In floating point you have:

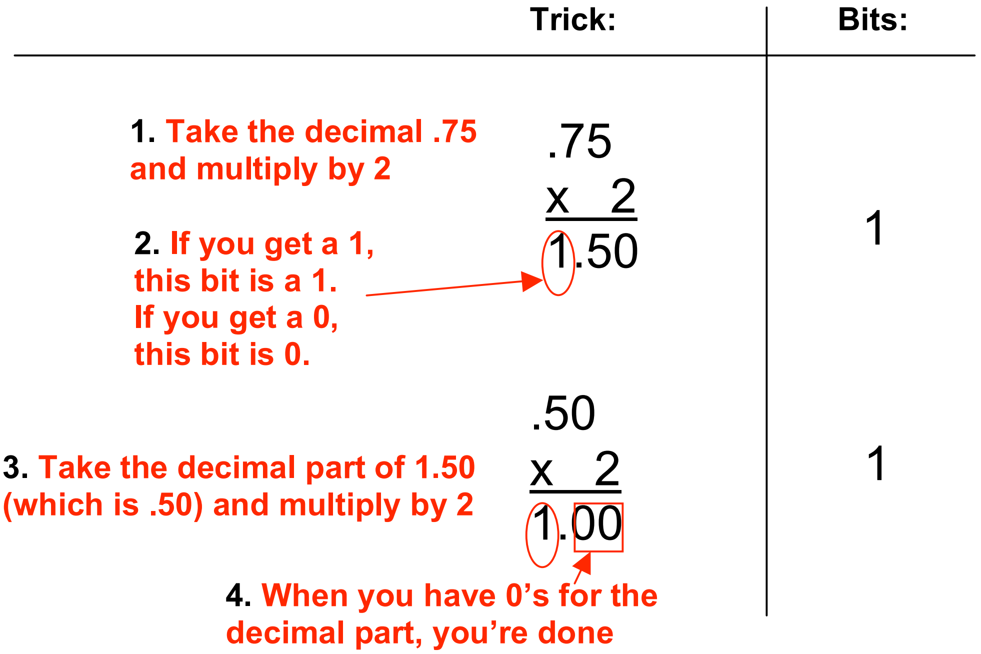
- a sign (+ or -), represented by 1 bit
- the exponent, represented by 8 bits
- fraction, represented by 23 bits.

1	8 bits	23 bits
s	biased exp.	fraction

So altogether you represent a floating point number in 32 bits.

Example number: 25.75 (From slides)

1. **Convert number into "hybrid" binary.**
   Convert 25 into binary => 11001 (not too bad of a conversion)
   Convert .75 into binary…here's a little trick:



1. Take the decimal .75 and multiply by 2

2. If you get a 1, this bit is a 1. If you get a 0, this bit is 0.

3. Take the decimal part of 1.50 (which is .50) and multiply by 2

4. When you have 0's for the decimal part, you're done

So now, .75 in binary is 11. You can fill in zeros after the "11", so it can be "110" or "11000". (Couple examples on this trick at the **end of this document**)

The total hybrid binary: 11001.110

2. **Floating Binary Point-** You want the number to be: **1.1001110.** So you must shift the decimal point to always have **only one 1** to the left of the decimal.
   Ex1: 1110.0001 => 1.1100001 **Ex2:** 00.111001001 => 1.11001001

To make the number 1.1001110, you have to move the decimal place **4 places to the left.** When you do so you need to keep track how many places you moved which can be represented as **2^4**. (Remember this is binary, that's why it's a 2^4 not 10^4 [the way to represent decimal shifts in decimal numbers])

Now its 1.1001110 x 2^4

3. **Floating point representation:**
   Now you need to fill in the fields.

   -Sign- The number initially was positive, so we represent the sign as a 0 for positive.

sign	exponent	fraction
0		

-Exponent- The value is 4. But the rule of filling this field is to **add 127 to the initial exponent value**. So 4 + 127 = 131.

131 is now the biased exponent, and just translate it into binary.
131 => 1000 0011, so we place this binary number into the field

