知乎

《深入理解计算机系统/CSAPP》Data Lab

不上岸不...
西南交通大学

28 人赞同了该文章

《深入理解计算机系统/CSAPP》Data Lab
yezhem.com/index.php/archives/6/

目标

填写 `bits.c` 源文件中的代码，并且满足题目要求(操作符的限制情况) PS:若有错误和更好解法请告知

文件说明:

- bits.c:需要填写的源代码文件
- dlc:检测文件是否符合题目要求(查看操作数./dlc -e bits.c)
- btest:检测得分(需要make)
- btest -f func:检测函数func正确性

谜题1 - absVal

- 获取x的绝对值
- 示例: absVal(-1) = 1
- 说明：-TMax <= x <= TMAX
- 限制操作: ! ~ & ^ | + << >>
- 操作数量: 10
- 难度: 4

()

对x处理可以分为两种情况， 取反+1 ， 不变+0 。 众所周知，一个数取反可以异或 1 ，不变可以异或 0 。 当 x<0 时， x>>31 为 0xFFFFFFFF ， x^(x>>31) 即取反， (x>>31)&1 为 0x1 。

```
int absVal(int x) {
  return (x^(x>>31))+((x>>31)&1);
}
```

谜题2 - addOK

- 判断x+y是否溢出，溢出返回0，反之
- 示例：addOK(0x8000000,0x8000000) = 0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：20
- 难度：3

众所周知，正数与负数之和不会溢出。 符号相同，则判断两数之和的最高位与加数中的任一数最高位是否相同即可，若相同则为发生溢出，反之。

```
int addOK(int x, int y) {
  return (((x^y)>>31)|~(((x+y)^x)>>31))&1;
}
```

- 判断一个二进制数偶数位是否全为1
- 示例：allEvenBits(0xFFFFFFFD) = 0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：12
- 难度：2

若一个二进制数偶数位为 `1` ，奇数位为 `0` ，则这个数为 `0x55555555` 。先将 `x=x&0x55555555` ，将这个数奇数为变为 `0` ，之后 `x^0x55555555` 判断该数是否为 `0x55555555` 。构造 `mask=0x55555555` ， `mask = 0x55 | 0x55 << 8;mask = 0x5555 | 0x5555 << 16;` 。

```
int allEvenBits(int x) {
  int mask = 0x55 | 0x55 << 8;
  mask = mask | mask << 16;
  x = x & mask;
  return !(mask^x);
}
```

谜题4 - allOddBits

- 判断一个二进制数奇数位是否全为1
- 示例：allOddBits(0xAAAAAAAA) = 1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：12
- 难度：2

与上一谜题相同，不过这次需要构造的数为 `0xAAAAAAAA` 。

```
int allOddBits(int x) {
  int mask = 0xAA | 0xAA << 8;
  mask = mask | mask << 16;
  x = x & mask;
  return !(mask^x);
}
```

谜题5 - anyEvenBit

- 判断一个二进制数任意偶数位是否有1
- 示例：anyEvenBit(0xE) = 1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：12
- 难度：2

判断偶数位是否含有 `1` ，只需要将所有偶数位与 `1` 相与，奇数位与 `0` 相与。若结果为 `0` ，则偶数位没有 `1` ，反之。

构造 `0x55555555` 与上题相同。

```
int anyEvenBit(int x) {
  int mask = 0x55 | 0x55 << 8;
  mask = mask | mask << 16;
  return !!(x&mask);
}
```

谜题6 - anyOddBit

- 操作数量：12
- 难度：2

思路同上一题，谜题 5 ， anyEvenBit 。

```
int anyOddBit(int x) {
  int mask = 0xAA | 0xAA << 8;
  mask = mask | mask << 16;
  return !!(x&mask);
}
```

谜题7 - bang

- 不使用 ！计算 !x
- 示例：bang(3)=0
- 限制操作：~ & ^ | + << >>
- 操作数量：12
- 难度：4

利用 +0/-0 最高位为 0 ， 而负数最高位为 1 ， 将正数转换为负数判断。

```
int bang(int x) {
  return ~(x|(~x+1))>>31&1;
}
```

谜题8 - bitAnd

- 只使用 ~ 和 ！ 计算x&y
- 示例：bitAnd(6,5)=4
- 限制操作：~ |
- 操作数量：8
- 难度：1

德·摩根律

```
int bitAnd(int x,int y) {
    return ~(~x|~y);
}
```

谜题9 - bitCount

- 计算二进制数中1的个数
- 示例：bitCount(7) = 3
- 限制操作：! ~ & ^ | + << >>
- 操作数量：40
- 难度：4

从两位比特数入手，计算两位比特数中1的个数，就是高位与低位之和。

令一个二进制数为令表示一个二进制数区间内含有的个数示例以下加法计算均用二进制表示设有位二进制数

我们发现为了方便计算，应当每次都保证计算时两个数的位数相同。

先构造一个只有低位 f(1,1) 的数 a=x&0x1 ,和高位 f(2,2) 的数 b=(x>>1)&0x1 。

但我们发现可以同时计算 f(3,3)+f(4,4) 等数。

只需要将构造改为 a=x&0x55555555 ， b=(x>>1)&0x55555555 ， x=a+b 。

计算 f(1,2)+f(3,4) 依次类推， a=x&0x33333333 , b=(x>>2)&0x33333333 ， x=a+b 。

自底向上推到，不一一叙述。

```
int bitCount(int x) {
  int mask_1 = 0x55 << 8 | 0x55;
  int mask_2 = 0x33 << 8 | 0x33;
  int mask_4 = 0x0f << 8 | 0x0f;
  int mask_8 = 0xff << 16 | 0xff;
  int mask_16 = ~0 + (1 << 16);
  mask_1 |= mask_1 << 16;
  mask_2 |= mask_2 | mask_2 << 16;
  mask_4 |= mask_4 | mask_4 << 16;
  x = (x&mask_1) + ((x>>1)&mask_1);
  x = (x&mask_2) + ((x>>2)&mask_2);
  x = (x&mask_4) + ((x>>4)&mask_4);
  x = (x&mask_8) + ((x>>8)&mask_8);
  x = (x&mask_16) + ((x>>16)&mask_16);
  return x;
}
```

