# Morello Supplement to the Arm C Language Extensions

## 01alpha

Date of Issue: 02 July 2021

arm

# 1 Preamble

## 1.1 Abstract

This document specifies the Arm C Language Extensions to enable C/C++ programmers to use the Morello architecture with minimal restrictions on source code portability.

## 1.2 Keywords

Predefined macros, built-in functions

## 1.3 Latest release and defects report

For the latest release of this document, see the ACLE project on GitHub.

Please report defects in this specification to the issue tracker page on GitHub.

## 1.4 License



## 1.5 About the license

As identified more fully in the License section, this project is licensed under CC-BY-SA-4.0 along with an additional patent license. The language in the additional patent license is largely identical to that in Apache-2.0 (specifically, Section 3 of Apache-2.0 as reflected at https://www.apache.org/licenses/LICENSE-2.0) with two exceptions.

First, several changes were made related to the defined terms so as to reflect the fact that such defined terms need to align with the terminology in CC-BY-SA-4.0 rather than Apache-2.0 (e.g., changing “Work” to “Licensed Material”).

Second, the defensive termination clause was changed such that the scope of defensive termination applies to “any licenses granted to You” (rather than “any patent licenses granted to You”). This change is intended to help maintain a healthy ecosystem by providing additional protection to the community against patent litigation claims.

## 1.6 Contributions

Contributions to this project are licensed under an inbound=outbound model such that any such contributions are licensed by the contributor under the same terms as those in the LICENSE file.

## 1.7 Trademark notice



## 1.8 Copyright



# 2 About This Document

## 2.1 Change Control

### *2.1.1 Current Status and Anticipated Changes*

The following support level definitions are used by the ACLE specifications:

**Release**

Arm considers this specification to have enough implementations, which have received sufficient testing, to verify that it is correct. The details of these criteria are dependent on the scale and complexity of the change over previous versions: small, simple changes might only require one implementation, but more complex changes require multiple independent implementations, which have been rigorously tested for cross-compatibility. Arm anticipates that future changes to this specification will be limited to typographical corrections, clarifications and compatible extensions.

**Beta**

Arm considers this specification to be complete, but existing implementations do not meet the requirements for confidence in its release quality. Arm may need to make incompatible changes if issues emerge from its implementation.

**Alpha**

The content of this specification is a draft, and Arm considers the likelihood of future incompatible changes to be significant.

All content in this document is at the **Alpha** quality level.

### *2.1.2 Change History*

| Issue | Date | Change |
|---|---|---|
| 00alpha | 30th September 2020 | Alpha release |
| 01alpha | 02 July 2021 | Open source release. NFCI. |

## 2.2 References

This document refers to, or is referred to by, the following documents.

| Ref | URL or other reference | Title |
|---|---|---|
| ACLE-morello | This document | Morello Supplement to the Arm C Language Extensions |
| ACLE | Document number: 101028 | Arm C Language Extensions |
| CHERI | UCAM-CL-TR-947, SSN 1476-2986 | CHERI C/C++ Programming Guide |

## 2.3 Terms & Abbreviations

**Capability**

The capability data type is an unforgeable token of authority which provides a foundation for fine grained memory protection and strong compartmentalisation.

**Permissions**

The permissions mask controls how the capability can be used - for example, by authorizing the loading and storing of data and/or capabilities.

**Deriving a capability**

A capability value CV2 is said to be derived from a capability value CV1 when CV2 is a copy of CV1 with optionally removed permissions and/or optionally narrowed bounds (base increased or limit reduced).

**Sealing a capability**

When a capability is sealed it cannot be modified or dereferenced, but it can be used to implement opaque pointer types.

# 3 Scope

The Morello Supplement to the Arm C Language Extensions highlights the language features added on top of the CHERI programming language to further exploit the Morello architecture. We recommend reading the CHERI Pure-Capability C/C++ Programming Guide as preliminary material.

# 4 Predefined macros

ACLE introduces several predefined macros that define how the C/C++ implementation uses the Morello architecture.

## 4.1 __ARM_FEATURE_C64

This macro indicates that the code is being compiled for the C64 ISA.

## 4.2 Capability Permissions

The following macros indicate capability permissions:

| Name | Value |
|---|---|
| `__ARM_CAP_PERMISSION_EXECUTIVE__` | 2 |
| `__ARM_CAP_PERMISSION_MUTABLE_LOAD__` | 64 |
| `__ARM_CAP_PERMISSION_COMPARTMENT_ID__` | 128 |
| `__ARM_CAP_PERMISSION_BRANCH_SEALED_PAIR__` | 256 |

Those can be used to form a bitmask that is acceptable for `cheri_perms_and()` and `cheri_perms_clear()`. The value of each macro corresponds to the permission bit as it appears in the architecture documentation.

## 4.3 Deviation from CHERI

The macro `__CHERI_CAP_PERMISSION_PERMIT_CCALL__` is not available on the Morello architecture.

# 5 Builtin functions

ACLE standardizes builtin functions to access the Morello architecture. These are the following:

## 5.1 Check subset and conditionally unseal or return null

```
void* __capability
__builtin_morello_subset_test_unseal_or_null(const void* __capability a,
                                             const void* __capability b)
```

Assuming two valid capabilities a and b, with the former being sealed and the latter being unsealed, if a can be derived from b, then it unseals a and returns it, otherwise it returns a null capability.

## 5.2 Check subset and conditionally unseal

```
void* __capability
__builtin_morello_chkssu(const void* __capability a,
                         const void* __capability b)
```

Assuming two valid capabilities a and b, with the former being sealed and the latter being unsealed, if a can be derived from b, then it unseals a and returns it, otherwise it just returns a.

## 5.3 Convert pointer to capability offset (zeroing form)

```
void* __capability
__builtin_morello_cvtz(const void* __capability a, size_t b)
```

If the specified offset b is zero, then it returns a null capability, otherwise it sets the offset of capability a to b and returns a. If capability a is sealed then the returned capability is marked invalid.