sign	exponent	fraction
0	1000 0011	

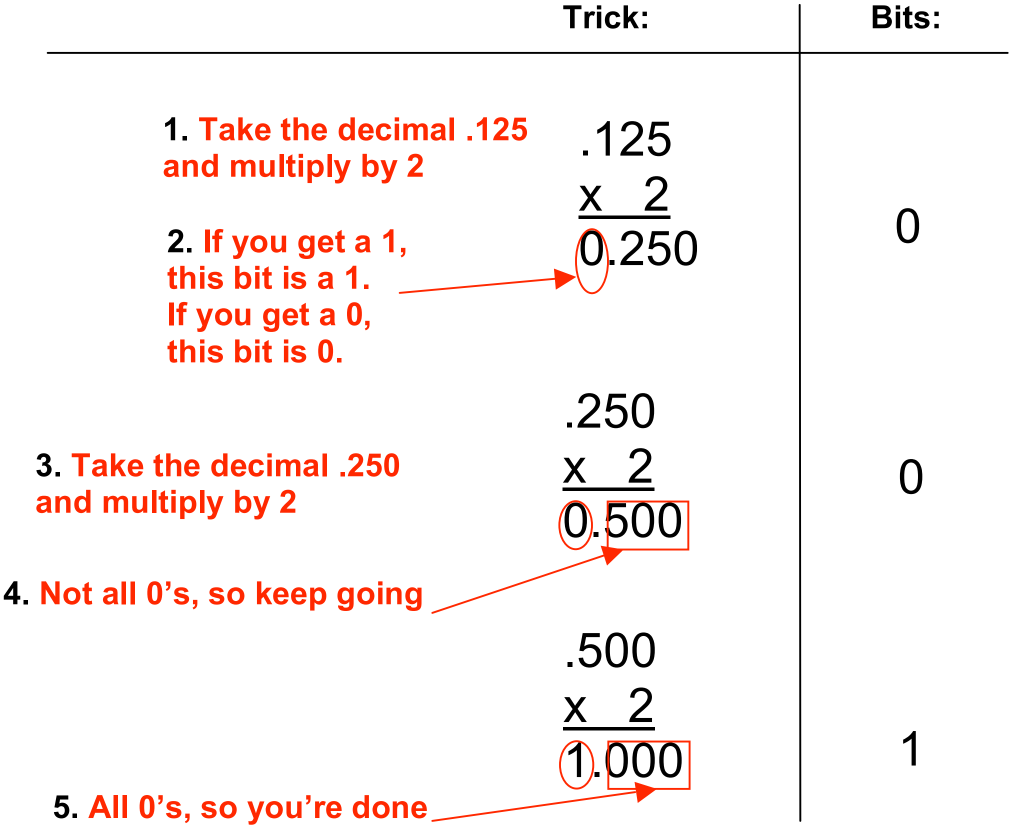
-Fraction- The fraction is all the numbers right of the decimal point. So in 1.1001110, our fraction is 1001110. Since this fraction has 23 bits to represent it, you just fill in the 0's. So it would be:

sign	exponent	fraction
0	1000 0011	1001 1100 0000 0000 0000 000

And so the floating point number representation is:
0 10000011 10011100000000000000000

Couple examples of the trick:

Example Number: 1.125



1. Take the decimal .125 and multiply by 2

2. If you get a 1, this bit is a 1. If you get a 0, this bit is 0.

3. Take the decimal .250 and multiply by 2

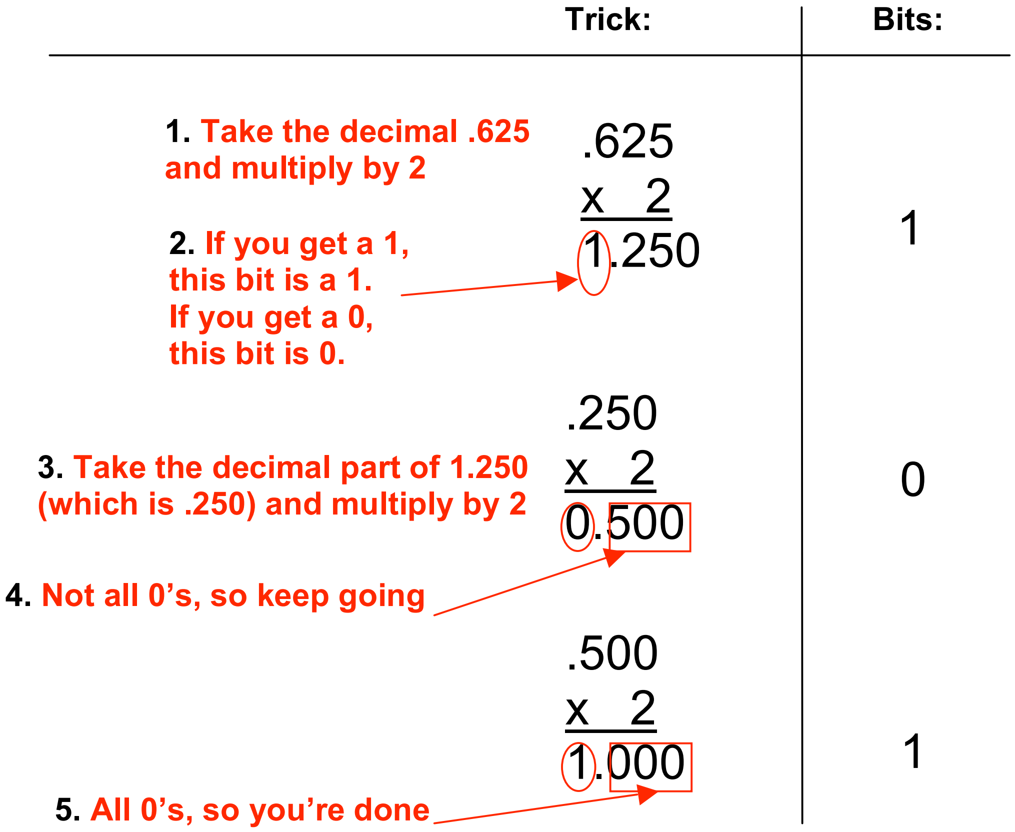
4. Not all 0's, so keep going

5. All 0's, so you're done

Trick:	Bits:
.125 x 2 0.250	0
.250 x 2 0.500	0
.500 x 2 1.000	1

Conversion of 1 => 1
Conversion of .125 => 001
"Hybrid" binary => 1.001

Example Number: 2.625



Trick: | Bits:

1. Take the decimal .625 and multiply by 2

.625
x 2
1.250 — 1

2. If you get a 1, this bit is a 1. If you get a 0, this bit is 0.

3. Take the decimal part of 1.250 (which is .250) and multiply by 2

.250
x 2
0.500 — 0

4. Not all 0's, so keep going

.500
x 2
1.000 — 1

5. All 0's, so you're done

Conversion of 2 => 10
Conversion of .625 => 101
"Hybrid" binary => 10.101