Document number:	P1956R1
Revises:	P1956R0
Date:	2020-02-11
Project:	ISO JTC1/SC22/WG21: Programming Language C++
Audience:	LWG
Reply to:	Vincent Reverdy École Normale Supérieure, rue d'Ulm, Paris, France vince.rev@gmail.com

On the names of low-level bit manipulation functions

Abstract

We provide wording to rename the bit functions following National Body comments PL326, US327, GB332, US328, GB331 and following LEWG votes in Belfast 2019.

Contents

Introduction

In Belfast 2019, and following several national body comments including PL326, US327, GB332, US328, GB331 and SG6 suggestions, LEWG voted to rename some of the bit manipulation functions in order to avoid references to powers of two since manipulating bits should work in the future on bytes and machine words that are not numbers and for which numeric operations are not defined.

The changes voted in LEWG are the following:

- `ispow2` ⇒ `has_single_bit`
- `ceil2` ⇒ `bit_ceil`
- `floor2` ⇒ `bit_floor`
- `log2p1` ⇒ `bit_width`

Additionally it should be noted that:

- The behavior of the functions remain unchanged
- Concerns raised on the mailing list during Belfast 2019 about the new names not being compatible on hypothetical ternary or n-ary machines have been studied, and the new names are in fact compatible with ternary or n-ary machines as of the definition of bit provided in ISO/IEC 80000-13 item 13-9
- Following discussions in LWG, it has been decided to keep powers of two in the current description of functions and to reopen this discussion when the functions will be made compatible with `std::byte` and other non-number types

Note to the editor

Update the value of the `__cpp_lib_int_pow2` in [version.syn] Header `<version>` synopsis to reflect the date of approval of this proposal.

0.1 Bit manipulation [bit]

0.1.1 General [bit.general]

1 The header `<bit>` provides components to access, manipulate and process both individual bits and bit sequences.

0.1.2 Header `<bit>` synopsis [bit.syn]

```
namespace std {
  // 0.1.3, bit_cast
  template<class To, class From>
    constexpr To bit_cast(const From& from) noexcept;

  // 0.1.4, integral powers of 2
  template<class T>
    constexpr bool ~~ispow2~~has_single_bit(T x) noexcept;
  template<class T>
    constexpr T ~~ceil2~~bit_ceil(T x);
  template<class T>
    constexpr T ~~floor2~~bit_floor(T x) noexcept;
  template<class T>
    constexpr T ~~log2p1~~bit_width(T x) noexcept;

  // 0.1.5, rotating
  template<class T>
    [[nodiscard]] constexpr T rotl(T x, int s) noexcept;
  template<class T>
    [[nodiscard]] constexpr T rotr(T x, int s) noexcept;

  // 0.1.6, counting
  template<class T>
    constexpr int countl_zero(T x) noexcept;
  template<class T>
    constexpr int countl_one(T x) noexcept;
  template<class T>
    constexpr int countr_zero(T x) noexcept;
  template<class T>
    constexpr int countr_one(T x) noexcept;
  template<class T>
    constexpr int popcount(T x) noexcept;

  // 0.1.7, endian
  enum class endian {
    little = see below,
    big    = see below,
    native = see below
  };
}
```

0.1.3 Function template `bit_cast` [bit.cast]

1 [*Note:* Nothing to modify. — *end note*]

0.1.4 Integral powers of 2 [bit.pow.two]

```
template<class T>
  constexpr bool ~~ispow2~~has_single_bit(T x) noexcept;
```

1 *Constraints:* `T` is an unsigned integer type ([basic.fundamental]).

2 *Returns:* `true` if `x` is an integral power of two; `false` otherwise.

```
template<class T>
  constexpr T ceil2bit_ceil(T x);
```

3 Let N be the smallest power of 2 greater than or equal to `x`.

4 *Constraints:* `T` is an unsigned integer type ([basic.fundamental]).

5 *Expects:* N is representable as a value of type `T`.

6 *Returns:* N.

7 *Throws:* Nothing.

8 *Remarks:* A function call expression that violates the precondition in the *Expects:* element is not a core constant expression ([expr.const]).

```
template<class T>
  constexpr T floor2bit_floor(T x) noexcept;
```

9 *Constraints:* `T` is an unsigned integer type ([basic.fundamental]).

10 *Returns:* If `x == 0`, `0`; otherwise the maximal value `y` such that `ispow2`has_single_bit`(y)` is `true` and `y <= x`.

```
template<class T>
  constexpr T log2p1bit_width(T x) noexcept;
```

11 *Constraints:* `T` is an unsigned integer type ([basic.fundamental]).

12 *Returns:* If `x == 0`, `0`; otherwise one plus the base-2 logarithm of `x`, with any fractional part discarded.

0.1.5 Rotating [bit.rotate]

1 [*Note:* Nothing to modify. — *end note*]

0.1.6 Counting [bit.count]

1 [*Note:* Nothing to modify. — *end note*]

0.1.7 Endian [bit.endian]

1 [*Note:* Nothing to modify. — *end note*]

Acknowledgements

This work has been made possible thanks to the National Science Foundation through the awards CCF-1647432 and SI2-SSE-1642411, as well as French institutions École Normale Supérieure, INRIA, Paris Observatory, and PSL.