

Morello extensions to ELF for the Arm[®] 64-bit Architecture (AArch64)

2023Q3

Date of Issue: 6th October 2023

arm

1 Preamble

1.1 Morello alpha

This document is an alpha proposal for Morello extensions to ELF for AArch64.

1.2 Abstract

This document describes the use of the Morello extensions to the ELF binary file format in the Application Binary Interface (ABI) for the Arm 64-bit architecture.

1.3 Keywords

ELF, AArch64 ELF, Morello, C64, ...

1.4 Latest release and defects report

Please check [Application Binary Interface for the Arm® Architecture](#) for the latest release of this document.

Please report defects in this specification to the [issue tracker page on GitHub](#).

1.5 Licence

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Grant of Patent License. Subject to the terms and conditions of this license (both the Public License and this Patent License), each Licensor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Licensed Material, where such license applies only to those patent claims licensable by such Licensor that are necessarily infringed by their contribution(s) alone or by combination of their contribution(s) with the Licensed Material to which such contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Licensed Material or a contribution incorporated within the Licensed Material constitutes direct or contributory patent infringement, then any licenses granted to You under this license for that Licensed Material shall terminate as of the date such litigation is filed.

1.6 About the license

As identified more fully in the [Licence](#) 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.

1.7 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 [Licence](#) section.

1.8 Trademark notice

The text of and illustrations in this document are licensed by Arm under a Creative Commons Attribution–Share Alike 4.0 International license ("CC-BY-SA-4.0"), with an additional clause on patents. The Arm trademarks featured here are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Please visit <https://www.arm.com/company/policies/trademarks> for more information about Arm's trademarks.

1.9 Copyright

Copyright © 2020-2023, Arm Limited and its affiliates. All rights reserved.

Contents

1	Preamble	2
1.1	Morello alpha	2
1.2	Abstract	2
1.3	Keywords	2
1.4	Latest release and defects report	2
1.5	Licence	2
1.6	About the license	2
1.7	Contributions	3
1.8	Trademark notice	3
1.9	Copyright	3
2	About this document	6
2.1	Change control	6
2.1.1	Current status and anticipated changes	6
2.1.2	Change history	6
2.2	References	6
2.3	Terms and abbreviations	7
3	About This Specification	8
4	Object Files	8
4.1	ELF Header	8
4.2	Sections	8
4.2.1	Special Sections	8
4.3	Symbol Table	8
4.3.1	Symbol Types	8
4.3.2	Symbol names	8
4.3.3	Mapping symbols	9
4.3.4	Symbol Values	10
4.4	Relocation	10
4.4.1	Relocation types	10
4.4.2	Static Morello relocations	12
4.4.3	Dynamic Morello relocations	14
4.4.4	Static linking with Morello	15
4.4.5	Dynamic linking with Morello	16
5	APPENDIX	17
5.1	Sample initialization of capabilities at runtime	17
5.2	Sample linker generated veneers	18
5.3	TLS for the pure capability ABI	18
5.3.1	TLS static block	18

5.3.2	Thread pointer	18
5.3.3	Resolver functions	18
5.3.4	Local Exec	19
5.3.5	Initial Exec	19
5.3.6	General Dynamic	20

2 About this document

2.1 Change control

2.1.1 Current status and anticipated changes

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.

This document is a draft and all content is at the **Alpha** quality level. The relocation codes in [Relocation](#) in particular are expected to change.

2.1.2 Change history

If there is no entry in the change history table for a release, there are no changes to the content of the document for that release.

Issue	Date	Change
00alpha	1 st October 2020	Alpha release.
2020Q4	21 st December 2020	Document released on Github.
2021Q3	1 st November 2021	ELF markers to identify Morello purecap binaries.
2022Q3	20 th October 2022	Describe the purecap TLS implementation.

2.2 References

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

Ref	External reference or URL	Title
MORELLO_AAELF64	This document	Morello extensions to ELF for the Arm 64-bit Architecture (AArch64).
AAELF64	IHI 0056	ELF for the Arm 64-bit Architecture (AArch64).
MORELLO_ARM	DDI0606	Arm® Architecture Reference Manual Supplement Morello for A-profile Architecture.