谜题10 - bitMask

- 产生一个在 [lowbit,highbit] 区间内全为 1 的数，其余位为 0
- 示例：bitMask(5,3) = 0x38
- 说明：0 <= lowbig <= 31，0 <= highbit <= 31，若low>high，返回0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：16
- 难度：3

将 0xFFFFFFFF 的高 [highbit,32] 位置为 0 ，低 [0,lowbit-1] 位置为 0 即可。

构造低位为 0 的数 (~0)<<lowbit ， 高位为 0 的数 (~0)+(1<<(highbit+1)) ，改为 (~0) + (1<<highbit<<1) 防止 highbit=32 时出现 undefined behavior 。

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

谜题11 - bitMatch

- 产生一个掩码，表示x和y中哪些位相等。只使用 ~ 和 &
- 示例：bitMatch(0x7,0xE) = 0x6
- 限制操作：~ &
- 操作数量：14
- 难度：1

XYbitMatch(X,Y)111001100010

根据真值表推导出表达式

```
}
```

谜题12 - bitNor

- 使用 ~ 和 & 实现 ~(x|y)
- 示例：bitNor(0x6,0x5) = 0xFFFFFFF8
- 限制操作：~ &
- 操作数量：8
- 难度：1

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

谜题13 - bitOr

- 使用 ~ 和 & 实现 x|y
- 示例：bitNor(0x6,0x5) = 0x7
- 限制操作：~ &
- 操作数量：8
- 难度：1

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

谜题14 - bitParity

- 若x中含有奇数个0返回1，反之
- 示例：bitParity(5) = 0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：20
- 难度：4

偶数之差为偶数，偶数与奇数只差为奇数。所以 32 位二进制数中 1 和 0 的个数，奇偶性相同。

将 32 位二进制中所有数字进行异或计算。若有偶数个 1 则异或结果为 0 ，反之。

使用如下公式，答案放在低位。每次可计算一半数字。

令二进制数

```
int bitParity(int x) {
  x^=x>>16;
  x^=x>>8;
  x^=x>>4;
  x^=x>>2;
  x^=x>>1;
  return x&1;
}
```

- 限制操作：! ~ & ^ | + << >>
- 操作数量：45
- 难度： 4

类似谜题 9 ， `bitCount` ，只有计算部分不同。采用二分的思想，若交换 32 位数，则假定高 16 位和低 16 位已经逆序。此时只需要将高低 16 位交换即可。 `((x>>16)&mask1)|((x<<16)&mask2)` ，掩码保证这个数高 16 位必须为 `0` ，低 16 位为 `1` 。则两掩码的关系为 `mask2 = mask1 << 16. mask1 = 0x0000FFFF` 。

如何保证高 16 位和低 16 位已经是逆序，同样处理方法，需要 16 位数中高 8 位和低 8 位已经逆序。自底向上依次求解。

```
int bitReverse(int x) {
        int bitReverse(int x) {
        int mask_1 = 0x55 << 8 | 0x55;
        int mask_2 = 0x33 << 8 | 0x33;
        int mask_4 = 0x0f << 8 | 0x0f;
        int mask_16 = 0xff << 8 | 0xff;
        int mask_8 = mask_16 << 8 ^ mask_16;
    mask_1 |= mask_1 << 16;
    mask_2 |= mask_2 | mask_2 << 16;
    mask_4 |= mask_4 | mask_4 << 16;
    x = ((x&mask_1)<<1)|((x>>1)&mask_1);
    x = ((x&mask_2)<<2)|((x>>2)&mask_2);
    x = ((x&mask_4)<<4)|((x>>4)&mask_4);
    x = ((x&mask_8)<<8)|((x>>8)&mask_8);
    x = ((x&mask_16)<<16)|((x>>16)&mask_16);
    return x;
}
```

谜题16 - bitXor

- 使用 ~ 和 & 实现 x^y
- 示例：bitNor(0x4,0x5) = 0x1
- 限制操作：~ &
- 操作数量：14
- 难度：1

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

谜题17 - byteSwap

- 交换第n，m字节
- 示例：byteSwap(0x12345678,1,3) = 0x56341278
- 说明：0 <= n <= 3, 0 <= m <= 3
- 限制操作：! ~ & ^ | + << >>
- 操作数量：25
- 难度：2

创建掩码分别获取第m，n字节的数字。 `mask = 0xff << (n<<3)` ， `m = ((mask & x) >> (n<<3))&0xff`

```
int byteSwap(int x, int n, int m) {
  int m_n = 0xff << (n<<3);
```

```
}
```

谜题18 - conditional

- 实现 x?y:z
- 示例：conditional(2,4,5) = 4
- 限制操作：! ~ & ^ | + << >>
- 操作数量：16
- 难度：3计算 x 逻辑右移 n 位

