We **subtract** 127 from the unsigned value in the exponent field of a single precision floats to allow negative exponents.  
  
The IEEE-754 float is encoded as follows:  
bit[31] : sign bit. Note that this allows two zeros (a positive and a negative one).  
bit[30:23]: exponent, 8-bit unsigned value.  
bit[22:0]: fraction, 22-bit unsigned value wit implicit leading 1 (except when fraction is 0, where the implicit leading bit is also 0).  
  
The encoded value is:  
 (−1)*sign*×(1+*fraction*×2*exponent*−127)  
  
As you can see, we subtract 127 from the numeric value encoded by the exponent bits to get the true floating point exponent. The 8 bits of exponent allow us to encode unsigned numbers [0 .. 255[. The IEEE standard uses a midpoint bias to center this range at zero and allow negative numbers. The range of (exponent-127) is [-126 .. 127], which allows negative exponents up to -126.  
  
The encoding of IEEE-754 floats is messy and rich with corner cases, but is also somewhat elegant in that the mechanisms by which these data structures are compared, added, multiplied, etc. are reasonably efficient. Furthermore, the standard allows special quantities to be encoded (such as not-a-number (NaN), which indicates numeric error, and infinities), which are correctly added, compared, multiplied, etc. with numeric quantities and naturally arise as a result overflow and other exceptional cases.

Shift Right Arithmetic replicates the sign bit on the left. The new MSB after the SRA operation will be the same as the old MSB (before the SRA operation). This models a "divide by two" operation on twos-complement numbers, such that a positive number (say 15) turns into 7 after the SRA operation (the integer result after division by two) and a negative number (say -18) turns into -9 after the SRA operation:   
  
15: 00001111 7: 00000111   
-18: 11101110 -9: 11110111

BIASED REPRESENTATION

---------------------

an integer representation that skews the bit patterns so as to

look just like unsigned but actually represent negative numbers.

examples: given 4 bits, we BIAS values by 2\*\*3 (8)

TRUE VALUE to be represented 3

add in the bias +8

----

unsigned value 11

so the bit pattern of 3 in 4-bit biased-8 representation

will be 1011

going the other way, suppose we were given a

biased-8 representation as 0110

unsigned 0110 represents 6

subtract out the bias - 8

----

TRUE VALUE represented -2

this representation allows operations on the biased numbers

to be the same as for unsigned integers, but actually represents

both positive and negative values.

choosing a bias:

the bias chosen is most often based on the number of bits

available for representing an integer. To get an approx.

equal distribution of true values above and below 0,

the bias should be 2 \*\* (n-1) or (2\*\*(n-1)) - 1