Ref	External reference or URL	Title
TLSDESC	http://www.fsfla.org/~lxoliva/writ-eups/TLS/paper-lk2006.pdf	TLS Descriptors for Arm. Original proposal document.
CHERI_ELF	https://github.com/CTSRD-CHERI/cheri-elf-gabi/blob/main/gabi.md	CHERI ELF gABI Extensions
TLS	https://akkadia.org/drepper/tls.pdf	ELF Handling For Thread-Local Storage

2.3 Terms and abbreviations

The ABI for the Morello extensions to the Arm 64-bit Architecture uses the following terms and abbreviations.

C64

The instruction set available when the Morello extensions are used.

A64

The instruction set available when in AArch64 state.

Other terms may be defined when first used.

3 About This Specification

This specification only provides the Morello-specific extensions to the base ELF specification for the Arm 64-bit Architecture (AArch64), and is expected to be used along with [AAELF64](#).

[Object Files](#) is structured to correspond to the chapter with the same name in [AAELF64](#):

4 Object Files

4.1 ELF Header

The ELF header provides a number of fields that assist in interpretation of the file. Most of these are specified in the base standard. The following fields have Morello-specific meanings.

`e_flags`

The processor-specific flags are shown in the following table.

Morello-specific `e_flags`

Value	Description
<code>EF_AARCH64_CHERI_PURECAP</code> (0x00010000)	The ELF file uses an ABI where all pointers are implemented using capabilities (Pure-capability ABI).

4.2 Sections

4.2.1 *Special Sections*

A Morello toolchain can emit ELF Note sections in accordance to [\[CHERI_ELF\]](#).

4.3 Symbol Table

4.3.1 *Symbol Types*

All code symbols exported from an object file (symbols with binding `STB_GLOBAL`) shall have type `STT_FUNC`. All extern data objects shall have type `STT_OBJECT`. No `STB_GLOBAL` data symbol shall have type `STT_FUNC`. The type of an undefined symbol shall be `STT_NOTYPE` or the type of its expected definition.

The type of any other symbol defined in an executable section can be `STT_NOTYPE`. A linker is only required to provide long-branch and PLT support for symbols of type `STT_FUNC`. A linker is also only required to provide interworking support for A64 and C64 symbols of type `STT_FUNC` (interworking for untyped symbols must be encoded directly in the object file)

4.3.2 *Symbol names*

A symbol that names a C or assembly language entity should have the name of that entity. For example, a C function called `calculate` generates a symbol called `calculate` (not `_calculate`).

Symbol names are case sensitive and are matched exactly by linkers.

Any symbol with binding `STB_LOCAL` may be removed from an object and replaced with an offset from another symbol in the same section under the following conditions:

- The original symbol and replacement symbol are not of type `STT_FUNC`, or both symbols are of type `STT_FUNC` and describe code of the same instruction set state (either both A64 or both C64).
- The symbol is not described by the debug information.
- The symbol is not a mapping symbol ([Mapping symbols](#)).
- The resulting object, or image, is not required to preserve accurate symbol information to permit de-compilation or other post-linking optimization techniques.
- If the symbol labels an object in a section with the `SHF_MERGE` flag set, the relocation using symbol may be changed to use the section symbol only if the initial addend of the relocation is zero.

No tool is required to perform the above transformations; an object consumer must be prepared to do this itself if it might find the additional symbols confusing.

4.3.3 Mapping symbols

A section of an ELF file can contain a mixture of A64 code, C64 code and data. There are inline transitions between code and data at literal pool boundaries.

Linkers, file decoders and other tools need to map binaries correctly. To support this, a number of symbols, termed mapping symbols, appear in the symbol table to label the start of each sequence of bytes of the appropriate class. All mapping symbols have type `STT_NOTYPE` and binding `STB_LOCAL`. The `st_size` field is unused and must be zero.

The mapping symbols are defined in [Mapping symbols table](#). It is an error for a relocation to reference a mapping symbol. Two forms of mapping symbol are supported:

- A short form that uses a dollar character and a single letter denoting the class. This form can be used when an object producer creates mapping symbols automatically. Its use minimizes string table size.
- A longer form in which the short form is extended with a period, followed by any sequence of characters that are legal for a symbol. This form can be used when assembler files have to be annotated manually and the assembler does not support multiple definitions of symbols.

Mapping symbols defined in a section (relocatable view) or segment (executable view) define a sequence of half-open intervals that cover the address range of the section or segment. Each interval starts at the address defined by the mapping symbol, and continues up to, but not including, the address defined by the next (in address order) mapping symbol or the end of the section or segment. A section that contains instructions must have a mapping symbol defined at the beginning of the section. If a section contains only data, no mapping symbol is required. A platform ABI should specify whether or not mapping symbols are present in the executable view; they will never be present in a stripped executable file.