若 x 的取值为 0x00000000 或 0xFFFFFFFF 。则答案为（x&y)|(~x&z）。

若 x! =0 时，将 x=0xFFFFFFFF 。 x = (!!x)<<31>>31 。

```
int conditional(int x, int y, int z) {
  x = (!!x)<<31>>31;
  return (y&x)|(z&~x);
}
```

谜题19 - copyLSB

- 将二进制数所有位的数置为最低位数
- 示例：copyLSB(6) = 0x00000000
- 限制操作：! ~ & ^ | + << >>
- 操作数量：5
- 难度：2

使用算数右移来拓展最低位。需将最低位放在最高位 x<<31 。

```
int copyLSB(int x) {
  return x<<31>>31;
}
```

谜题20 - distinctNegation

- 判断 x != -x ，满足返回1，反之。
- 限制操作：! ~ & ^ | +
- 操作数量：5
- 难度：2

判断 ~x+1 != x ，若两数相等异或值为 0

```
int distinctNegation(int x) {
  return !!((~x+1)^x);
}
```

谜题21 - dividePower2

- 计算 x/(2^n)， divpwr2
- 说明：0 <= n <= 30
- 示例：dividePower2(-33,4) = -2
- 限制操作：! ~ & ^ | + << >>
- 操作数量：15

令二进制数

右移最高位补齐与 b31 相同，若小数部分大于 0 则需要在 bn 位加一修正。

使用n位1的二进制数相加，若在 0 到 n-1 的任意位上有1将会产生进位。

构造 [0,n-1] 区间全为 1 的数 (1<<n)+0 ，当 x<0 会加上这个修正值。

```
int dividePower2(int x, int n) {
  return (x+(x>>31&((1<<n)+~0)))>>n;
}
```

谜题22 - evenBits

- 返回二进制数偶数位为1的数，即 0x55555555
- 限制操作：! ~ & ^ | + << >>
- 操作数量：8
- 难度：1

0x55555555 = 0x5555 << 16 | 0x5555 ，依次类推。

```
int evenBits(void) {
  int a = 0x55 << 8 | 0x55;
  return a << 16 | a;
}
```

谜题23 - ezThreeFourths

- 计算 x*3/4
- 示例：ezThreeFourths(11) = 8
- 限制操作：! ~ & ^ | + << >>
- 操作数量：12
- 难度：3

x*3 = x*2 + x = x >> 1 + x ， x/4 参考谜题 21 ， dividePower2 。

```
int ezThreeFourths(int x) {
  x = x + (x << 1);
  return (x+(x>>31&3))>>2;
}
```

谜题24 - fitsBits

- 判断 x 是否能用 n 位补码表示
- 说明：1 <= n <= 32
- 示例：fitsBits(5,3) = 0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：15
- 难度：2

判断其 [n+1,3] 区间上的数是否全为 1 ，或 0 即可。 x = x >> (b+~1+1) 。

```
int fitsBits(int x, int n) {
  x = x >> (n+~1+1);
  return !~x|!x;
```

- 判断 x 是否能用 16 位补码表示
- 示例：fitsShort(33000) = 0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：8
- 难度：1

参考上一题，谜题 24， `fitsShort` 。

```
int fitsShort(int x) {
  x = x >> 0xf;
  return !~x|!x;
}
```

谜题26 - floatAbsVal

- 返回浮点数 f 的绝对值
- 说明：若参数为NaN，返回该参数
- 限制操作：所有整数相关操作，||，&&，if，while
- 操作数量：10
- 难度：2

判断浮点数是否为 `NaN` ，是返回，否则，将最高位（符号位）置为 `0` 返回。

判断 `exp` 段是否全为 `1` ， `(uf&0x7f800000) >> 23 == 0xff` 。

判断 `frac` 段是否全为 `0` ， `uf << 9 != 0` 。

```
unsigned floatAbsVal(unsigned uf) {
  if((uf&0x7f800000)>>23 == 255 && uf<<9) return uf;
  return uf & 0x7fffffff;
}
```

谜题27 - floatFloat2Int

- 将浮点数转换为有符号整数， `float_f2i`
- 说明：超出整形范围(NaN和无穷大)返回0x80000000
- 限制操作：所有整数相关操作 || && if while
- 操作数量：30
- 难度：4

先将浮点数分成三段，符号部分 `s_ = uf>>31` ，指数大小 `exp_ = ((uf&0x7f800000)>>23)-127` ，获取小数部分，并补上浮点数缺省的 `1` ， `frac_ = (uf&0x007fffff)|0x00800000` 。

处理特殊情况全为 `0` 是返回 `0` ，若指数大于 `31` ，整数无法表示溢出返回 `0x80000000` 。若指数小于 `0` ，该数 `0<x<1` 返回 `0` 。

若指数部分大于 23 则将小数部分向左移动 `frac_ <<= (exp_ - 23)` ， `exp_` 代表指数大小。

若指数部分小于 23 则将小数部分向右移动 `frac_ >>= (23 - exp_)` ， `exp_` 代表指数大小。

考虑最后符号，正数转换为负数不会产生溢出。若 `frac_` 为正数，则根据 `s_` 调整正负输出即可。

若 `frac_` 为负数，唯一正确情况为 `0x80000000` 。

```
  if(!(uf&0x7fffffff)) return 0;

  if(exp_ > 31) return 0x80000000;
  if(exp_ < 0) return 0;

  if(exp_ > 23) frac_ <<= (exp_-23);
  else frac_ >>= (23-exp_);

  if(!((frac_>>31)^s_)) return frac_;
  else if(frac_>>31) return 0x80000000;
  else return ~frac_+1;
}
```

