Author: Roshan Adhikari
DAte: feb-01/2016

ByteAnd:idea->
First I converted given example into binary and did flip operation to every bit and converted another example(integer) into binary and flipped it. did the OR operation to these two flipped bits and again did flip operation to the result. I basically got this idea from simple algebraic operation like this:
a+b = -(-a)-(-b)
****

getByte:idea->
multiply by 8 i.e 2^3 has equivalent effort as shifting n left by 3. That means wanted byte is shifted to be least significant. I have shifted left to get rid of the bits on the left, then shift right to get rid of the bits on the right and move the wanted byte into the least significant spot.
****

bitcount: idea->
X>>1,res>>2,res>>4 operations helps in adding every two, four,eight bits respectively. counting the every bits of a byte and keeping the sum. 0x55 in hexadecimal is 1010101. It is kind of divide and conquer techinique. chunking the bytes and summing it up.
****

logical shift: idea->
1<<31 means moving 1 to the left most place.
x >> n shift n bits right, but if x is negative sign bit will be copied to right. & operations helps to do a masking.shifting using XOR to clear any bits on the left of the result that should be zero instead of one.
****

bang: idea->
The most significant bit will be 1 if MSB is zero.~ performs the bitwise not in x. Right shift by 31 place traverses the entire value which results in either 0 or 1111. This works for x or input = zero because positive zero and negative zero are equal.
****

tmin: idea->
moving 1 to the left most position and hence to sign bit which must be negative and hence gives the smallest two's complement integer.
****

fitsbits: idea->
checking whether if x is shifted by n bits then back by n bits again is still equal or not
****

divpwr2: idea->
Dividing by two can be obtained by shifting right in bitwise operations. Checked whether its multiple of two or not and whether it is negative or not as well. rounding up to the nearest multiple is done to get the result.
****

negate: idea->
idea derived from two's complement
****

ispositive: idea->
shifting right, and then taking complement and masking considering special cases like x=0 as well.
****

islessorequal: idea->
shifting the bits to the right and checking whether x is < o or y>=0 or not or x complement y is negative when both x and y have same sign.
****
****
float_neg: idea->
frac variable holds just the fractional value and later flipping the sign bit after checking whehter it is infinty or Not a Number.
****
*****
float i2f: idea->
checking for special cases like x equals to zero. IF x is less than zero then change the value of sign, abs variable. Use while loop to count the shift and changedbits variable where shift variable helps in keeping track of left shift operation.
****
****
float_twice: idea->check whether uf(floatnumber variable) assigned at the start is not-a-number(NaN) or not, after multiplying floatnumber variable by 2 as per instruction. If it is, then return floatnumber. left shifting is done by a bit. incrementing exp.