Mapping symbols

Name	Description
<code>\$x</code> <code>\$x.<any...></code>	Start of a sequence of A64 instructions.
<code>\$c</code> <code>\$c.<any...></code>	Start of a sequence of C64 instructions.
<code>\$d</code> <code>\$d.<any...></code>	Start of a sequence of data items (for example, a literal pool).

4.3.4 Symbol Values

In addition to the normal rules for symbol values, the following rules shall also apply to symbols of type `STT_FUNC` and `STT_GNU_IFUNC`:

- If the symbol addresses an A64 instruction, its value is the address of the instruction (in a relocatable object, the offset of the instruction from the start of the section containing it).
- If the symbol addresses a C64 instruction, its value is the address of the instruction with bit 0 set (in a relocatable object, the section offset with bit 0 set).

Note

This allows a linker to distinguish A64 and C64 code symbols without having to refer to the map. An A64 symbol will always have an even value, while a C64 symbol will always have an odd value. However, a linker should strip the discriminating bit from the value before using it for relocation.

4.4 Relocation

4.4.1 Relocation types

4.4.1.1 Relocation codes

Morello uses the private relocation code space for vendor experiments [0xE000, 0xF000) specified in [AAELF64](#).

Static Morello relocation codes begin at 0xE000(57344); dynamic ones at 0xE800(59392). Relocation codes starting at 0xEA00(59904) are reserved for private Morello experiments.

4.4.1.2 Relocation operations

The following nomenclature is used in the descriptions of relocation operations:

- `S` (when used on its own) is the address of the symbol.
- `A` is the addend for the relocation.
- `P` is the address of the place being relocated (derived from `r_offset`).
- `C` is 1 if the target symbol `S` has type `STT_FUNC` and the symbol addresses a C64 instruction; it is 0 otherwise.
- `X` is the result of a relocation operation, before any masking or bit-selection operation is applied
- `Page(expr)` is the page address of the expression `expr`, defined as `(expr & ~0xFFF)`. This applies even if the machine page size supported by the platform has a different value.
- `GOT` is the address of the Global Offset Table, the table of code and data addresses to be resolved at dynamic link time. The `GOT` and each entry in it must be aligned to the pointer-size.
- `GDATA(S+A)` represents a pointer-sized entry in the `GOT` for address `S+A`. The entry will be relocated at run time with relocation `R_MORELLO_GLOB_DAT(S+A)`.
- `G(expr)` is the address of the `GOT` entry for the expression `expr`.
- `GTLSDESC(S+A)` represents a consecutive pair of pointer-sized entries in the `GOT` which contain a `tlsdesc` structure describing the thread-local variable located at offset `A` from thread-local symbol `S`. The first entry holds a pointer to the variable's TLS descriptor resolver function and the second entry holds a platform-specific offset or pointer. The pair of pointer-sized entries will be relocated with `R_MORELLO_TLSDESC(S+A)`.
- `TPREL(S)` resolves to a pair of two 64-bit values. The first value contains the offset in the static TLS block of the thread-local symbol `S`. The second value contains the size of the symbol `S`.

- `GTPREL(S)` represents an entry in the GOT containing a pair of two 64-bit values. The first value contains the offset in the static TLS block of the symbol `S`. The second value contains the size of the symbol `S`.
- `TLSDESC(S+A)` resolves to a contiguous pair of pointer-sized values, as created by `GTLSDDESC(S+A)`.
- `CAP_INIT` generates a capability with all required information. When used on its own represents the operations needs to be done for handling `R_MORELLO_CAPINIT`.
- `CAP_SIZE` is the size of the underlying memory region that the capability can reference. This may not directly map to the symbol size.
- `SIZE(S)` is the symbol size of the symbol `S`.
- `CAP_PERM` is the permission of the capability. This may not directly map to the type of the symbol.
- `[msb:lsb]` is a bit-mask operation representing the selection of bits in a value. The bits selected range from `lsb` up to `msb` inclusive. For example, 'bits [3:0]' represents the bits under the mask `0x0000000F`. When range checking is applied to a value, it is applied before the masking operation is performed.

pointer-size

The pointer-size is 64 bits for the A64 ABI and 128 bits for the pure capability (C64) ABI.