谜题28 - floatInt2Float

- 将有符号整数转换为浮点数， float_i2f
- 限制操作：所有整数相关操作 || && if while
- 操作数量：30
- 难度：4

与上一题相同，分三部分处理，获取符号位 s_ = x&0x80000000 ，若为负数 -x ，变为正数，则 0x80000000 为特殊情况分开处理，考虑特殊情况， 0x0 和 0x80000000 ，这两种情况直接返回 0 和 0xcf000000 。

获取最高位的 1 所在位置， while(!(x&(1<<n_))) n_--; 。

若 n_ <= 23 这个数需要向左移动到小数部分起始位置（将 1 放在第 23 位上）， if(n_<=23) x<<=(23-n_); 。

若 n_ > 23 这个数需要向右移动到小数部分起始位置（将 1 放在第 23 位上），这时候需要考虑移出部分的舍入问题，若移出部分大于 0.5 则向上舍入，若小于 0.5 则向下舍去，若等于 0.5 则向偶数舍入。

先将 >=0.5 情况等同考虑，向上舍入 x+=(1<<(n_-24)) 。若 ==0.5 时，舍入情况若为奇数，我们需要 -1 操作变为偶数，即将最低位的 1 变为 0 ， x&=(0xffffffff<<(n_-22)) ，若向上舍入时最高位产生了进位，还需要加上进位 if(x&(1<<n_)) ;else n_++; 。之后拼接浮点数即可。

```
unsigned floatInt2Float(int x) {
  int s_ = x&0x80000000;
  int n_ = 30;
  if(!x) return 0;
  if(x==0x80000000) return 0xcf000000;
  if(s_) x = ~x+1;
  while(!(x&(1<<n_))) n_--;
  if(n_<=23) x<<=(23-n_);
  else{
    x+=(1<<(n_-24));
    if(x<<(55-n_)) ;else x&=(0xffffffff<<(n_-22));
    if(x&(1<<n_))  ;else n_++;
    x >>= (n_-23);
  }
  x=x&0x007fffff;
  n_=(n_+127)<<23;
  return x|n_|s_;
}
```

谜题29 - floatIsEqual

- 操作数量：25
- 难度：2

直接使用 `uf == ug` 判断即可，但需要注意 `+0/-0` 还有 `NaN` 这两种特殊情况。

判断 `NaN` 时，指数段为 `0xff` ，小数段不全为 `0` ， `(uf&0x7fffffff) > 0x7f800000` 。

```
int floatIsEqual(unsigned uf, unsigned ug) {
  if(!(uf&0x7fffffff) && !(ug&0x7fffffff)) return 1;
  if((uf&0x7fffffff) > 0x7f800000) return 0;
  if((ug&0x7fffffff) > 0x7f800000) return 0;
  return uf == ug;
}
```

谜题30 - floatIsLess

- 判断浮点数 f<g
- 说明：+0和-0相等，若参数为NaN，返回0
- 限制操作：所有整数相关操作 || && if while
- 操作数量：30
- 难度：3

获取浮点数的三部分，符号位 `0x1&(uf>>31)` ，指数部分 `0xff&(uf>>23)` ，小数部分 `uf&0x7fffff` 。

和上题 29 ，谜题 `floatIsEqual` 相同，若为特殊情况 `+0/-0/NaN` ，返回 `0` 。依次比较符号位，指数位，小数位，即可。结果与符号位相异或可以取反，即根据正负判断条件相反。

```
int floatIsLess(unsigned uf, unsigned ug) {
  int uf_s = (uf>>31)&0x1 , ug_s = (ug>>31)&0x1;
  int uf_exp = (uf>>23)&0xff , ug_exp = (ug>>23)&0xff;
  int uf_frac = uf&0x007fffff, ug_frac = ug&0x007fffff;
  if(!(uf&0x7fffffff) && !(ug&0x7fffffff)) return 0;
  if((ug_exp==0xff&&ug_frac) || (uf_exp==0xff&&uf_frac)) return 0;
  if(uf_s^ug_s) return uf_s > ug_s;
  if(uf_exp^ug_exp) return (uf_exp<ug_exp)^(uf_s);
  if(uf_frac^ug_frac) return (uf_frac<ug_frac)^(uf_s);
  return 0;
}
```

谜题31 - floatNegate

- 计算浮点数 -f
- 说明：若参数为NaN，返回参数
- 限制操作：所有整数相关操作 || && if while
- 操作数量：10
- 难度：2

考虑特殊情况 `NaN` ，和谜题 29 相同。使用异或，将最高符号位取反。

```
unsigned floatNegate(unsigned uf) {
  if((uf&0x7fffffff) > 0x7f800000) return uf;
 return uf ^ 0x80000000;
}
```

谜题32 - floatPower2

- 限制操作：所有整数相关操作 || && if while
- 操作数量：30
- 难度：4

考虑特殊情况，指数 exp<-127 和 exp>128 ，分别返回 0 和 0x7f800000 。

```
unsigned floatPower2(int x) {
  int exp = x + 127;
  if(exp <= 0) return 0;
  if(exp >= 255) return 0x7f800000;
  return exp << 23;
}
```

谜题33 - floatScale1d2

- 计算浮点数 0.5*f
- 说明：若参数为NaN，返回参数
- 限制操作：所有整数相关操作 || && if while
- 操作数量：30
- 难度：4

考虑 NaN 和 INF 的特殊情况，即指数 exp==255 ， if((uf&0x7fffffff) >= 0x7f800000) ，返回参数。考虑为 +0/-0 的情况， if(!(uf&0x7fffffff)) ，返回 0 。

