XC Implementation-Defined Behavior

A conforming XC implementation is required to document its choice of behavior for all parts of the language specification that are designated implementation-defined. In the following section, all choices that depend on an externally determined application binary interface are listed as "determined by ABI," and are documented in the Application Binary Interface Specification (see XM-000967-PC).

- **The value of a multi-character constant (§1.5.2).**

  The value of a multi-character constant is the same as the value of its first character; all other characters are ignored.

- **Whether identical string literals are distinct (§1.6).**

  Identical string literals are not distinct; they are implemented in a single location in memory.

- **The available range of values that may be stored into a `char` and whether the value is signed (§3.2).**

  The size of `char` is 8 bits. Whether values stored in a `char` variable are signed or not is determined by the ABI.

- **The number of pins interfaced to a port and used for communicating with the environment; and the value of a port or clock not declared `extern` and not explicitly initialized (§3.2, §7.7).**

  The number of pins connected to a port for communicating with the environment is defined either by the explicit initializer for its declarator. If no initializer is provided, the compiler produces an error message.

- **The notional transfer type of a port, the notional counter type of a port, and the notional counter type of a timer (§3.2).**

  The notional types are determined by the ABI.

- **The value of an integer converted to a signed type such that its value cannot be represented in the new type (§5.2).**

  When any integer is converted to a signed type and its value cannot be represented in the new type, its value is truncated to the width of the new type and sign extended.

- **The handing of overflow, divide check, and other exceptions in expression evaluation (§6).**

  All conditions (divide by zero, mod zero and overflow of signed divide / mod) result in undefined behaviour.

- **The notion of alignment (§6.3.4).**

  An alignment of 2^n guarantees that the least significant n bits of the address in memory are 0. The specific alignment of the types is determined by the ABI.

- **The value and the type of the result of `sizeof` (§6.4.6).**

  The value of the result of the `sizeof` operator is determined by the ABI. The type of the result is `unsigned int`.

- **Attempted run-time division by zero (§6.6).**

  Attempted run-time division by zero results in a trap.

- **The extent to which suggestions made by using the `inline` function specifier are effective (§7.3).**

  The `inline` function specifier is taken as a hint to inline the function. The compiler tries to inline the function at all optimization levels above `-O0`.

- **The extent to which suggestions made by using the `register` storage class specifier are effective (§7.7.4).**

  The `register` storage class specifier causes the register allocator to try and place the variable in a register within the function. However, the allocator is not guaranteed to place it in a register.

- **The supported predicate functions for input operations (§8.3).**

  The set of supported predicate functions is documented in XM-000969-PC.

- **The meaning of inputs and outputs on ports (§8.3.2).**

  The inputs and outputs on ports map to in and out instructions on port resources, the behaviour of which is defined in the XS1 Ports Specification (see X1373).

- **The extent to which the underlying communication protocols are optimized for transaction communications (§8.9).**

  The communication protocols are determined by the ABI.

- **Whether a transaction is invalidated if a communication occurs such that the number of bytes output is unequal to the number of byte input, and the value communicated (§11).**

  This is determined by the ABI.

- **The behavior of an invalid operation (§12).**

  Except as described below, all invalid operations are either reported as compilation errors or cause a trap at run-time.

  - The behavior of an invalid master transaction statement is undefined; an invalid slave transaction always traps.
  - The `unsafe` pragma (see XM-000959-PC) can be used to disable specific safety checks, resulting in undefined behavior for invalid operations.