4.4.2 Static Morello relocations

Warning

The ELF64 Code of the relocations are subject to change.

Relocations to generate 19, 21, and 33 bit PC-relative addresses

ELF64 Code	Name	Operation	Comment
57348	R_MORELLO_LD_PREL_LO17	$S+A - (P \& \sim 0xF)$	Set a load-literal immediate value to bits [20:4] of X. Check that $-2^{20} \leq X < 2^{20}$. Check that $X \& 15 = 0$.
57349	R_MORELLO_ADR_PREL_PG_HI20	$\text{Page}(S+A) - \text{Page}(P)$	Set an ADRP immediate value to bits [31:12] of the X. Check that $-2^{31} \leq X < 2^{31}$.
57350	R_MORELLO_ADR_PREL_PG_HI20_NC	$\text{Page}(S+A) - \text{Page}(P)$	Set an ADRP immediate value to bits [31:12] of the X. No overflow check. Although overflow must not be checked, a linker should check that the value of X is aligned to a multiple of the datum size.

Relocations for control-flow instructions - all offsets are a multiple of 4

ELF64 Code	Name	Operation	Comment
57344	R_MORELLO_TSTBR14	$((S+A) C) - P$	Set the immediate field of a TBZ/TBNZ instruction to bits [15:2] of X. Check that $-2^{15} \leq X < 2^{15}$. See Call and Jump relocations .
57345	R_MORELLO_CONDBR19	$((S+A) C) - P$	Set the immediate field of a conditional branch instruction to bits [20:2] of X. Check that $-2^{27} \leq X < 2^{27}$. See Call and Jump relocations .
57346	R_MORELLO_JUMP26	$((S+A) C) - P$	Set a B immediate field to bits [27:2] of X. Check that $-2^{27} \leq X < 2^{27}$. See Call and Jump relocations .
57347	R_MORELLO_CALL26	$((S+A) C) - P$	Set a BL immediate field to bits [27:2] of X. Check that $-2^{27} \leq X < 2^{27}$. See Call and Jump relocations .

Group relocations to create a 16-, 32-, 48-, or 64-bit symbol size inline

ELF64 Code	Name	Operation	Comment
57353	R_MORELLO_MOVW_SIZE_G0	SIZE	Set a MOV[KZ] immediate field to bits [15:0] of X; check that $0 \leq X < 2^{16}$
57354	R_MORELLO_MOVW_SIZE_G0_NC	SIZE	Set a MOV[KZ] immediate field to bits [15:0] of X. No overflow check

ELF64 Code	Name	Operation	Comment
57355	R_MORELLO_MOVW_SIZE_G1	SIZE	Set a MOV[KZ] immediate field to bits [31:16] of X; check that $0 \leq X < 2^{32}$
57356	R_MORELLO_MOVW_SIZE_G1_NC	SIZE	Set a MOV[KZ] immediate field to bits [31:16] of X. No overflow check
57357	R_MORELLO_MOVW_SIZE_G2	SIZE	Set a MOV[KZ] immediate field to bits [47:32] of X; check that $0 \leq X < 2^{48}$
57358	R_MORELLO_MOVW_SIZE_G2_NC	SIZE	Set a MOV[KZ] immediate field to bits [47:32] of X. No overflow check
57359	R_MORELLO_MOVW_SIZE_G3	SIZE	Set a MOV[KZ] immediate field to bits [63:48] of X (no overflow check needed)

Note

The group relocations to create a 16-, 32-, 48-, or 64-bit symbol size inline do not accept an addend.

4.4.2.1 Call and Jump relocations

There is one relocation code (R_MORELLO_CALL26) for function call (BL) instructions and one (R_MORELLO_JUMP26) for jump (B) instructions.

A linker may use a veneer (a sequence of instructions) to implement a relocated branch if the relocation is either

R_MORELLO_CALL26 or R_MORELLO_JUMP26 and:

- The target symbol has type STT_FUNC.
- Or, the target symbol and relocated place are in separate sections input to the linker.
- Or, the target symbol is undefined (external to the link unit).

In all other cases a linker shall diagnose an error if a relocation cannot be effected without a veneer. A linker generated veneer may corrupt register `c16` and the condition flags, but must preserve all other registers. Linker veneers may be needed for a number of reasons, including, but not limited to:

- Interworking: The branch source and target symbol are in different execution states (A64/C64).
- Range Extension: The branch source and target symbol are in C64 execution state and the target is outside the addressable span of the branch instruction (+/- 128MB).
- The target address will not be known until run time, or the target address might be pre-empted.

