

Document Number: N3972

Date: 2026-09-26

Author: Robert C. Seacord

Reply-to: rcseacord@gmail.com

Audience: WG14

Target: C2Y

Proposed Clause: 6.3.3.1 Lvalues, arrays, and function designators, paragraph 2

Remove Itanium NaT-Motivated Undefined Behavior

Abstract

Subclause 6.3.3.1 paragraph 2 of the C2Y working draft (N3886) contains a sentence that makes it undefined behavior to read the value of an uninitialized object of automatic storage duration whose address was never taken. This text exists solely to accommodate the "Not a Thing" (NaT) bit found in general registers of the Itanium (IA-64) architecture. Itanium is no longer manufactured, no longer supported by any current compiler toolchain in active development, and has no successor architecture that reproduces this trapping behavior. This paper proposes removing the sentence, allowing such reads to fall back to the general, weaker rule that reading a value with an indeterminate representation produces an unspecified value in the ordinary case. This paper is aligned with the committee's ongoing "Slay Some Earthly Demons" effort to eliminate undefined behavior that no longer serves a real hardware or optimization purpose.

1. Introduction

Subclause 6.3.3.1 (Lvalues, arrays, and function designators) paragraph 2 currently reads, verbatim, in the N3886 working draft (emphasis added on the sentence proposed for removal):

Except when it is the operand of:

- the sizeof operator,*
- the typeof operators,*
- the unary & operator,*
- the ++ operator,*
- the -- operator,*
- the left operand of the . operator,*
- or, an assignment operator*

*an lvalue that does not have array type is converted to the value stored in the designated object (and is no longer an lvalue); this is called lvalue conversion. If the lvalue has qualified type, the value has the unqualified version of the type of the lvalue; additionally, if the lvalue has atomic type, the value has the non-atomic version of the type of the lvalue and the operation synchronizes with `memory_order_seq_cst` memory consistency; otherwise, the value has the type of the lvalue. **If the lvalue (that does not have array type) designates an object of automatic storage duration that never had its address taken, and that object is uninitialized (not declared with an initializer and no assignment to it has been performed prior to use), the behavior is undefined. An address of an object is taken by application of the address operator to an lvalue designating the object, if the object is an element of an array that was designated by an expression that is converted to a pointer as described below, or by being a member of a structure or union object whose address is taken.***

The bolded sentences are the subject of this proposal: the first is the undefined behavior being removed, and the second is a definition that exists solely to support it and is deleted as a consequential edit once it is removed (see Section 7).

2. Background: Why This Text Exists

This sentence traces back to C11:

1. Defect Report 338 ([DR #338](#)) was submitted on March 9, 2007, by Rich Peterson (HP) against committee draft ([N1124](#)). The defect noted that C99 trap-representation constraints technically precluded compilers from maintaining uninitialized auto scalars within registers on architectures featuring register-level trap bits, such as the Itanium "Not-a-Thing" (NaT) mechanism.
2. Initial proposed resolutions in 2007 attempted to modify the definition of "indeterminate value," though these adjustments were debated and ultimately declined.
3. The committee resolution in Spring 2008 formulated the specific sentence added to subclause 6.3.2.1 paragraph 2, designated as a "Change for C1X" (the draft subsequently ratified as C11).
4. Relevant drafting analysis is documented in WG14 [N1793](#), "Stability of indeterminate values in C11."

The modification originated as a defect resolution ([DR #338](#)) applied to the C11 draft rather than an independent proposal paper against C99. The corresponding document closest in scope is N1793.

Its purpose, as documented in committee discussion and the published Rationale for the C standard, is architecture-specific: the Itanium (IA-64) instruction set architecture includes a "Not a Thing" (NaT) bit shadowing each general hardware register. Speculative loads on Itanium can set the NaT bit of the destination hardware register instead of faulting immediately; a later,

non-speculative *use* of a hardware register with its NaT bit set triggers a fault at that point of use.

