Datalab实验报告

bitXor

当x, y 有一个为1时(x&y) 与~(x &~y)必全为1，则返回值为1。

```
int bitXor(int x, int y) {
    return ~(~x & ~y) & ~(x & y);
}
```

thirdBits

设置初值为八进制的111，每次将其向右移动对应位并将原数或上，一共移动两次则填满一个字节。

```
int thirdBits(void) {
    int t = 0111;
    int p = (t << 9) | t;
    return (p << 18) | p;
}
```

fitsShort

用t表示x的第16位与q表示x的符号位，当t与q不相等时则它表示的正数大于等于 2^{16} 或小于 -2^{16}，则不可以被16位表示，反之这可以被表示。

```
int fitsShort(int x) {
    int t = x >> 15;
    int q = t >> 16;
    return !(q ^ t);
}
```

isTmax

用t表示x+1， 当t为tmax或-1时，符号位会发生变化，其余情况则不变。当t为tmin时t+t为0，以此判断t为-1的情况。

```
int isTmax(int x) {
    int t = x + 1;
    return !((t + t) | !t);
}
```

fitbits

与fitshort类似，取出x的符号位和x的最高位，当它们不同时表示越界，不可以被正确表示。

```
int fitsBits(int x, int n) {
    int p = ~0;
    int t = x >> n + p;
    int q = t >> 16;
    return !(q ^ t);
}
```

upperbits

先判断n是否为0，若n不为0则构造tmin并右移n-1位。

```
int upperBits(int n) {
    int t = !!n;
    int p = t << 31;
    return p >> (n + ~0);
}
```

anyOddBit

构造掩码为奇数位为1，偶数位为0，将掩码与原数位与可以保留原数所有奇数位的值病删去所有偶数位的值。

```
int anyOddBit(int x) {
    int t = 0xaa;
    t = t << 8 | t;
    t = t << 16 | t;
    int p = t & x;
    return !!p;
}
```

byteSwap

先将需要交换的byte位n ,m映射到bit的单位的p，q上，通过a = a ^ b, b = a ^ b, a = a ^ b 的方式对x >> p的低四位与x>>q的低四位交换。

```
int byteSwap(int x, int n, int m) {
    int p = m << 3;
    int q = n << 3;
    int t = ((x >> p) ^ (x >> q)) & 0xff;
    return x ^ ((t << p) | (t << q));
}
```

absVal

x先将x的符号位取出为p, 若x为正数, p = 0, 则保持不变，若x为负数, p = -1,(x + p) ^ p等效于将x-1并取反得到相反数。

```
int absVal(int x) {
    int t = x >> 31;
    return (x + t) ^ t;
}
```

divpwr2

首先取出x的符号位，当x为正数时，直接右移1位，否则x加上1<<n后再右移n确保负数向0舍入。

```
int divpwr2(int x, int n) {
    int t = x >> 31;
    int p = 1 << n;
    return (x + t + (p & t)) >> n;
}
```

float_neg

当先将res设为x^{tmin}, 当x << 1后的25 - 32位全为1即大于0xff000000时表示x为NAN， 返回x，否则返回res。

```
unsigned float_neg(unsigned uf) {
    unsigned sign = 0x80000000;
    unsigned res = uf ^ sign;
    unsigned ut = 0xff000000;
    if(ut < uf << 1) return uf;
    return res;
}
```

logicalNeg

当x为0或tmin时，x的相反数为其本身，判断x与其相反数的符号是否全为0，若不全为0，x与其相反数位或后右移31位后为-1，加1后返回0，否则右移后为0，加1后返回1。

```
int logicalNeg(int x) {
    int t = ~x + 1;
    int p = (t | x) >> 31;
    return p + 1;
}
```

bitMask

a为highbit即一下都设为1，b为lowbit以上都设为1，返回a&b，当highbit小于lowbit时，a与b为1的部分无交集，返回0。

```
int bitMask(int highbit, int lowbit) {
    int t = ~0;
    int a = (1 << highbit << 1) + t;
    int b = t << lowbit;
    return a & b;
}
```

isGreater

将p的符号位设为x^y，t设为x-y-1，当p的符号位与y的符号位都为1时，则x与y符号不同且y<0则返回1，当p=0且 t=0时，则x与y符号相同且x-y-1>=0即x>y则返回1，其余情况返回0。

```
int isGreater(int x, int y) {
    int p = x ^ y;
    int t = x + ~y;
    return (~(t | p) | (p & y)) >> 31 & 1;
}
```

logicalShift

首先将x算数右移n位，当x的符号位为1时，设置符号位以上为0，符号位及一下的位为1的掩码，与算数右移后的结果作位与。

```
int logicalShift(int x, int n) {
    int t = x >> n;
    int sign = 1 << 31 >> n;
    sign = ~(sign << 1);
    return t & sign;
}
```