规格化小数 *0.5 若结果仍为规格化小数，这时候只需要指数 -1 ，其他部分不变即可。 (uf&0x807fffff)|(--exp_)<<23 。规格化小数 *0.5 ，可能结果变为非规格化。即 exp==0 或 exp==1 时的情况，这时候只处理小数部分，因为非规格化小数其指数部分为 0 ，右移一位表示 *0.5 ，考虑小数舍入，与谜题 28 相同，不过此时，只需要考虑最低两位数，舍入判断 if((uf&0x3) == 0x3) uf = uf + 0x2; 。只有当最低两位数为 11 时才需要向偶数进位舍入。

```
unsigned floatScale1d2(unsigned uf) {
  int exp_ = (uf&0x7f800000) >> 23;
  int s_ = uf&0x80000000;
  if((uf&0x7fffffff) >= 0x7f800000) return uf;
  if(exp_ > 1) return (uf&0x807fffff)|(--exp_)<<23;
  if((uf&0x3) == 0x3) uf = uf + 0x2;
  return ((uf>>1)&0x007fffff)|s_;
}
```

谜题34 - floatScale2

- 计算浮点数 2*f ， flaot_twice
- 说明：若参数为NaN，返回参数
- 限制操作：所有整数相关操作 || && if while
- 操作数量：30
- 难度：4

考虑原数为非规格化小数或 0 时，处理小数部分 if(exp_ == 0) return (uf<<1)|s_; 。若为 NaN 或 INF 时 if(exp_ == 255) return uf; ，直接返回。若结果，即指数加一 ++exp_ ,为 INF 时，保证其不为 NaN ，即小数部分全为 0 ， if(exp_ == 255) return 0x7f800000|s_; 。

```
unsigned floatScale2(unsigned uf) {
  int exp_ = (uf&0x7f800000)>>23;
  int s_ = uf&0x80000000;
  if(exp_ == 0) return (uf<<1)|s_;
```

```
}
```

谜题35 - floatScale64

- 计算浮点数 64*f
- 说明：若参数为NaN，返回参数
- 限制操作：所有整数相关操作 || && if while
- 操作数量：35
- 难度：4

与上一谜题 floatScale2 相同，不过对于非规格化小数处理不同，若直接左移 6 位，还需要处理其指数部分。非规格化小数时，若第 [22,17] 全为 0 ，即 uf&0x007e0000 == 0 ，最终结果仍然为非规格化小数。反之，结果为规格化小数，需要重新计算指数。步骤如下：取得 [22,17] 中 1 所在最高位 cnt_ ，即 while(!(uf&(1<<cnt_))) cnt_--; ，最终结果为左移 uf <<= (23-cnt_) 位，指数为 exp_ = 7-(23-n) 。

```
unsigned floatScale64(unsigned uf) {
  int exp_ = (uf&0x7f800000)>>23;
  int s_ = uf&0x80000000;
  int cnt_ = 22;
  if(exp_ == 0) {
    if(!(uf&0x007e0000)) return (uf<<6)|s_;
    while(!(uf&(1<<cnt_))) cnt_--;
    uf <<= (23-cnt_);
    return s_ | (uf&0x807fffff) | ((cnt_-16)<<23);
  }
  if(exp_ == 255) return uf;
  exp_+=6;
  if(exp_ >= 255) return 0x7f800000|s_;
  return (uf&0x807fffff)|(exp_<<23);
}
```

谜题36 - floatUnsigned2Float

- 无符号整数转化为浮点数， float_u2f
- 限制操作：所有整数相关操作 || && if while
- 操作数量：30
- 难度：4

参考谜题 28 ， floatInt2FLoat 。将符号位的处理修改即可。

```
unsigned floatUnsigned2Float(unsigned u) {
  int n_ = 31;
  if(!u) return 0;
  while(!(u&(1<<n_))) n_--;
  if(n_<=23) u<<=(23-n_);
  else{
    u+=(1<<(n_-24));
    if(u<<(55-n_)) ;else u&=(0xffffffff<<(n_-22));
    if(u&(1<<n_))  ;else n_++;
    u >>= (n_-23);
  }
  u=u&0x007fffff;
  n_=(n_+127)<<23;
  return u|n_;
}
```

- 限制操作：! ~ & ^ | + << >>
- 操作数量：6
- 难度：2

1byte=8bit ， 直接右移 n*8,n<<3 位取字节 n&0xff 。

```
int getByte(int x, int n) {
  return (x>>(n<<3))&0xff;
}
```

谜题38 - greatestBitPos

- 生成掩码，只保留二进制数 x 中为 1 的最高比特位
- 说明：若x=0,返回0
- 示例：greatestBitPos(96) = 0x40
- 限制操作：! ~ & ^ | << >>
- 操作数量：70
- 难度：4

转化问题为获取二进制数 x ， 为 1 的最高比特位为 n 。

令二进制数令函数若若二进制数第位是为的最高比特位满足并且若则最高位。采二分得到答案令若，否则若否则依次类推

判断 g(n+x) 是否为 1 ， 使用 & 操作， !!(x&((~0)<<(n+16)) 。最后若结果为 0 时，使用 & 操作再处理一次。

```
int greatestBitPos(int x) {
  int n = 0;
  n += ((!!(x&((~0)<<(n+16)))) << 4);
  n += ((!!(x&((~0)<<(n+8)))) << 3);
  n += ((!!(x&((~0)<<(n+4)))) << 2);
  n += ((!!(x&((~0)<<(n+2)))) << 1);
  n += (!!(x&((~0)<<(n+1))));
  return (1<<n)&x;
}
```