Long branches with 64-bit range are not supported yet for range extensions or for interworking. Interworking between ABIs are not supported yet.

GOT-relative instruction relocations

ELF64 Code	Name	Operation	Comment
57351	R_MORELLO_ADR_GOT_PAGE	$\text{Page}(G(\text{GDAT}(S+A))) - \text{Page}(P)$	Set the immediate value of an ADRP to bits [31:12] of X. Check that $-2^{31} \leq X < 2^{31}$.

ELF64 Code	Name	Operation	Comment
57352	R_MORELLO_LD128_GOT_LO12_NC	$G(GDAT(S+A))$	Set the LD/ST immediate field to bits [11:4] of X. No overflow check. Check that $X \& 15 = 0$. Also see Static linking with Morello .

4.4.2.2 Relocations for thread-local storage

Morello only defines the relocations needed to implement the descriptor based thread-local storage (TLS) models in a SysV-type environment. The details of TLS descriptors are beyond the scope of this specification; a general introduction can be found in [\[TLSDESC\]](#). Also, only the relocations needed to implement the General Dynamic (GD) access model and the Local Executable (LE) access models are defined.

Relocations needed to define the traditional TLS models are undefined.

TLS descriptor relocations

ELF64 Code	Name	Operation	Comment
57600	R_MORELLO_TLSDESC_ADR_PAGE20	$Page(G(GTLSDESC(S+A))) - Page(P)$	Set the immediate value of an ADRP to bits [31:12] of X. Check that $-2^{31} \leq X < 2^{31}$.
57601	R_MORELLO_TLSDESC_LD128_LO12	$G(GTLSDESC(S+A))$	Set the LD/ST immediate field to bits [11:4] of X. No overflow check. Check that $X \& 15 = 0$.
57602	R_MORELLO_TLSDESC_CALL	None	For relaxation only. Must be used to identify a BLR instruction which performs an indirect call to the TLS descriptor function for $S + A$.
57603	R_MORELLO_TLSIE_ADR_GOTTPREL_PAGE20	$Page(G(GTPREL(S))) - Page(P)$	Set the immediate value of an ADRP to bits [31:12] of X. Check that $-2^{31} \leq X < 2^{31}$.
57604	R_MORELLO_TLSIE_ADD_LO12	$G(GTPREL(S))$	Set the ADD immediate field to bits [11:0] of X. No overflow check.

4.4.3 Dynamic Morello relocations

Dynamic relocations

ELF64 Code	Name	Operation	Comment
59392	R_MORELLO_CAPINIT	$CAP_INIT(S, A, CAP_SIZE, CAP_PERM)$	See note below.

ELF64 Code	Name	Operation	Comment
59393	R_MORELLO_GLOB_DAT	CAP_INIT(S, A, CAP_SIZE, CAP_PERM)	See note below.
59394	R_MORELLO_JUMP_SLOT	CAP_INIT(S, A, CAP_SIZE, CAP_PERM)	See note below.
59395	R_MORELLO_RELATIVE	CAP_INIT(S, A, CAP_SIZE, CAP_PERM)	See note below.
59396	R_MORELLO_IRELATIVE	CAP_INIT(S, A, CAP_SIZE, CAP_PERM)	See note below.
59397	R_MORELLO_TLSDESC	TLSDESC(S+A)	Identifies a TLS descriptor to be filled.
59398	R_MORELLO_TPREL128	TPREL(S)	See note below.

Note

R_MORELLO_CAPINIT instructs the runtime or dynamic loader to create a 16-byte capability at `r_offset`. `r_offset` must be 16-byte aligned. An object producer may communicate a hint about the size of the capability to the static linker in the 16-byte fragment identified by `r_offset`. The fragment has the following format:

```
| 64-bits empty | 64-bits size |
```

R_MORELLO_GLOB_DAT instructs the runtime or dynamic loader to create a 16-byte capability in the GOT entry identified by `r_offset`. The capability holds the address of a data symbol which must be resolved at load time when dynamic linking.

R_MORELLO_JUMP_SLOT instructs the dynamic loader to create a 16-byte capability in the GOT entry identified by `r_offset`. The capability holds the address of a function symbol which must be resolved at load time.

