2.60

unsigned replace\_byte(unsigned x,int i,unsigned char b)

{

int shift\_w=i<<3;

unsigned right=x&((1<<shift\_w)-1);

unsigned left=((x>>(shift\_w+8)<<8)|b)<<shift\_w;

return right|left;

}

**shift\_w** 表示在第i个byte右边还有几个bits

**right**通过对原数&一个长为shift\_w的全为1的二进制数来得到，其目的是截取第i个byte右边的数字

**left** 通过将x左移(shift\_w+8)位，再右移8位，使得第i个byte全为0，然后与b进行或操作，再右移shift\_w位，得到左边的数字，此时left右边全为0

将 right与left或起来就是答案

2.61

while (T--)

{

printf("Enter x(int):\n");

if (!scanf("%u",&x)) break;

printf("A:%d\n",!!x);

printf("B:%d\n",!!~x);

printf("C:%d\n",!!(x&255));

printf("D:%d\n",!!((x>>((sizeof(int)-1)<<3))^255));

}

A：如果x有任意一位是1，换言之x不为0，用！！x就可以等价于x！＝0

B：如果x有任意一位是0，将x取反，就是～x有任意一位是1，套用A中的！！运算即可

C：如果x的最低byte有任意一位是1，将x与255进行&运算，提取最低byte，套用A中！！运算即可

D：如果x的最高byte有任意一位是0，将x右移28位，将x与255进行异或运算，提取最高byte并取反，套用A中！！运算即可

2.65

int odd\_ones(unsigned x)

{

/\*int w\_16=(x>>16)^(x&65536);

int w\_8=(w\_16>>8)^(w\_16&255);

int w\_4=(w\_8>>4)^(w\_8&15);

int w\_2=(w\_4>>2)^(w\_4&3);

int w\_1=(w\_2>>1)^(w\_2&1)\*/

x=x^(x>>1);

x=x^(x>>2);

x=x^(x>>4);

x=x^(x>>8);

x=x^(x>>16);

return x&1;

}

在注释中的是我的第一个版本，其方法很好理解，但是超过了最高操作次数限制。其思想是讲一个数分为两端，分别异或，递归对异或后的数进行操作。

第二个版本的思想也类似，关键是其不是分为两段，而是先每隔1(=2^0)位分为两段，递归的时候处理每隔2(=2^1)位的情况，以此类推。