Proposal for C2x
WG14 N2775

Title:	Literal suffixes for bit-precise integers
Author, affiliation:	Aaron Ballman, Intel
	Melanie Blower, Intel
Date:	2021-07-13
Proposal category:	New features
Target audience:	C application programmers

Abstract: C23 will have bit-precise integer types. One of the salient properties of these types is that they do not undergo default integer promotions for performance reasons. Having a literal suffix will further reduce unnecessary conversions for expressions involving a bit-precise integer type and a literal.

Literal suffixes for bit-precise integers

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2775
Date: 2021-07-13

Summary of Changes

N2775

- Original proposal, split off from N2590

Introduction and Rationale

To support forming `_BitInt` literals, we propose adding two new integer literal suffixes, spelled `wb` and `uwb`, which designate a constant of type `_BitInt(N)` or `unsigned _BitInt(N)`, respectively, where `N` is calculated based on the given literal. Thus, `wb` results in a `_BitInt` of the smallest width for a signed representation of the literal, and `uwb` results in an `unsigned _BitInt` of the smallest width for an unsigned representation of the literal.

Within the preprocessor, if the constant value is too large to fit within the range of values supported by `[u]intmax_t`, the constant cannot appear within the controlling expression of a `#if` directive (6.10.1p7) but it will still form a valid integer constant suitable for use within an expression or initialization (6.4.8p4, 6.4.4p3). This can lead to a subtle surprise with code like:

```
#define FOO 0xFFFF…FFFFuwb // … is replaced by a lot of hex digits

_BitInt(…) I = FOO; // OK (… is replaced by the # of expected bits)

#if FOO // invalid constant expression; value too large for uintmax_t
#endif
```

The primary motivation for having a bit-precise literal suffix is to give programmers a way to avoid the performance penalty of implicit conversions in expressions involving literals without having to resort to cast operations that may obfuscate the expression. Consider this example:

```
_BitInt(7) DoMath(_BitInt(7) Value1, _BitInt(7) Value2) {
  return Value1 + Value2 * 2;
}
```

The arithmetic binary expressions require picking a common arithmetic type on which to perform the binary operation. Because `int` has greater width than `_BitInt(7)`, `Value2` will be implicitly converted to an `int` when performing the multiplication. For the same reason, `Value1` will also be converted to `int` when performing the addition. Finally, the result of the expression will be converted back to `_BitInt(7)` when returning from the function. While the programmer can use explicit casts to avoid the conversions, that approach becomes unpalatable as you add subexpressions also involving literals because the casts become distracting after a certain amount. Using an integer literal suffix is more expressive while being syntactically more succinct.

When polled about whether the committee is in favor of supporting integer literals of `_BitInt` type using the `xi` suffix at the Oct 2020 meeting, the results were 13/4/3 (consensus).

Suffix Design Choices

The literal suffix was part of the proposed bit-precise integer type feature in N2590 where it was specified as `xi` and `uxi`. Despite potential lexing difficulties with differentiating a bit-precise integer suffix from a hexadecimal literal, such as with `0xi`, this suffix made sense when the feature was called `_ExtInt`. However, the committee renamed the datatype to `_BitInt` and we no longer think that `xi` and `uxi` are a good choice given the similarity with hexadecimal literals. While `bi` and `ubi` would be natural choices for a suffix due to the similarity to the spelling of `_BitInt`, such a suffix is problematic due to conflicts with existing suffixes. Consider a literal like `0x3bi`, which is a valid imaginary literal whose value is `0x3b`.

Due to these concerns, we are using the specific-width length modifier adopted in N2680 as the basis for both a bit-precise length modifier (proposal forthcoming) and for the literal suffix. If it helps, you can remember the suffix as specifying a "**W**ide-enough **B**it-precise integer".

Proposed Straw Polls

We would like to see literal suffixes for bit-precise integer types added into C23. To that end, we would like to poll the following:

Does WG14 wish to adopt N2775 into C23?

In the event the previous poll does not gain consensus, we would like to poll:

Is WG14 in favor of supporting integer literals of `_BitInt` type using the `wb` suffix?

Proposed Wording

The wording proposed is a diff from WG14 N2596 with WG14 N2763 applied. Green text is new text, while ~~red~~ text is deleted text.

Modify 6.4.4.1p1:

integer-suffix:
unsigned-suffix long-suffix$_{opt}$
unsigned-suffix long-long-suffix
long-suffix unsigned-suffix$_{opt}$
long-long-suffix unsigned-suffix$_{opt}$
unsigned-suffix$_{opt}$ *bit-precise-int-suffix*
bit-precise-int-suffix unsigned-suffix$_{opt}$

bit-precise-int-suffix: one of **wb WB**

Modify 6.4.4.1p5 to add 2 rows to the bottom of the table: *Drafting note: mixing uppercase and lowercase letters for the suffix, like* `Wb`*, is not allowed; this is consistent with the long long suffix.*

wb or WB	**_BitInt(N) Where the width N is the smallest N greater than 1 which can accommodate the value and the sign bit.**	**_BitInt(N) Where the width N is the smallest N greater than 1 which can accommodate the value and the sign bit.**
Both u or U and wb or WB	**unsigned _BitInt(N) Where the width N is the smallest N**	**unsigned _BitInt(N) Where the width N is the smallest N**

	greater than 0 which can accommodate the value.	**greater than 0 which can accommodate the value.**

Add a new paragraph below 6.4.4.1p6:

EXAMPLE 1 The `wb` suffix results in an `_BitInt` that includes space for the sign bit even if the value of the constant is positive or was specified in hexadecimal or octal notation.

```
-3wb  /* Yields an _BitInt(3) that is then negated; two value
         bits, one sign bit */
-0x3wb /* Yields an _BitInt(3) that is then negated; two value
          bits, one sign bit */
3wb   /* Yields an _BitInt(3);  two value bits, one sign bit */
3uwb  /* Yields an unsigned _BitInt(2) */
-3uwb /* Yields an unsigned _BitInt(2) that is then negated,
         resulting in wrap-around */
```

Acknowledgements

I would like to recognize the following people for their help with this work: Erich Keane.

References

[N2590]
Adding a fundamental type for N-bit integers. Ballman, et al. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2590.pdf

[N2680]
Specific width length modifier. Seacord. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2680.pdf