R_MORELLO_RELATIVE represents an optimization of R_MORELLO_GLOB_DAT. It can be used when the symbol resolves to the current shared object or executable. `S` must be the Null symbol (Index 0). The address and permissions must be written to the fragment. See [Dynamic linking with Morello](#) for details.

R_MORELLO_IRELATIVE is used by the linker when transforming IFUNC `s`. The rest are the same as R_MORELLO_RELATIVE

R_MORELLO_TLSDESC : identifies a TLS descriptor to be filled by the dynamic loader. If the size of `S` is known by the static linker the 256-bit fragment will contain the size of the symbol in the last 64 bits of the fragment. Otherwise the fragment will contain all zeroes. The fragment has the following format:

```
| 192-bits empty | 64-bits size |
```

R_MORELLO_TPREL128 : instructs the dynamic loader to create a pair of two 64-bit integers, the first integer containing the offset of `S` in the TLS block and the second integer containing the size of the symbol `S`. The first 64-bit integer (the offset) has the same fragment encoding as R_AARCH64_TLS_TPREL. If the size of `S` is known by the static linker the second 64-bit integer in the fragment will contain the size of the symbol. The fragment has the following format:

```
| 64-bits offset | 64-bits size |
```

4.4.4 Static linking with Morello

A capability has more associated information than a conventional pointer. It has extra information. For example: base, offset, size and permissions.

Capabilities cannot be statically initialised. Global capability initialization when static linking is performed by the runtime at program startup. The communication between the static linker and runtime is implementation defined. This document describes an implementation based on a table of capability descriptions created at static link time, where each capability-generating relocation results in one entry in the table. When static linking, all capability descriptions will

be explicitly grouped into a single table of capability descriptions where each table entry is a `struct capdesc` (listed below).

In the current LLVM based Morello toolchain, the runtime iterates through each `capdesc` entry creating a capability in the location pointed to by `cap_location`, with the specified base, offset, size and permissions given by the entry. To aid in the finding of the capability descriptions table, the linker emits two symbols to denote the start and end of the table: `__cap_relocs_start` and `__cap_relocs_end` respectively. The capability descriptions table is placed inside the `__cap_relocs` section.

```
struct capdesc
{
    void* __capability cap_location;
    void* base;
    uint64_t offset;
    uint64_t size;
    uint64_t permissions;
};
```

The permission bits of a capability constructed for a `capdesc` entry is the inverse of the `permissions[17:0]` field in the `capdesc` entry. Additionally, the MSB (bit 64) of the `capdesc permissions` field is set for Executable symbols to indicate that the PCC is to be used to construct the Capability.

Permission	Encoding
Executable	0x80000000000013DBCULL
Read-Write Data	0x8FBEULL
Read-Only Data	0x1BFBEULL

When a Morello-capable assembler sees a `.capinit` instruction, it reserves a 16-byte (128 bits) location (`fragment`) and generates a `R_MORELLO_CAPINIT` relocation for the linker to create a capability in the `fragment`. The assembler may use the `fragment` with the following format to give out size hints for the linker to use before processing the relocation:

64-bit: empty	64-bit: size
---------------	--------------

This size hint will be incorporated into the `capdesc size` field, if not superseded by more accurate information.

In case of position independent code (PIC), the assembler will generate a `R_MORELLO_LD128_GOT_LO12_NC` relocation, which causes the linker to generate a 16-byte aligned, 16-byte sized entry in the `.got` that will be initialised by a `capdesc` entry in a capability descriptions table with the address of the `.got` entry as its `location` field. All information required to initialize the capability is self-contained in the `capdesc` entry, so the linker is not required to provide any size hints in the `.got` entry.

4.4.5 Dynamic linking with Morello

When dynamic linking, capability initialization is done by the dynamic linker as a result of processing one of the dynamic relocations listed in [Dynamic relocations table](#). For `R_MORELLO_RELATIVE` and `R_MORELLO_IRELATIVE` relocations, the static linker must write the following information to the fragment identified by `r_offset`.

64-bit: address	56-bits: length	8-bits: permissions
-----------------	-----------------	---------------------

The 8-bit permission field of the fragment encodes the symbol permissions as below.

