perplexity

where did the 1 before decimal is accounted then

Great question! This refers to the **"hidden" or "implicit leading 1"** in IEEE 754 floating-point representation.

Where Does the 1 Before the Decimal Go?

When you convert a number like **42.75** to normalized binary form for floating-point storage, you get:

- **101010.11_2**
- Normalized: **1.0101011×2^5**

The **"1."** before the decimal (the "leading 1") is always present in normalized numbers, and **it is NOT stored in the mantissa**. Instead:

IEEE 754 floating-point numbers assume this "1" is always there—so it is "hidden" or "implicit."

Why?

- Storing only the digits after the decimal ("fractional part") in the mantissa saves a bit for greater precision.
- The leading 1 is known to always exist (for normalized values), so hardware can restore it when reading the value.

How Mantissa is Stored:

- **Mantissa (or significand):** Only the bits after the decimal point are stored (right of the binary point).
- For **1.0101011**, mantissa stores: **0101011...** (pad with zeroes to fill the required bits: 23 for single, 52 for double precision)

So, **the "1" before the decimal is accounted for by hardware during calculations**—it's restored automatically whenever the floating-point number is used.

Example (IEEE 754 Double precision):

Field	What's Actually Stored
Sign bit	0
Exponent	Encodes (actual exponent + 1023 bias)
Mantissa	0101011... (bits after the decimal)

When reconstructing the value:

- Hardware “remembers” the hidden leading 1.
- To form the value: **1.[mantissa bits] × 2^{exponent - bias}**

Summary

- **The “1” before the decimal point is implicit and not stored in the mantissa.**
- Only the bits after the decimal are stored.
- When using the number, the floating-point system assumes the leading 1.

This trick is called the “implicit leading 1” and it increases the precision you can store in a fixed number of bits!

If you have a corner case (like denormal/subnormal numbers), let me know for more details.