If a compiler allocates an automatic-storage-duration scalar to a hardware register and that variable is never explicitly initialized and never has its address taken (so the compiler is free to keep it purely in a hardware register, with no addressable memory backing), naively copying that register's value with an *indeterminate representation* could copy a NaT-flagged register and trigger a hardware fault merely from the act of reading it — even before the value is inspected or used arithmetically. Ordinary "indeterminate representation" wording (which otherwise permits reading a garbage-but-inert bit pattern as an unspecified value) was judged inadequate to cover this trap-on-read behavior because the problem and solution are not really related to trap representation as defined in the standard at all. Consequently, the committee defined undefined behavior specifically to give implementations targeting NaT-bearing hardware license not to guarantee a well-defined non-crashing read.

3. Current Status of the Itanium Architecture

Itanium is obsolete:

- Intel ended all sales of Itanium processors on July 29, 2021, and ended support in 2025; no Itanium hardware has been in production for several years.
- HP-UX and OpenVMS, the last major operating systems shipping on Itanium, have completed or announced end-of-life for Itanium-based platforms.
- No actively maintained C compiler continues to target Itanium as a supported architecture in new development; GCC and other toolchains have removed or deprecated IA-64 backends.
- No architecture that has succeeded Itanium in general-purpose, embedded, or high-performance computing (x86-64, AArch64, RISC-V, POWER, etc.) implements a register-level NaT-style trap-on-read mechanism that this sentence is needed to accommodate.

The rationale that originally motivated this undefined behavior therefore no longer applies to any implementation a conforming C2Y compiler is realistically expected to target.

4. Problems With Retaining the Current Wording

The sole documented justification for this undefined behavior is a hardware feature of a discontinued architecture. Retaining undefined behavior in the standard for hardware that no longer exists provides no benefit to any current or future implementation.

Undefined behavior is a well-recognized source of security vulnerabilities and unpredictable program behavior, and the committee has an active, multi-paper effort ("Slay Some Earthly Demons," e.g., the status papers "Ghosts and Demons: Undefined Behavior in C2Y") specifically aimed at removing undefined behavior that does not correspond to a genuine,

present-day implementation need. This sentence is a clear candidate for that effort: it is narrower, better-documented, and more clearly obsolete than many of the other cases under discussion.

Determining whether an automatic object has "never had its address taken" requires only local analysis within its enclosing function, as its address cannot be taken outside this lexical scope. Consequently, no interprocedural or translation-unit-wide analysis is necessary. Nevertheless, the distinction creating "fully undefined" versus "merely indeterminate" evaluations for seemingly identical uninitialized variables serves no general language design principle. It was introduced solely to accommodate the register-trapping mechanics of a now-obsolete architecture. Furthermore, developers frequently conflate this rule with general optimization latitude rather than an architectural workaround for a single retired hardware platform. Eliminating this provision simplifies the conceptual model of C: accessing an entity with an indeterminate representation yields either a non-value representation (governed by separate undefined behavior rules) or an unspecified value.

Removing this sentence does not force implementations to guarantee any particular value is read; it only removes the license to invoke full undefined behavior (including behaviors unbounded in scope, such as miscompilation of surrounding code) specifically to model a NaT-style trap. Ordinary indeterminate-representation rules elsewhere in the standard already allow the value read to be any unspecified value, and separate text already covers non-value representations for types that have them. No implementation loses the ability to defer register allocation decisions or to diagnose reads of uninitialized variables.

5. Impact on Existing Practice

No currently-maintained implementation relies on this specific undefined behavior to model Itanium NaT semantics, because no currently-maintained implementation targets Itanium. Compilers remain free to warn about (and even trap for debugging purposes, e.g., pattern-filling or sanitizer instrumentation) reads of uninitialized automatic variables under the general "indeterminate representation" and "non-value representation" provisions elsewhere in the standard; this proposal removes only the redundant, architecture-specific *undefined* behavior carve-out layered on top of those provisions.

Martin Uecker has written a (not upstream ready) patch to detect local use of this undefined behavior and this sanitizer relies on this UB being preserved for hardening purposes.