Permission	Encoding
Executable	0x4ULL
Read-Write Data	0x2ULL

Permission	Encoding
Read-Only Data	0x1ULL

As in [Static linking with Morello](#), the linker creates a 16-byte aligned, 16-byte sized entry in the `.got` for the `R_MORELLO_LD128_GOT_LO12_NC` relocation generated by the assembler. However, a capability descriptions table is not generated to initialize the `.got` entry. Instead it is expected that the dynamic linker generates the table itself based on the `R_MORELLO_GLOB_DAT` and `R_MORELLO_JUMP_SLOT` relocations created by the static linker. The dynamic linker writes the generated capabilities back into the `.got` entry.

5 APPENDIX

The status of this appendix is `informative`.

5.1 Sample initialization of capabilities at runtime

The following code is sample runtime initialization code that initializes global capabilities created by an LLVM-based Morello toolchain.

```
__init_global_caps:
    mrs     c2, DDC      /* Default data capability */
    adrp    c0, __cap_relocs_start
    add     c0, c0, #:lo12:__cap_relocs_start
    adrp    c1, __cap_relocs_end
    add     c1, c1, #:lo12:__cap_relocs_end
    gcvalue x1, c1
    gcvalue x0, c0
    cmp     x0, x1
    b.eq    .CapInitEnd
    sub     x5, x1, x0    /* __cap_relocs_size */
    scvalue c0, c2, x0
    scvalue c1, c2, x1
    /* Clear permissions that we're not going to want on global capabilities. */
    ldr     x5, =(BIT_07 | \ /* Compartment ID */
                  BIT_08 | \ /* Branch Unseal */
                  BIT_10 | \ /* Unseal */
                  BIT_11 ) /* Seal */
    clrperm c2, c2, x5
.CapInit:
    ldr     x5, [c0], #8   /* Capability location */
    ldr     x24, [c0], #8  /* Object referred by the capability */
    cbnz    c24, .CapNonNull
    add     c0, c0, #24
    mov     x4, #0         /* c4 <- nullptr */
    b       .CapCont
.CapNonNull:
    ldr     x25, [c0], #8  /* Offset in the object */
    ldr     x26, [c0], #8  /* Size */
    ldr     x9, [c0], #8   /* Permissions */
    /* Set the executive permission for executable capabilities */
    scvalue c4, c2, x24    /* Set capability base */
    scbndse c4, c4, x26    /* Set size */
    scoff   c4, c4, x25    /* Add offset */
    clrperm c4, c4, x9     /* Clear permission bits set in __cap_desc */
.CapCont:
    scvalue c5, c2, x5
    str     c4, [c5]
```

```

        cmp     c0, c1
        b.ne    .CapInit
.CapInitEnd:
        ret

```

5.2 Sample linker generated veneers

For C64 to A64 interworking, the following veneer is used:

```

adrp c16, sym
add c16, c16, :lo12:sym
br c16

```

For A64 to C64 interworking, and for C64 to C64 Range Extension, the following veneer is used. The BX changes the execution state from A64 to C64:

```

bx #4
adrp c16, sym
add c16, c16, :lo12:sym
br c16

```

5.3 TLS for the pure capability ABI

The design is based on TLSDESC, with the purpose of minimizing the performance differences between A64 and C64, while providing strict bounds when resolving TLS globals.

5.3.1 TLS static block

The static block layout is the same used in AArch64 (Variant 1, see [\[TLS\]](#)), with the only exception that TCB and the DTV pointer are capabilities.

5.3.2 Thread pointer

The thread pointer is a capability, held in `CTPIDR_EL0`. The thread pointer needs to have the read, write, read capability and write capability permissions and bounds such that the entire TLS static block is accessible.

5.3.3 Resolver functions

A resolver function takes arguments in `c0` (address of the TLS GOT slot), and `c2` (a copy of the thread pointer) and returns a pointer to the TLS global in `c0`. The resolver function has a custom calling convention that must preserve all registers except `c0` and `c1`.

Considerations:

- Any dynamically loaded modules will be placed outside of the bounds of the thread pointer, so a resolver function cannot return an offset from the thread pointer, but rather needs to return a pointer (capability).
- To minimize reading of `CTPIDR_EL0`, the resolver functions take a copy of `CTPIDR_EL0` as an argument and preserve it.

5.3.3.1 Static TLS block resolver

If the TLS variable is in the static block, while resolving the `R_MORELLO_TLSDESC` relocation, the dynamic linker will place in the two GOT slots associated with this variable:

- A capability to the static TLS block resolver function at offset 0.
- The offset of the variable in the static TLS block at offset 16 (8 bytes).
- The size of the variable at offset 24 (8 bytes).

