大作业1报告
葛世杰
5140309294

1、bitAnd：从word x中取出第n个byte
```
int bitAnd(int x, int y) {
  return ~((~x)|(~y));
}
```
应用摩根律，~x|~y = ~（x&y），使用两次取反，得到&

2、getByte：从word x中取出第n个byte
```
int getByte(int x, int n) {
  return (x>>(n<<3)) & 0xff;
}
```
首先把第n个byte的内容右移n*8个单位到第1个byte。
再用& 0xff来取出第一个byte中的内容

3、logicalShift：逻辑右移
```
int logicalShift(int x, int n) {
  return ((x & 0x7fffffff) >> n) | (0x80000000 & x) >> n;
}
```
难点在于右移的补位部分要都补0.
分成两步，第一步把第31位到第1位的内容右移n个单位，第二步把符号位单独右移n个单位，加到第一步的结果中。

4、bitCount：计算1的个数
```
int bitCount(int x) {
  int n = 0;
  n = (x&0x55555555) + ((x>>1)&0x55555555);
  n = (n&0x33333333) + ((n>>2)&0x33333333);
  n = (n&0x0f0f0f0f) + ((n>>4)&0x0f0f0f0f);
  n = (n&0x00ff00ff) + ((n>>8)&0x00ff00ff);
  n = (n&0x0000ffff) + ((n>>16)&0x0000ffff);
  return n;
}
```
先计算x每两位中1的个数，并用对应的两队来存储这个个数。然后计算每4位1的个数，在用对应的4位进行存储。依次类推。
最后整合得到16位中1的个数，即x中的1的个数。

5、bang：表示！
```
int bang(int x) {
  x = (x >> 16) | x;
  x = (x >> 8) | x;
  x = (x >> 4) | x;
  x = (x >> 2) | x;
  x = (x >> 1) | x;
  return ~x & 0x00000001;
}
```
符号！的思想是如果word中有1，返回0；如果都是0，则返回1。
我的做法是把32位中任何一位上的1扩展到整个32位word中，再用第1位检测
第一步是把x右移16位再并x，这样可以把x的高16位的1添加到低16位
后面以此类推，分别右移8，4，2，1位再并x，把任何一位上存在的1扩展到32位
最后把x取反与1做和运算。如果x是1，返回0；x是0，返回1.

6、tmin：返回2补码所能表示的最小负数
```
int tmin(void) {
```

最后把n取反与1做和运算。如果n是1，返回0，n是0，返回1.

6、tmin：返回2补码所能表示的最小负数
```
int tmin(void) {
  return 0x80000000;
}
```
值为2的32次方

7、fitsBits：如果x能用n位表示，返回1；否则返回0
```
int fitsBits(int x, int n) {
  int shift= 32 + (~n + 1);
  return !(x^((x<<shift)>>shift));
}
```
先左移32-n位，在右移32-n位，即保留最后n位数。在与x异或
若两者相同表示x可被表示为一个n位整数

8、divpwr2：计算x除2的n次方
```
int divpwr2(int x, int n) {
  return ((x>>n) + (x>>31&0x00000001));
}
```
如果是正数，则直接右移n位即可
如果是负数，直接右移所得结果会比希望得到的结果小1
我的做法是取符号位加直接右移的结果得到最终答案。

9、negate：取反
```
int negate(int x) {
  return ~x+1;
}
```
2补码的步骤是取反加1

10、isPositive：判断是否是正数
```
int isPositive(int x) {
  return !((x >> 31) | (!x));
}
```
如果该数非0，则只需要看符号位，取符号位取反即可
如果该数为0，在本题中0与负数都返回0.用！x判断是否为0，再并符号位取反。

11、isLessOrEqual：判断两数大小
```
int isLessOrEqual(int x, int y) {
  x = ~x+1;
  return ((y+x)>>31 & 0x00000001)^1;
}
```
可以用y-x判断，将x取反与y相加，取得数的符号位判断

12、ilog2：计算log
```
int ilog2(int x) {
  int num=0;
  num=(!!(x>>16))<<4;
  num=num+((!!(x>>(num+8)))<<3);
  num=num+((!!(x>>(num+4)))<<2);
  num=num+((!!(x>>(num+2)))<<1);
  num=num+(!!(x>>(num+1)));
  num=num+(!!num)+(~0)+(!(1^x));
  return num;
}
```
本题我的思路是找到出现1的最高位
使用！！符号将x返回1或者0。
如果高16位存在1，则将结果加16：首先将x右移16位，判断是否存在1.如果存在1，则在结果的第5位上添加1.
如果高16位存在1，下一步是在24到32位寻找1；如果高16位不存在1，下一步是在8到16位寻找1.
以此类推，逐步缩小范围，直到找到出现1的最高位。

在8到16位寻找1.
以此类推，逐步缩小范围，直到找到出现1的最高位。

13、float_neg：浮点数取反

```
unsigned float_neg(unsigned uf) {
  unsigned num = uf ^ 0x80000000;
  unsigned nan = uf & 0x7fffffff;
  if(nan > 0x7f800000) return uf;
  return num;
}
```

如果浮点数是NaN，直接返回浮点数
否则将符号位取反，返回。

14、float_i2f：返回浮点数的二进制形式

```
unsigned float_i2f(int x) {
  unsigned shiftLeft=0;
  unsigned afterShift, tmp, flag;
  unsigned absX=x;
  unsigned sign=0;
  if (x==0) return 0;
  if (x<0)
  {
     sign=0x80000000;
     absX=-x;
  }
      afterShift=absX;
  while (1)
  {
     tmp=afterShift;
     afterShift<<=1;
     shiftLeft++;
     if (tmp & 0x80000000) break;
  }
  if ((afterShift & 0x01ff)>0x0100)
     flag=1;
  else if ((afterShift & 0x03ff)==0x0300)
     flag=1;
  else
     flag=0;

  return sign + (afterShift>>9) + ((159-shiftLeft)<<23) + flag;
}
```

如果x是0，返回0
如果x小于0，将符号位置位1，保存x的绝对值
对x的绝对值不断左移，直至最高位，保存右移次数
结果等于符号位，加移至最高位的x绝对值再移回frac位，加移动至指数位的右移次数

15、float_twice：浮点数乘2

```
unsigned float_twice(unsigned uf) {
  unsigned f = uf;
  if ((f & 0x7F800000) == 0)
  {
     f = ((f & 0x007FFFFF) << 1) | (0x80000000 & f);
  }
  else if ((f & 0x7F800000) != 0x7F800000)
  {
     f =f + 0x00800000;
  }
  return f;
}
```

分两种情况：
1、浮点数不是NaN，先将符号位置零，右移1位做乘2操作，再将符号

}
分两种情况：

1、 浮点数不是NaN。先将符号位置零，左移1位做乘2操作。再将符号位还原。

2、 浮点数是NaN。如果指数位是11111111而且不是无穷大，则判断为NaN，将指数位清零，返回。