> abs(x)
> abs64(x)

```
/**
 * upper_32_bits - return bits 32-63 of a number
 * @n: the number we're accessing
 *
 * A basic shift-right of a 64- or 32-bit quantity.  Use this to suppress
 * the "right shift count >= width of type" warning when that quantity is
 * 32-bits.
 */
#define upper_32_bits(n) ((u32)(((n) >> 16) >> 16))

/**
 * lower_32_bits - return bits 0-31 of a number
 * @n: the number we're accessing
 */
#define lower_32_bits(n) ((u32)(n))
```

Sample:

```
uint64_t n64 = 0x1234567887654321;
printk("upper-32-bit: 0x%x\n", upper_32_bits(n64));
printk("lower-32-bit: 0x%x\n", lower_32_bits(n64));
```

output:

```
upper-32-bit: 0x12345678
lower-32-bit: 0x87654321
```

> round_up(x, y)
> round_down(x, y)

```
    printk("round_up\(23, 8\) = %u\n", round_up(23, 8));
    printk("round_down\(23, 8\) = %u\n", round_down(23, 8));
```

output:

```
round_up(23, 8) = 24
round_down(23, 8) = 16
```

```
#define DIV_ROUND_UP(n,d) (((n) + (d) - 1) / (d))
```

```
    printk("DIV_ROUND_UP\(23, 8\) = %u\n", DIV_ROUND_UP(23, 8));
```

output:

> DIV_ROUND_UP(23, 8) = 3

> barrier();

```
    if (unlikely(con == 0)) {
        printk("con is zero\n");
    } else {
        printk("con is nonzero\n");
    }
```

> swap(a, b)
> min(x, y)
> max(x, y)
> clamp(val, lo, hi)
> min3(x, y, z)
> max3(x, y, z)

求平方根 **`int_sqrt(unsigned long x)`**

```
/**
 * int_sqrt - rough approximation to sqrt
 * @x: integer of which to calculate the sqrt
 *
 * A very rough approximation to the sqrt() function.
 */
unsigned long int_sqrt(unsigned long x)
```

```
/*
 * Divide positive or negative dividend by positive divisor and round
 * to closest integer. Result is undefined for negative divisors and
 * for negative dividends if the divisor variable type is unsigned.
 */
#define DIV_ROUND_CLOSEST(x, divisor)(          \
```

sample:

```
    printk("DIV_ROUND_CLOSEST\(23, 8\) = %u\n", DIV_ROUND_CLOSEST(23, 8));
    printk("DIV_ROUND_CLOSEST\(23, 5\) = %u\n", DIV_ROUND_CLOSEST(23, 5));
```

output:

```
DIV_ROUND_CLOSEST(23, 8) = 3
DIV_ROUND_CLOSEST(23, 5) = 5
```

```
/**
 * might_sleep - annotation for functions that can sleep
 *
 * this macro will print a stack trace if it is executed in an atomic
 * context (spinlock, irq-handler, ...).
 *
 * This is a useful debugging help to be able to catch problems early and not
 * be bitten later when the calling function happens to sleep when it is not
 * supposed to.
 */
# define might_sleep()
```

`might_sleep()` 有助于debug.
标记该code是可能 `sleep` 的，如果是在不能sleep的场合(atomic context)，那么马上 `stack trace` 。