6. Relationship to Other Proposals

This proposal is a small, targeted contribution to the broader "Slay Some Earthly Demons" undefined-behavior-reduction effort (Uecker, Coates, Svoboda, and others) tracked in status papers such as "Ghosts and Demons: Undefined Behavior in C2Y." It is independent of and does not conflict with `_Optional`, contracts, provenance-aware memory object model (TS

6010), or other in-flight C2Y proposals; it may be adopted on its own merits regardless of the disposition of those other papers.

7. Proposed Wording

Changes are relative to subclause 6.3.3.1, paragraph 2, of the C2Y working draft (N3886).

Deleted text is struck through in red.

Except when it is the operand of:

- the sizeof operator,*
- the typeof operators,*
- the unary & operator,*
- the ++ operator,*
- the -- operator,*
- the left operand of the . operator,*
- or, an assignment operator*

an lvalue that does not have array type is converted to the value stored in the designated object (and is no longer an lvalue); this is called lvalue conversion. If the lvalue has qualified type, the value has the unqualified version of the type of the lvalue; additionally, if the lvalue has atomic type, the value has the non-atomic version of the type of the lvalue and the operation synchronizes with memory_order_seq_cst memory consistency; otherwise, the value has the type of the lvalue. ~~If the lvalue (that does not have array type) designates an object of automatic storage duration that never had its address taken, and that object is uninitialized (not declared with an initializer and no assignment to it has been performed prior to use), the behavior is undefined. An address of an object is taken by application of the address operator to an lvalue designating the object, if the object is an element of an array that was designated by an expression that is converted to a pointer as described below, or by being a member of a structure or union object whose address is taken.~~

Consequential deletion of the "address taken" definition sentence. The final sentence of the paragraph, defining what constitutes taking the address of an object, exists in N3886 solely to support the undefined-behavior sentence removed above; the author found no other normative use of this defined term elsewhere in the working draft (informal references to "taking the address" of an object elsewhere in the standard, e.g. in the footnote to 6.7.2 on the 'register' storage-class specifier and footnote 120 in subclause 6.7.4.1 paragraph 5, do not invoke this specific definition). Because this proposal removes the only rule that depends on it, the definition becomes vestigial and is deleted along with the undefined-behavior sentence, rather than left orphaned in the standard text.

Required consequential edit to Annex J.2 Subclause J.2 currently includes the following bullet regarding the subclause under discussion:

— An lvalue designating an object of automatic storage duration that could have been declared with the register storage class is used in a context that requires the value of the designated object, but the object is uninitialized. (6.3.3.1)

Adopting the proposed modifications necessitates deleting this item. Project editors should observe that this text—referencing the register specifier—already diverges from the normative phrasing in subclause 6.3.3.1 paragraph 2 ("never had its address taken"). Regardless of whether this paper passes, the committee ought to audit Annex J.2 for synchronization with the core standard.

No additional clauses require modification. Evaluations of uninitialized automatic entities remain subject to the indeterminate representation provisions of subclause 6.2.4 (alongside the representation rules in 3.25 and 3.26) and the non-value representation specifications in 6.2.6.1, both of which are left entirely unaltered.

8. Acknowledgments

Thanks to the "Slay Some Earthly Demons" contributors (Uecker, Coates, Svoboda) for establishing the framework and precedent for evaluating and retiring obsolete undefined behavior in the C standard. Thanks to Aaron Ballman and Martin Uecker for reviewing.

References

- ISO/IEC 9899:202y, N3886, Working Draft (pre-Ottawa), 2026-05-24.
- "Ghosts and Demons: Undefined Behavior in C2Y," N3861 / N3957.
- Intel, Itanium processor family end-of-life / end-of-support product change notifications.
- [Rationale for International Standard — Programming Language — C](#), discussion of 6.3.2.1 (Lvalues, arrays, and function designators) indeterminate-representation wording in prior editions.
- Uecker, M. and Gustedt, J., "Indeterminate Values and Trap Representations (updates N2772)," ISO/IEC JTC1/SC22/WG14 N2861, 2021-11-28; source of modern terminology for non-value and indeterminate representations in N3886.