谜题39 - howManyBits

- 判断 x 需要多少为补码表示
- 示例：howManyBits(12) = 5
- 限制操作：! ~ & ^ | << >>
- 操作数量：90
- 难度：4

令二进制数若能用位补码表示，则若

所以我们只需要异或相邻的数 x^=(x<<1) ， 找出为 1 的最高位在哪一位就可以了。参考上一谜题 greatestBitPos 。

```
int howManyBits(int x) {
  int n = 0;
  x ^= (x<<1);
  n += ((!!(x&((~0)<<(n+16)))) << 4);
  n += ((!!(x&((~0)<<(n+8)))) << 3);
  n += ((!!(x&((~0)<<(n+4)))) << 2);
  n += ((!!(x&((~0)<<(n+2)))) << 1);
```

谜题40 - implication

- 判断命题逻辑 x–>y ， x 蕴含 y
- 示例：implication(1,1) = 1
- 限制操作：! ~ ^ |
- 操作数量：5
- 难度：2

```
int implication(int x, int y) {
  return (!x)|y;
}
```

谜题41 - intLog2

- 计算 floor(log2(x))
- 说明：x>0
- 示例：intLog2(16) = 4
- 限制操作：! ~ & ^ | + << >>
- 操作数量：90
- 难度：4

令二进制数且

问题转化为寻找 1 所在最高位,参考谜题 38 ， greatestBitPos 。

```
int intLog2(int x) {
  int n = 0;
  n += ((!!(x&((~0)<<(n+16)))) << 4);
  n += ((!!(x&((~0)<<(n+8)))) << 3);
  n += ((!!(x&((~0)<<(n+4)))) << 2);
  n += ((!!(x&((~0)<<(n+2)))) << 1);
  n += (!!(x&((~0)<<(n+1))));
  return n;
}
```

谜题42 - isAsciiDigit

- 判断 x 是否可以是表示数字的 Ascii 码
- 说明：0x30 <= x <= 0x39，返回1
- 示例：isAsciiDIgit(0x35) = 1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：15
- 难度：3

判断是否为 0x30 ， !!((x>>4)^3) 。判断低位是否为 0x0000~0x1001 ，即当第 3 位出现 1 时，第 1,2 位均不能为 0 。 x>>3&x>>1 | x>>3&x>>2 。

```
int isAsciiDigit(int x) {
  return !((((!!((x>>4)^3))|(x>>3&x>>1)|(x>>3&x>>2))&1);
}
```

- 限制操作：! ~ & ^ | + << >>
- 操作数量：5
- 难度：2

判断相等使用异或。

```
int isEqual(int x, int y) {
  return !(x^y);
}
```

谜题44 - isGreater

- 判断 x>y
- 示例：isGreater(4,5) = 0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：24
- 难度：3

以下情况会返回 1 。当 x>0,y<0 ， 或者当 x 和 y 符号相同时 mark_ = ~((x^y)>>31) ， 满足 x+~y+1>0 ， x!=y 时， equl_ = !!(x^y)=1 ， x+~y+1>=0 时， (~(x+~y+1))>>31 = 0xffffffff

```
int isGreater(int x, int y) {
  int sign_ = ((~x&y)>>31)&1;
  int mark_ = ~((x^y)>>31);
  int equl_ = !!(x^y);
  return sign_ | ((mark_)&(~(x+~y+1))>>31&equl_);
}
```

谜题45 - isLess

- 判断 x<y
- 示例：isGreater(4,5) = 1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：24
- 难度：3

思路与上一谜题 isGreater 相同。修改 x 和 y 的符号判断，以及 x+~y+1 的符号判断条件即可。小于 0 的数最高位只能为 1 ， 不用判断是否相等。

```
int isLess(int x, int y) {
  int sign_ = ((x&~y)>>31)&1;
  int mark_ = ~((x^y)>>31);
  return sign_ | (((mark_)&((x+~y+1))>>31)&1);
}
```

谜题46 - isLessOrEqual

- 判断 x<y
- 示例：isGreater(4,5) = 1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：24
- 难度：3

思路与上一谜题 isLess 相同，综合谜题 isGreater 修改相等判断条件即可。

```
  int mark_ = ~((x^y)>>31);
  int equl_ = !(x^y);
  return sign_ | ((((mark_)&((x+~y+1))>>31)|equl_)&1);
}
```

谜题47 - isNegative

- 判断 x<0
- 示例：isNegative(-1) = 1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：6
- 难度：2

判断最高位。若为 1 返回 1 ，反之。

```
int isNegative(int x) {
  return ((x>>31)&1);
}
```

谜题48 - isNonNegative

- 判断 x>=0
- 示例：isNegative(-1) = 0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：6
- 难度：2

思路同上谜题 isNegative 。

```
int isNonNegative(int x) {
  return (~(x>>31)&1);
}
```

谜题49 - isNonZero

- 不使用 ！操作，判断 x==0
- 示例：isNonZero(3) = 1
- 限制操作：~ & ^ | + << >>
- 操作数量：10
- 难度：4

谜题 7 解法相同， bang ，取反即可。

```
int isNonZero(int x) {
  return (x|(~x+1))>>31&1;
}
```

谜题50 - isNotEqual

- 判断 x!=y
- 示例：isNotEqual(5,5) = 0
- 限制操作：! ~ & ^ | + << >>

```
int isNotEqual(int x, int y) {
  return !!(x^y);
}
```

谜题51 - isPallindrome

- 判断二进制数 x，比特串是否为镜像
- 示例：isPallindrome(0x01234567E6AC2480) = 1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：45
- 难度：4

