IEEE single precision floating point format (IEEE 754 floating point) is used commonly in computer systems to represent real numbers.

The format is 32 bits and with the bits having the following meaning:

0 23 22 31 30

- Sign bit
- Exponent
- Fraction or mantissa

0 = positive in bias 127
1 = negative

IEEE SP FP numbers are signed magnitude numbers, i.e., they have a sign bit. The numbers are also always in scientific notation, meaning they are normalized.

Since the numbers are always normalized (and are binary), then there will always be a leading 1 bit. Since it is always there, we can leave it out, giving us another bit to use for the fraction. This left-out bit is called the "hidden bit", we take this out when putting a number in to SP FP format and put it back when taking a number out of SP FP format.

Example 1

Given this SP FP number, find its decimal equivalent.

0x 43d0 000016

1. Break into 3 fields (sign, exponent, fraction) need to put in binary 1st

| 1 | 00 0011 1101 0000 0 | →
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign</td>
<td>Exponent</td>
</tr>
</tbody>
</table>

2. Take bias out of exponent, it is in bias 127.

\[
\begin{align*}
\frac{10000111}{00000000} &= \text{(128)} \quad \text{same as subtracting 127} \\
+ \frac{10000011}{00000000} &= \text{(1)}
\end{align*}
\]
000 1000₂ = 8₁₀

The true exponent of the number is thus 8₁₀.

3) Put the hidden bit back into the fraction.

1. 101 0

→ this was the fraction field of the 8 FP number.

4) Put together 2 & 3 with sign

+ 1.101 x 2⁸

5) Now take out of scientific notation

1.101 0000 x 2⁸

Hop the decimal over 8 times to get:

1 101 0000₂ = [416₁₀]

.: 0 x 4320 0000 is 416₁₀ in 8 FP format.

Example 2

Put -4.625₁₀ into 8 FP format.

1) Sign bit is 1 since negative.

2) Find binary representation for 4.625₁₀

a) What is 4₁₀ in binary? 4₁₀ = 100₂

b) What is 0.625₁₀ in binary? 0.625₁₀ = 0.101₂

<table>
<thead>
<tr>
<th>0.625</th>
<th>1.25</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0.125</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>0.0625</td>
<td>0.125</td>
<td>0</td>
</tr>
</tbody>
</table>

4.625₁₀ = 100. 101₂

3) Normalize the number.

100. 101₂ = 1.00101 x 2⁵

& hops
(4) Now that we have an exponent,
put it into bias 127 notation.

\[ 2 + 127 = 129_{10} \text{ which we would need to convert.} \]

or better \[ 00000010 \quad (2) \]

\[ +01111111 \quad (127) \]

\[ \underline{10000001}_{2} \]

or even better \[ 00000010 \quad (2) \]

\[ +10000000 \quad (128) \]

\[ \underline{10000001}_{2} \]

(5) Now put together (4) (3) (1) leaving out the hidden bit (the 1. in (3))

\[ \begin{array}{c|c|c|c}
\text{Sign} & \text{Exponent} & \text{Fraction} \\
\hline
0 & 00000001 & 001010 \\
\end{array} \]

to put in HEX group by 4.

\[ \begin{array}{c|c}
0 & 0000 \\
\hline
0 & 0100 \\
\hline
0 & 0000 \\
\hline
0 & 0000 \\
\hline
\end{array} \]

\[ 0 \times C0 94 0000 \text{ in SP FP.} \]

\[ \therefore -4.625_{10} = 0 \times C0 94 0000 \text{ in SP FP.} \]
See the review of single precision floating point for the format and conversion of numbers into & out of IEEE 32-bit.

Remember that 32-bit numbers are in signed magnitude notation thus what starts as addition may become subtraction.

**Example #1**

\[
\begin{align*}
0 \times 4780 \ 0000 \ & \text{positive} \# \\
-0 \times C560 \ 0000 \ & \text{negative} \#
\end{align*}
\]

```
÷ ÷ ⇒ ÷ ÷ turns into an addition
```

**Example #2**

\[
\begin{align*}
0 \times 4780 \ 0000 \ & \text{positive} \# \\
+0 \times C560 \ 0000 \ & \text{negative} \#
\end{align*}
\]

```
÷ ÷ ⇒ ÷ ÷ turns into a subtraction
```

Remember that the above 2 examples dealt with just the sign bit the magnitudes were unaffected. See the review on signed magnitude addition & subtraction for more examples.

**Example #3**

Perform the following operation on these two 32-bit numbers.

\[
\begin{align*}
0 \times 47FC \ 0000 \ & \text{positive} \#
-0 \times C560 \ 0000 \ & \text{negative} \#
\end{align*}
\]

1. Convert to binary & break into fields.

2. Resolve the sign & operation

```
÷ ÷ ⇒ ÷ ÷ so really an addition
```

3. Match exponents. Make smaller exponent the same as larger one. This will affect the smaller #’s fraction.

\[
\begin{align*}
1000 \ 1111 \ & \text{larger exponent} \\
-1000 \ 1010 \ & \text{smaller exponent} \\
0000 \ 0101 \ & \text{difference}
\end{align*}
\]
4. Shift smaller #'s fraction by difference, since by making exponent larger need to make fraction smaller so number stays the same. Be sure to put hidden bit back first.

\[ \begin{align*}
    0.000011 & \leq \text{smaller # fraction with hidden bit put back.} \\
    0.000011 & \leq \text{larger # fraction with hidden bit} \\
    \text{becomes} & \quad 0.000011
\end{align*} \]

5. Now that fractions decimal places match, since exponents are the same, we can do the operation.

\[ \begin{align*}
    1.1111100 + 0.0000111 & \leq \text{larger # fraction with hidden bit} \\
    \frac{1.1111100}{10.0000011} & \leq \text{larger # fraction after exponents matched.}
\end{align*} \]

6. Renormalize result if needed.

\[ \begin{align*}
    10.0000011 & \rightarrow 1.00000011 \\
    \text{this makes it smaller by a power of 2. Need to adjust exponent.}
\end{align*} \]

\[ \begin{align*}
    1.0001111 & \leq \text{old exponent} \\
    + \frac{1}{10010000} & \leq \text{4 of hops} \\
    1.0010000 & \leq \text{final exponent}
\end{align*} \]

7. Put 2, 5, & 6 together, removing hidden bit first.

\[ \begin{align*}
    0.10010000 \mid 00000110 \\n    \text{Now group by 4 to put in HEX} \\
    0 \times 48 \times 018000
\end{align*} \]