satMul2

判断x第31位和第32位的数是否相等，若相等返回x << 1, 否则返回tmin ^ x << 1 >> 31。

```
int satMul2(int x) {
    int t = x << 1;
    int p = (x ^ t) >> 31;
    int tmin = 1 << 31;
    int q = tmin ^ (t >> 31);
    return (t & ~p) + (q & p);
}
```

subOK

计算x，y, x-y的符号位，当x与y符号相同时或x-y与y的符号不同时返回1，否则返回0。

```
int subOK(int x, int y) {
    int t = ~y;
    int sub = x + t + 1;
    int op = x ^ t;
    int r = y ^ sub;
    return (op | r) >> 31 & 1;
}
```

trueThreeFourths

先通过x后三十位计算无舍入部分为x+(x <<1),再通过x的前两位计算舍入。为保证向0舍入，当x为负数时小数位在乘3时同时加上3.最后返回。

```
int trueThreeFourths(int x)
{
    int t = x & 3;
    x = x >> 2;
    int exp = x + (x << 1);
    int frac = t + t + t + (x >> 31 & 3) >> 2;
    return exp + frac;
}
```

isPower2

先计算t=x-1&x, 当x为2的整数倍时，t为0，最后通过(x-1) >> 30是否为0排除x=0和tmin的情况。

```
int isPower2(int x) {
    int p = x + ~0;
    int t = x & p;
    int q = p >> 30;
    return !(t | q);
}
```

float_i2f

首先得出原数的符号位，并将原数转化为它的绝对值。当原数为0时，特判返回0。之后将原数的最高位1移至32位，同时计算其阶码位的值。最后通过判断移动之后数的后十位，当第8位为1且前7位不同时为0时向上舍入，当第8位与第9位同时为1是向偶数舍入。sign + (shouldLeftShift << 23) + (ux >> 9)。

```
unsigned float_i2f(int x)
{
    unsigned tmin = 0x80000000;
    unsigned sign = x & tmin;
    if(sign) x = -x;
    unsigned ux = x;
    if(!ux) return 0;
    unsigned shouldLeftShift = 159, temp = 0;
    do{
        temp = ux;
        ux = ux << 1;
        shouldLeftShift -- ;
    }while(temp < tmin);
    if((temp & 0x0ff) > 0x080 || (temp & 0x0180) == 0x0180)  sign += 1;
    return  sign + (shouldLeftShift << 23) + (ux >> 9);
}
```

howManyBits

首先对原数进行符号位取反，之后利用分治的方法计算原数最高位1的位置，最后将所得结果加1来代表所需位数的符号位。

比较直观的代码

```
int howManyBits(int x) {
    int sign = x >> 31;
    x = x ^ sign;
    int gt16 = !!(x >> 16) << 4;
    x = x >> gt16;
    int gt8 = !!(x >> 8) << 3;
    x = x >> gt8;
    int gt4 = !!(x >> 4) << 2;
    x = x >> gt4;
    int gt2 = !!(x >> 2) << 1;
    x = x >> gt2;
    int gt1 = x >> 1;
    x = x >> gt1;
    return gt16 + gt8 + gt4 + gt2 + gt1 + x + 1;
}
```

用trick优化后的代码

```
int howManyBits(int x) {
    x = x ^ (x << 1);
    int gt16 = !!(x >> 16) << 4;
    x = x >> gt16;
    int gt8 = !!(x >> 8) << 3;
    x = x >> gt8;
    int gt4 = !!(x >> 4) << 2;
    x = x >> gt4;
    int t = !(x >> 2);
    int gt2 = (t << 1) ^ 3;
    x = x >> gt2;
    int gt1 = x & 1;
    return gt16 + gt8 + gt4 + gt2 + gt1;
}
```

float_half

先取出浮点数的符号位，并计算取半之后的偏移量b =(x & 3 == 3)再将原数左移1位覆盖符号位。之后求出原数的exp值与frac值。当exp值全为1时，代表INF或NAN，返回原数，当符号位的值<=1时，将左移后的数右移两位并加上偏移量，否则阶码减1，最后或上符号位返回。

```
unsigned float_half(unsigned uf) {
    unsigned b = (uf & 3) == 3;
    unsigned tmin = 0x80000000;
    unsigned p = 0x1000000, pm = 0xff000000;
    unsigned sign = uf & tmin;
    unsigned uft = uf << 1;
    unsigned exp = uft & pm;
    unsigned t = (uft >> 2) + b;
    if(exp == pm) return uf;
    else if(exp <= p) uft = t;
    else uft = (uft - p) >> 1;
    return sign | uft ;
}
```