`subtract_with_carry_engine` Constructor Correction Proposal

Refining `n` and `k` Derivation for Accuracy and Compatibility

Document #:	D0000R1
Date:	2025-05-28
Project:	Programming Language C++
Audience:	Library Evolution Library
Revises:	D0000R0

Reply-to: Author Juan Lucas Rey <juanlucasrey@gmail.com>

1 Introduction

This proposal suggests a correction to the expressions used in computing `n` and `k` within the constructors of `subtract_with_carry_engine`. The change aims to improve compatibility and correctness, especially in edge cases such as `w = 32`.

2 Problem Description

In the current standard:

— In the **second constructor**, `n` is defined as:

```
std::size_t(w / 32) + 1
```

— In the **third constructor**, `k` is defined identically:

```
std::size_t(w / 32) + 1
```

This leads to over-allocation of state in some scenarios. Specifically, when `w = 32`, the value of `n` or `k` becomes 2. However, this is unnecessary because the highest index term contributes nothing to the final result:

$$z_{n-1} \cdot 2^{32 \cdot (n-1)} \mod m = 0$$

This indicates that the second term is redundant and contributes no value, resulting in inefficient and potentially confusing behavior.

3 Proposed Correction

We propose replacing the current expression:

```
std::size_t(w / 32) + 1
```

with either:

— A **backward-compatible form**:

```
std::size_t(w / 32) + std::size_t(bool(w % 32))
```

— Or, more concisely:

```
  std::size_t((w + 31) / 32)
```

This change ensures that:

— Only the minimal required number of 32-bit words are used to represent the `w`-bit state.
— Redundant state elements (which contribute nothing) are avoided.

4 Justification

4.1 1. Mathematical correctness

As noted, when `w = 32`, using two 32-bit words (`n = 2`) causes the higher-order term to be zero modulo `m`. Therefore, allocating more than one word is unnecessary in such cases.

4.2 2. Implementation convergence

Both GCC and Clang appear to have already adopted this refined interpretation. This behavior can be confirmed by instantiating:

```
std::subtract_with_carry_engine<std::uint_fast32_t, 32, 10, 24>
```

on either compiler, where the derived value of `k` or `n` reflects the corrected expression.

5 Impact on Existing Code

This change affects only the computed size of the internal state buffer. It does not alter any API or observable behavior when `w` is not a multiple of 32.

For `w = 32`, the corrected expression reduces redundant state storage and brings the behavior in line with both mathematical expectations and common compiler implementations.

6 Conclusion

This minor correction enhances efficiency, compatibility, and clarity in the `subtract_with_carry_engine` definition, without breaking existing code or altering engine semantics.