Clarify requirements on extended floating point types

Doc. No: P3397R0
Contact: Hans Boehm (hboehm@google.com)
Audience: SG6
Date: Sept. 16, 2024
Target: C++26

Abstract

There is disagreement on what the requirements on extended floating-point types, such as `std::float32_t,` are. Are arithmetic operations on these types required to conform to ISO/IEC 60559? We should make sure we agree and fix the wording to make the intent clear enough to provide sufficient implementation guidance.

The problem

The C++ standard has traditionally been silent on floating-point accuracy requirements. More recently, P1467 added the optional "extended floating-point types" `float16_t`, `float32_t`, `float64_t`, `float128_t`, and `bfloat16_t` in [basic.extended.fp]. It is unclear whether these are intended to be similarly silent on accuracy requirements.

For example, paragraph 2 in [basic.extended.fp] states:

"If the implementation supports an extended floating-point type whose properties are specified by the ISO/IEC 60559 floating-point interchange format binary32, then the typedef-name `std::float32_t` is defined in the header <stdfloat> and names such a type, the macro `__STDCPP_FLOAT32_T__` is defined, and the floating-point literal suffixes `f32` and `F32` are supported."

The question is what "properties are specified by" means. The ISO/IEC 60559 floating point standard initially states its "purpose" (1.2) as:

"This standard provides a method for computation with floating-point numbers that will yield the same result whether the processing is done in hardware, software, or a combination of the two. The results of the computation will be identical, independent of implementation, given the same input data. Errors, and error conditions, in the mathematical processing will be reported in a consistent manner regardless of implementation."

and then goes on to state:

"A conforming operation shall return results correctly rounded for the applicable rounding direction for all operands in its domain." This implies that arithmetic operations and many `<cmath>` functions are required to produce a precisely-defined correct result; other "close approximations" are unacceptable.

On the other hand, in a small group that discussed this, there seemed to be a weak consensus that the C++ extended floating point 60559 constraint was only intended to apply to the format, and thus was really primarily a statement about reinterpret_cast(<*extended fp type*>).

However, I believe that around half of us (the half not involved in developing the wording?) initially read it differently. And SG6 voted at one point that "IEC (60)559 semantics apply to operations, in addition to representations", though with relatively small attendance.

IMO, the standard is not nearly clear enough on what was intended here. If `__STDCPP_FLOAT32_T__` is defined, then are arithmetic operations (and `<cmath>` functions>?) also required to abide by very strict ISO/IEC 60559 rounding rules?

With the "format only" interpretation, I believe that current compilers should generally define e.g. `float32_t` in the same way they define `float`. With the "operations too" interpretation, they clearly cannot without treating `float32_t` quite differently from `float`. As it stands, with the right options, g++ defines these types (and macros), but clang++ does not. Thus we suspect that implementers also disagree on the intended interpretation here.

This is of course a profound distinction. With the format-only interpretation, we expect that it is easy to implement types like `float32_t`, and its performance should be comparable to `float`. With the "operations too" interpretation, optimizations are significantly handicapped: `fma` instructions generally cannot be generated, and any optimizations that reassociate operations become invalid. Many sophisticated loop transformations do need to reassociate. Since these types were presumably introduced in large part to support ML programming, which is clearly performance-sensitive, that seems quite unfortunate.

In the "operations too" interpretation these types can also not easily be supported on hardware unless it supports IEEE gradual underflow. I believe that excludes some bfloat16 hardware implementations, among others. A complicating factor is that X86 and ARM hardware is often either IEEE conformant or not, depending on control word settings, which are not really known at compile time.

Our discussion also raised the very similar question of what `is_iec559` in [numeric.limits.members] means. There is related information in gcc bug 84949 .

Strawman wording proposals

I feel strongly that the current wording is unacceptably vague, but do not feel strongly about how to resolve that vagueness. For reasons given above, I lean weakly against requiring full 60559 operation conformance, and instead leaving support of that to other proposals.

A somewhat drastic revision of the above statement, that also attempts to correct some possible editorial weaknesses:

"The implementation defines either all of or none of

- the predefined macro `__STDCPP_FLOAT32_T_`
- the typedef name `std::float32_t` in <stdfloat>
- floating-point literal suffixes `f32` and F32

If these are defined. then `std::float32_t` is an extended floating-point type whose representation is specified by the ISO/IEC 60559 floating-point interchange format binary32. It is implementation defined whether the precision of arithmetic operations and mathematical functions also conforms to ISO/IEC 60559.

Note: an implementation that claims its operations also conform to ISO/IEC 60559 will usually need to forego a number of optimizations, including the generation of multiply-add instructions."

Corresponding changes would be made to the other optional extended floating-point types.

The above change is known to be controversial, mainly due to the use of "implementation defined". It is probably not as controversial as requiring full requiring full 60559 operation conformance, which is not explored here in detail.

A significantly different version that, in the opinion of some experts better reflects the original intent, would be to simply add something like the following to the beginning of [basic.extended.fp]:

"The ISO/IEC 60559 conformance requirements in this section apply only to the representation of types, and thus the results obtained from reinterpret_casts between extended floating-point and integral types, not to the precision of results of arithmetic operations on these types."

Acknowledgments

This grew out of a long email discussion in which Guy Davidson, Matthias Kretz, Jens Maurer, and David Olson also actively participated. They provided some of the information here. They are not listed as authors here only because it is unclear to me that we sufficiently agree on either the interpretation of the current standard, or what needs to be done to address it.