An implementation of the static block resolver could be the following:

```
ldp x0, x1, [c0, #16]
add c0, c2, x0
scbnds c0, c0, x1
ret c30
```

5.3.4 Local Exec

The capability to the TLS variable is derived from `CTPIDR_EL0`. There are no requirements on how this is performed or the registers used, except that the sequence doesn't produce a dynamic relocation. A possible instruction sequence could be:

```
mrs c0, CTPIDR_EL0
movz x8, #:tprel_g1:local_exec_var
movk x8, #:tprel_g0_nc:local_exec_var
movz x9, #:size_g1:local_exec_var
movk x9, #:size_g0_nc:local_exec_var
add c0, c0, x8, uctx
scbnds c0, c0, x9
```

5.3.5 Initial Exec

The capability to the TLS variable is derived from `CTPIDR_EL0`. The size and offset of the TLS variable is stored in a GOT slot (first 8 bytes contains the offset and the second 8 bytes the size). This GOT slot is initialized by a `R_MORELLO_TPREL128` dynamic relocation. The access must use the `R_MORELLO_TLSIE_ADR_GOTTPREL_PAGE20` and `R_MORELLO_TLSIE_ADD_LO12` relocations in order to allow relaxation to Local Exec. There are no other requirements on how this is performed or the registers used. A possible instruction sequence could be:

```
adrp c0, :gottprel:initial_exec_var
add c0, c0, :gottprel_lo12:initial_exec_var
ldp x0, x8, [c0]
mrs c1, CTPIDR_EL0
add c0, c1, x0, uctx
scbnds c0, c0, x8
```

5.3.5.1 Initial Exec to Local Exec relaxation

The linker will generate 16 bytes in a read-only section, containing the offset in the static TLS block in the first 8 bytes and the size of the symbol in the next 8 bytes:

```
.section .rodata
_sym_data:
```

```
.xword tlsoffset(sym)
.xword sizeof(sym)
```

Note

`tlsoffset(sym)` denotes the offset in the static TLS block of the symbol `sym`, while `sizeof(sym)` denotes the size of the symbol `sym`. These are not valid assembler directives.

The relaxation is performed by:

- changing the `R_MORELLO_TLSIE_ADR_GOTTPREL_PAGE20` relocation on the symbol `sym` to a `R_MORELLO_ADR_PREL_PG_HI20` with the symbol `_sym_data`
- changing the `R_MORELLO_TLSIE_ADD_LO12` relocation on symbol the `sym` to a `R_AARCH64_ADD_ABS_LO12_NC` relocation with the symbol `_sym_data`.

Note

The symbol and section names in the example above are only used for explanation purposes. An implementation does not need to create an additional symbol when performing this relaxation. There is no constraint on the name of the read-only section where the data is placed.

5.3.6 General Dynamic

The instruction sequence used for the General Dynamic access model is similar to that of other TLSDESC implementations, with the exception that the result doesn't need to be added to the thread pointer. However `c2` needs to contain the thread pointer. The instruction sequence contains an additional NOP instruction in order to permit the static linker to perform a relaxation to Local Exec or Initial Exec.

The General Dynamic access sequence must be output in the following form to allow correct linker relaxation:

```
adrp c0, :tlsdesc:sym
ldr c1, [c0, :tlsdesc_lo12:sym]
add c0, c0, :tlsdesc_lo12:sym
nop
.tlsdescscall sym
blr c1
```

5.3.6.1 General Dynamic to Initial Exec relaxation

The relaxed sequence is:

```
adrp c0, :gottprel:sym
add c0, c0, :gottprel_lo12:sym
ldp x0, x1, [c0]
add c0, c2, x0
scbnds c0, c0, x1
```

5.3.6.2 General Dynamic to Local Exec relaxation

The linker will generate 16 bytes in a read-only section, containing the offset in the static TLS block in the first 8 bytes and the size of the symbol in the next 8 bytes:

```
.section .rodata
_sym_data:
    .xword tlsoffset(sym)
    .xword sizeof(sym)
```

Note

`tlsoffset(sym)` denotes the offset in the static TLS block of the symbol `sym`, while `sizeof(sym)` denotes the size of the symbol `sym`. These are not valid assembler directives.

The relaxed sequence is:

```
adrp c0, _sym_data
add c0, c0, :lo12:_sym_data
ldp x0, x1, [c0]
add c0, c2, x0
scbnds c0, c0, x1
```