Document Number: P2614R0

Date: 2022-07-06

Reply-to: Matthias Kretz <m.kretz@gsi.de>

Audience: SG6, LEWG

Target: C++26

Deprecate `NUMERIC_LIMITS::HAS_DENORM`

ABSTRACT

Since C is intent on obsoleting the `*_HAS_SUBNORM` macros, we should consider the analogue change in C++: the deprecation of `numeric_limits::has_denorm`. In general, compile-time constants that describe floating-point *behavior* are problematic, since behavior might change at run-time. Let's also deprecate `numeric_limits::has_denorm_loss` while we're at it.

CONTENTS

1   INTRODUCTION

`numeric_limits` has a member called `has_denorm` of type `float_denorm_style`:

```cpp
enum float_denorm_style {
  denorm_indeterminate = -1,
  denorm_absent        = 0,
  denorm_present       = 1
};
```

As Tydeman [N2993] states:

> There are several ways subnormals are "supported" in the field:
>
> - Partial support - hardware has encodings, but operations "fail".
>   - Operands are flushed to zero; results are kept.
>   - Operands are kept; results are flushed to zero.
>   - Some operations flush, others do not flush.
>   - Results suffer double rounding.
>   - Support can be changed at runtime (not by means in Stardard C).
> - Not at all. There are no hardware encodings of subnormals.
> - Full support as per IEEE-754.

Since hardware can change in future iterations, an implementation that does not want to risk an ABI break via `numeric_limits` will never set `has_denorm` to `denorm_absent` or `denorm_present`. The only ABI-safe sensible value is `denorm_indeterminate`. I.e. implementations cannot give a compile-time guarantee about *run-time behavior*.

The `has_denorm` value is not helping C++ users. Worst case, it is misleading users, resulting in incorrect assumptions and possibly breaking algorithms at some point.

1.1   HAS_DENORM_LOSS

The subsequent member in `numeric_limits`, `has_denorm_loss` is also calling for deprecation. Who can tell me the meaning of: "true if loss of accuracy is detected as a denormalization loss, rather than as an inexact result.[1]" cppreference.com explains [1]:

> No implementation of denormalization loss mechanism exists (accuracy loss is detected after rounding, as inexact result), and this option was removed in the 2008 revision of IEEE Std 754.

---

1 See ISO/IEC/IEEE 60559.

libstdc++, libc++, libCstd, and stlport4 define this constant as false for all floating-point types. Microsoft Visual Studio defines it as true for all floating-point types.

I don't own a IEEE 754 revision older than the 2008 revision, so it's hard to check. But at least the 2008 revision has no occurence of the word "loss" and no relevant occurence of "accuracy". The footnote's reference to the IEEE 754 standard is impossible to follow.

2   PROPOSED SOLUTION

The `has_denorm` and `has_denorm_loss` values should not be used.

A shallow code search[2] suggests that no code actually relies on `has_denorm`. However, a removal of the value would be a major compatiblity break. We can deprecate it, but without an actual intent of removal (since it would break too much). As an alternative to deprecation, we could change paragraph 46 "Meaningful for all floating-point types." to state that it's not even meaningful for floating-point types. Thus, user-defined types could still define a meaning for `has_denorm`.

The preference of SG6 after discussing [N2993] was deprecation of `has_denorm`.

`has_denorm_loss` should simply be deprecated (without actual intent of removal, though). The reference to IEEE 754 should be removed in any case.

3   WORDING

TBD.

4   SUGGESTED STRAW POLLS

Poll: Should the use of `numeric_limits::has_denorm` be discouraged via a change in the standard?

SF	F	N	A	SA

Poll: Deprecate `numeric_limits::has_denorm`?

SF	F	N	A	SA

---

2 `https://codesearch.isocpp.org/cgi-bin/cgi_ppsearch?q=has_denorm&search=Search`

Poll: Document `numeric_limits::has_denorm` as not meaningful for arithmetic types?

SF	F	N	A	SA

Poll: Deprecate `numeric_limits::has_denorm_loss`?

SF	F	N	A	SA

A   ACKNOWLEDGEMENTS

B   BIBLIOGRAPHY

[N2993]   Fred Tydeman. *N2993: Make *_HAS_SUBNORM be obsolescent*. ISO/IEC C Standards Committee Paper. 2022. URL: `https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2993.htm`.

[1]   User:Cubbi. *std::numeric_limits<T>::has_denorm_loss - cppreference.com*. 2021. URL: `https://en.cppreference.com/w/cpp/types/numeric_limits/has_denorm_loss` (visited on 07/05/2022).