__builtin_ffs(x)：返回 **x** 中最后一个为 **1** 的位是从后向前的第几位，如 **__builtin_ffs(0x789)=1, __builtin_ffs(0x78c)=3**。于是，**__builtin_ffs(x) - 1** 就是 **x** 中最后一个为 **1** 的位的位置。

__builtin_popcount(x)：**x** 中 **1** 的个数。

__builtin_ctz(x)：**x** 末尾 **0** 的个数。**x=0** 时结果未定义。

__builtin_clz(x)：**x** 前导 **0** 的个数。**x=0** 时结果未定义。

上面的宏中 **x** 都是 **unsigned int** 型的，如果传入 **signed** 或者是 **char** 型，会被强制转换成 **unsigned int**。

__builtin_parity(x)：**x** 中 **1** 的奇偶性。

__builtin_return_address(n)：当前函数的第 **n** 级调用者的地址，用的最多的就是 **__builtin_return_address(0)**，即获得当前函数的调用者的地址。注意，该函数实现是体系结构相关的，有些体系结构只实现了 **n=0** 的返回结果。

uint16_t __builtin_bswap16 (uint16_t x)
uint32_t __builtin_bswap32 (uint32_t x)：按字节翻转 **x**，返回翻转后的结果。

__builtin_prefetch (const void *addr, ...)：它通过对数据手工预取的方法，在使用地址 **addr** 的值之前就将其放到 **cache** 中，减少了读取延迟，从而提高了性能，但该函数也需要 **CPU** 的支持。该函数可接受三个参数，第一个参数 **addr** 是要预取的数据的地址，第二个参数可设置为 **0** 或 **1**（**1** 表示我对地址 **addr** 要进行写操作，**0** 表示要进行读操作），第三个参数可取 **0-3**（**0** 表示不用关心时间局部性，取完 **addr** 的值之后便不用留在 **cache** 中，而 **1**、**2**、**3** 表示时间局部性逐渐增强）。

__builtin_constant_p (exp)：判断 **exp** 是否在编译时就可以确定其为常量，如果

exp 为常量，该函数返回 **1**，否则返回 **0**。如果 **exp** 为常量，可以在代码中做一些优化来减少处理 **exp** 的复杂度。

__builtin_types_compatible_p(type1, type2)：判断 **type1** 和 **type2** 是否是相同的数据类型，相同返回 **1**，否则返回 **0**。该函数不区分 **const/volatile** 这样的修饰符，即 **int** 和 **const int** 被认为是相同的类型。

```
#define foo(x)
({
typeof(x) tmp = (x);\
if(__builtin_types_compatible_p(typeof(x), int))\
//do something...\
else \
//do something...\
tmp;
})
```

__builtin_expect (long exp, long c)：用来引导 **gcc** 进行条件分支预测。在一条指令执行时，由于流水线的作用，**CPU** 可以完成下一条指令的取指，这样可以提高 **CPU** 的利用率。在执行一条条件分支指令时，**CPU** 也会预取下一条执行，但是如果条件分支跳转到了其他指令，那 **CPU** 预取的下一条指令就没用了，这样就降低了流水线的效率。内核中的 **likely()**和 **unlikely()**就是通过**__builtin_expect** 来实现的。

__builtin_expect (long exp, long c)函数可以优化程序编译后的指令序列，使指令尽可能的顺序执行，从而提高 **CPU** 预取指令的正确率。该函数的第二个参数 **c** 可取 **0** 和 **1**，

例如：

```
if (__builtin_expect (x, 0))
          foo ();
```

表示 **x** 的值大部分情况下可能为 **0**，因此 **foo()**函数得到执行的机会比较少。**gcc** 就不必将 **foo()**函数的汇编指令紧挨着 **if** 条件跳转指令。

由于第二个参数只能取整数，所以如果要判断指针或字符串，可以像下面这样写：

```
if (__builtin_expect (ptr != NULL, 1))
          foo (*ptr);
```

表示 **ptr** 一般不会为 **NULL**，所以 **foo** 函数得到执行的概率较大，**gcc** 会将 **foo** 函数的汇编指令放在挨着 **if** 跳转执行的位置。