\n"
7.          "ldmfd sp!,{r1}\n"
8.      );
9.      ……
10. }
```

这段代码中的内联汇编既无输出又无输入，也没有资源被更改，只留下了汇编代码的部分。由于程序在修改R1之前已经将寄存器R1的值压入了堆栈，在使用完之后，又将R1的值从堆栈中弹出，所以，通过被更改资源列表来临时保存R1的值就没什么必要了。

在以上两段代码中，汇编指令都是独立运行的。但更多的时候，C和内联汇编之间会存在一种交互。C程序需要把某些值传递给内联汇编运算，内联汇编也会把运算结果输出给C代码。此时就可以通过将适当的值列在输入运算符列表和输出运算符列表中来实现这一要求。请看下面的例子：

```
1.  void test(void)
2.  {
3.      int tmp=5;
4.      asm(
5.          "mov r4,%0\n"
6.          :
7.          :"r"(tmp)
8.          :"r4"
9.      );
10. }
```

上面的代码中有一条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中的第一个值。如果%1存在的话，那么它就代表出现在列表中的第二个值，依此类推。所以，在该段代码中，%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GCC4中与之相类似的操作符还包括“m”、“I”，等等，其含义见下表：

```
1.  void test(void)
2.  {
3.      int tmp;
4.      asm(
5.          "mov %0,#1\n"
6.          :"=r"(tmp)
7.          :
8.      );
9.  }
```

在上面的代码中，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，上面的代码中操作符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。几种修饰符的含义如下表所示：

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。当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。