先通过逆序低 16 位，再与高 16 比较，即可以知道是否是镜像。逆序过程参考谜题 15，`bitReverse` 。

```
int isPallindrome(int x) {
  int mask_1 = 0x55 << 8 | 0x55;
  int mask_2 = 0x33 << 8 | 0x33;
  int mask_4 = 0x0f << 8 | 0x0f;
  int mask_8 = 0xff;
  int mask_16 = 0xff << 8 | 0xff;
  int tmp_=x;

  tmp_ = ((tmp_&mask_1)<<1)|((tmp_>>1)&mask_1);
  tmp_ = ((tmp_&mask_2)<<2)|((tmp_>>2)&mask_2);
  tmp_ = ((tmp_&mask_4)<<4)|((tmp_>>4)&mask_4);
  tmp_ = ((tmp_&mask_8)<<8)|((tmp_>>8)&mask_8);
  tmp_&=mask_16;
  x=(x>>16)&mask_16;
  return !(tmp_^x);
}
```

谜题52 - isPositive

- 判断 x>0
- 示例：isPositive(-1) = 0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：8
- 难度：2

判断最高位即可，对于 `0` 需要特殊处理，使用 `!!x` 得到，若 `x=0` 这该值为 `0`，否则为 `1` 。

```
int isPositive(int x) {
  return (((~x)>>31)&(!!x));
}
```

谜题53 - isPower2

- 判断 x==(2^n)
- 说明：没有任何负数对上述等式成立
- 示例：isPower2(5)=0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：20
- 难度：4

与正数。我们发现若
个性质，再排除特殊情

```
int isPower2(int x) {
  int ret = ((!(x&(x+~0))) & ((~(x>>31)&(!!x))));
  return ret;
}
```

谜题54 - isTmax

- 判断 x==Tmax
- 限制操作：! ~ & ^ | +
- 操作数量：10
- 难度：1

判断相等使用异或操作。 Tmax 满足 tmax == ~(tmax+1) ，排除同样满足条件的 0xffffffff 。

```
int isTmax(int x) {
  return !((x^(~(x+1)))|(!(~x)));
}
```

谜题55 - isTmin

- 判断 x==Tmin
- 限制操作：! ~ & ^ | +
- 操作数量：10
- 难度：1

判断相等使用异或操作。 Tmin 满足 tmin == ~tmin+1 ，排除同样满足条件的 0x00000000 。

```
int isTmin(int x) {
  return !((x^(~x+1))|(!x));
}
```

谜题56 - isZero

- 判断 x==0
- 限制操作：! ~ & ^ | + << >>
- 操作数量：2
- 难度：1

```
int isZero(int x) {
  return !x;
}
```

谜题57 - leastBitPos

- 生成掩码，只保留二进制数 x 中为 1 的最低比特位用字节数 c 来代替 n 中的第 x 字节数
- 示例：leastBitPos(96) = 0x20
- 限制操作：! ~ & ^ | + << >>
- 操作数量：6
- 难度：2

使用谜题 53 的思路，假设二进制数 x 为 1 的最低位在第 n 位，将 x-1 时，则 [n-1,0] 变为全 1 ， [n+1,31] 位不变，这部分做异或操作就可变为 0 ，即 x^(x-1) ，这时候 [n,0] 会全部变为 1 ，所以做与操作。

```
}
```

谜题58 - leftBitCount

- 返回二进制数从高位到低位，连续为 1 的个数
- 示例：lieftBitCount(0xFFF0F0F0) = 12
- 限制操作：! ~ & ^ | + << >>
- 操作数量：50
- 难度：4

若二进制数 x>>n ，为 0xffffffff ，那么可以说明其 [n,31] 位上的数为全 1 ，只要找出这个最大的数 n ，本题答案即为 31-n=31+~n+1 ，运用类似于题目 41 的技巧来找出这位，若 n=0 时有两种情况， 0xffffffff 和 0xfffffffe ，需要区别。

```
int leftBitCount(int x) {
  int cnt = 0;
  int off = 1&(!(~x));
  cnt += (!!(~(x>>16)))<<4;
  cnt += (!!(~(x>>(cnt+8))))<<3;
  cnt += (!!(~(x>>(cnt+4))))<<2;
  cnt += (!!(~(x>>(cnt+2))))<<1;
  cnt += (!!(~(x>>(cnt+1))));
  return 32+~cnt+off;
}
```

谜题59 - logicalNeg

- 不使用 ！计算 !x
- 示例：bang(3)=0
- 限制操作：~ & ^ | + << >>
- 操作数量：12
- 难度：4

与谜题 7 重复。

```
int bang(int x) {
  return ~(x|(~x+1))>>31&1;
}
```

谜题60 - logicalShift

- 计算 x 逻辑右移 n 位
- 说明：0 <= n <= 31
- 示例：logicalShift(0x87654321,4) = 0x08765432
- 限制操作：! ~ & ^ | + << >>
- 操作数量：20
- 难度：3

默认右移为算数右移， x>>n ，后只需要将高 [31-n+1,31] 位变为 0 即可，构造掩码。 ~0+(1<<(32+~n)<<1) ，将 31-n 分开计算避免溢出，出现未定义行为。

```
int logicalShift(int x, int n) {
  int mask_ = (~0)+(1<<(32+~n)<<1);
  return (x>>n)&mask_;
}
```

- 返回 -1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：2
- 难度：1

返回 `0xffffffff` ，即 `~0` 。

```
int minusOne(void) {
  return ~0;
}
```

谜题62 - multFiveEighths

- 计算 `x*5/8`
- 示例：multFiveEighths(77) = 48
- 限制操作：! ~ & ^ | + << >>
- 操作数量：12
- 难度：3

与谜题 23 为相同问题。

```
int multFiveEighths(int x) {
  x = x + (x<<2);
  return (x+(x>>31&7))>>3;
}
```

谜题63 - negate

- 计算 -x
- 示例：negate(1) = -1
- 限制操作：! ~ & ^ | + << >>
- 操作数量：5
- 难度：2

利用补码 `-x = ~x+1` 。

```
int negate(int x) {
  return ~x+1;
}
```

谜题64 - oddBits

- 返回奇数位上为 1 的二进制数
- 限制操作：! ~ & ^ | + << >>
- 操作数量：8
- 难度：2

即返回 `0xaaaaaaaa` ，利用 `0xaa` 移位以及或运算。

```
int oddBits(void) {
  int x = 0xaa;
  x |= x<<8;
  x |= x<<16;
  return x;
}
```

- 计算 x%(2^n)
- 说明：0 <= n <= 30
- 示例：remainderPower2(15,2) = 3
- 限制操作：! ~ & ^ | + << >>
- 操作数量：20
- 难度：3

保留这个数的第 [0,n-1] 位即可，负数转化为正数处理，最后再将结果转换回来。使用如下方法转化正负， x = ((~s_)&x)|(s_&(~x+1)); ，若 s_==0 即为正数，反之。

```
int remainderPower2(int x, int n) {
  int s_ = x>>31;
  x = ((~s_)&x)|(s_&(~x+1));
  x &= (~0+(1<<n));
  return ((~s_)&x)|(s_&(~x+1));
}
```

谜题66 - replaceByte

- 用字节数 c 来代替 n 中的第 x 字节数
- 说明：0 <= n <= 3，0 <= c <= 255
- 示例：replaceByte(0x12345678,1,0xab) = 0x1234ab78
- 限制操作：! ~ & ^ | + << >>
- 操作数量：10
- 难度：3

首先去除 x 的第 n 字节数，与 ~(0xff<<(n*8)) 相与，然后与 c<<(n*8) 相或。

```
int replaceByte(int x, int n, int c) {
  int mask_ = 0xff << (n<<3);
  c <<= (n<<3);
  return (x&(~mask_))|c;
}
```

谜题67 - rotateLeft

- 循环左移 n 位
- 说明：0 <= n <=31
- 示例：rotateLeft(0x87654321,4) = 0x76543218
- 限制操作：~ & ^ | + << >> !
- 操作数量：10
- 难度：3

构造低 n 位为 1 的掩码，用 & 获取移出位，用 | 补齐补充位。

```
int rotateLeft(int x,int n){
    int mask = (~0) + (1<<n);
    int r = (x>>(32+(~n)+1))&mask;
    return ((x<<n)&(~mask))|r;
}
```

谜题68 - rotateRight

- 循环右移 n 位

- 难度：3

同谜题67， `rotateLeft` 。

```
int rotateRight(int x,int n){
    int mask = (~0) + (1<<n);
    int offset = 32 + ~n + 1;
    int r = (x&mask)<<offset;
    return ((x>>n)&(~(mask<<offset)))|r;
}
```

谜题69 - satAdd

- 计算 x+y 并处理正/负溢出
- 说明：发生正溢出时，返回正数最大值。发生负溢出时，返回负数最大值。
- 示例：satAdd(0x40000000,0x40000000) = 0x7fffffff
- 限制操作：! ~ & ^ | + << >>
- 操作数量：30
- 难度：4

参考谜题2， `addOK` 来判断两数相加是否溢出，若溢出 `ok=0x00000000` ，否则 `ok=0xffffffff` 。利用 `(x+y)&ok` ，若未溢出则是正确答案，反之。下面考虑如何生成 `0x7fffffff` 和 `0x00000000` 。通过让 `x>>31` 来判断正负，正为 `0x00000000` ，若溢出则结果为 `0x7fffffff` ；负为 `0xffffffff` ，若溢出则结果为 `0x80000000` 。显而易见 `~((x>>31)+0x80000000)` 为溢出结果。此题 笔者发现题目错误，测试程序也存在错误，望纠正。

```
int satAdd(int x,int y){
    int ok = (((x^y)>>31)|(~((x+y)^x)>>31));
    return (ok&(x+y))+((~ok)&(~((x>>31)+(1<<31))));
}
```

编辑于 2021-01-04 23:38

深入理解计算机系统（书籍）

写下你的评论...

5 条评论 默认 时间

乌贼熊
很详细，受教了
2021-09-12 赞

peileiscott
请问执行dlc时出现permission denied是为什么
2021-03-12 赞

梦游家
dlc没有可执行权限，可以用chmod +x ./dlc来添加权限
2021-10-24 赞

猫猫虫
请教：谜题46 - isLessOrEqual这题是不是有点小问题.....
2021-02-16 赞

知乎

推荐阅读

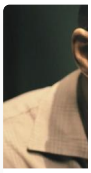
洛伊 陆景御 381-400

第381章 送你回去 “你这种男人还会对其他女人付出真心吗？”洛伊开口讥讽道。 封辰本来在串着蔬菜，听到她的话忽然抬起头问她，“在你心里我是什么样的男人？”洛伊想了想，看着他的眼睛...

桃子推荐

好妈妈育儿平台：孩子经常摇头是不是有病？这2种情况要小...

好妈妈育儿平台



经典高
你的心

灵性摆渡