Morello Instruction Emulator User Guide

Release 1.1

Arm Ltd

Feb 18, 2021

CONTENTS

Document ID: 102270

Date of Issue: February 2021

Change history

Issue	Date	Change
0.1	October 2020	Initial version
0.2	February 2021	Clarification of known issues and limitations. New command line options. Updates of instruction and memory tracer. New tools: debugger and cache model.

License

LES-PRE-20349 version 21.0

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is for a Beta product, that is a product under development.

Web Address

https://www.arm.com

CHAPTER

ONE

INTRODUCTION

1.1 Product revision status

Morello Instruction Emulator version: 1.1.

1.2 Intended audience

The Morello Instruction Emulator (Morello IE) is a tool for software developers and researchers who are interested in the Morello architecture. It allows you to run userspace Morello applications on AArch64 Linux systems in a non-Morello environment. It also includes runtime instrumentation that can collect information about events and counters related to Morello and an interactive debugger to help with running Morello applications.

The emulator can be used for:

- Experiments with the Morello architecture in userspace on AArch64 Linux systems.
- Porting existing software to Morello and testing ported libraries and applications.
- Analysing performance of Morello applications.
- Functional modelling of capabilities.

The Morello Instruction Emulator is an experimental tool and it does not provide a full Morello user space environment. The Morello IE must not be used for running code in production.

1.3 System requirements

Morello IE is released in pre-built binary form and requires the following:

- AArch64 v8.x GNU/Linux system (for example, Red Hat 7.x or Ubuntu 18.04).
- AArch64 v8.x hardware and existing user space environment of a Linux system.
- Arm v8.0 hardware (v8.2 or above hardware is recommended).
- The host system must have Glibc 2.17 or above.

1.4 Installation

The distribution bundle for Morello IE contains all required libraries and files including examples and documentation. To install this tool, unpack the bundle at any location. To use the emulator launcher, add the `bin` directory of the installation folder to your `PATH` environment variable. The installation is self-consistent and portable.

Morello architecture is backwards compatible with AArch64, and you can run an AArch64 application with the emulator. Run the following command to check that installation is successful:

```
morelloie -- uname -m
```

This command runs `uname -m` in the emulator, and displays the following result:

```
aarch64
```

Installation directory includes:

- `bin` directory with launcher and frontend binaries.
- `lib` directory with instrumentation clients.
- `license_terms` directory with licence information.
- `docs` directory with this user guide.
- `samples` directory with some useful examples.

1.5 Additional reading

This document contains information that is specific to this product. See the following documents for other relevant information:

Reference	Document name	Document ID
[Morello IE]	This document	102270
[Morello AAPCS]	Morello Supplement to the Procedure Call Standard for the Arm 64-bit Architecture (AArch64)	102205
[Morello AAELF]	Morello Supplement to ELF for the Arm64-bit Architecture (AArch64)	102272
[Morello LLVM]	LLVM toolchain user guide	102216
[CHERI]	CHERI Hybrid C/C++ Programming Guide	

CHAPTER

TWO

OVERVIEW

2.1 General description

Morello Instruction Emulator (Morello IE) is a dynamic binary translation tool. It is based on the JIT translation of each Morello instruction into a series of AArch64 instructions using DynamoRIO[1] instrumentation. The emulator maintains a consistent emulated CPU state as well as emulated memory tags. It also provides a layer of compatibility between Morello user space application and non-Morello C library and kernel. This compatibility layer creates an environment which allows Morello applications to work on a non-Morello system.

2.2 Components

Morello Instruction Emulator includes the following components:

- Emulator and debugger (`libmie.so` instrumentation client library).
- Tracer (`libtracer.so` instrumentation client library).
- Launcher (`launcher` binary and `drrun` binary).
- Morello IE frontend (`morelloie` binary, a wrapper for the launcher and instrumentation clients).

2.2.1 Emulator

The emulator of Morello instructions implements Morello instructions by replacing them with AArch64 code which can be executed natively. The implementation relies on the emulated CPU state that is maintained by the emulator for each process thread. This state is synchronised on demand with real execution context, including the values of hardware registers and the objects and memory managed through the API of the C library.

2.2.2 Debugger

A simple interactive debugger gives access to the emulated state and runtime information for every executed instruction. It supports basic commands such as printing the emulated CPU state, working with PC-based breakpoints, and identifying location of the current execution point (for example, printing backtrace). The debugger provides access to data within the emulator that is not available by means of `lldb` or `gdb`. Both emulator and debugger are parts of `libmie.so` instrumentation client library.

[1] https://dynamorio.org/

2.2.3 Tracer

Instruction and memory access tracer is implemented as a separate instrumentation client for DynamoRIO and can be used independently of the instruction emulation client. For example, it can be used for analysing non-Morello AArch64 applications. It allows capturing runtime trace of executed instructions and accesses to memory. The scope of tracing can be managed to include either entire application execution flow or certain regions of interest.

Tracer also includes a module for collecting various runtime statistics, including counters specific for Morello, and also data related to CPU cache modelling based on DynamoRIO's `drcachesim` tool. Tracer, statistics counter, and cache model are parts of `libtracer.so` instrumentation client library.

2.2.4 Launcher

Running applications with instrumentation requires using the `drrun` binary from DynamoRIO. The `launcher` binary is an alternative for `drrun` and can be used together with the emulator and tracer clients providing easier command line usage.

The `morelloie` frontend is a binary that aims to simplify using the Morello IE tool by managing control over its various components from a single point.

2.3 Command line options

This section gives an overview of all the command line options supported by Morello IE.

2.3.1 Frontend options

Use the Morello Instruction Emulator through its frontend, the `morelloie` binary. The general usage is:

```
morelloie [options] -- application [application's argument(s)]
```

The general rule is that the double hyphen delimiter `--` separates options for `morelloie` from the command line used to create the process of the application.

In general, any boolean option can be inverted by adding `-no` in front of the flag. For example, use `-no-enable-foo` to invert option `-enable-foo`. For options that have a value, separate the option and the value with a whitespace. For options that take a PC address, provide the address value as a hexadecimal literal with `0x` prefix. Specify options that accept integer values using decimal integer literals.

The frontend supports the command line options described in the following sections.

Options to control instrumentation

You can supply the following options, except for `-no-mie`, directly to the `launcher` binary:

- `-v` – verbose output from the launcher (by default this output is suppressed). The output is directed to `stderr`.
- `-f` – use the `fork` syscall instead of `execve` to launch the process. The default is to use `execve`.
- `-fsz <n>` – maximum fragment size. This option limits the number of application instructions included in a single instrumented block of code, and sets the `max_bb_instrs` option of DynamoRIO. The default value is `64`. Increasing this value might improve instrumentation performance, but might also result in exceeding internal code cache limits in DynamoRIO.
- `-no-mie` – disable use of Morello emulation client. This option can be helpful when only the tracer client needs to be used for analysing non-Morello applications.

Options to control Morello emulation client

The following options can be supplied directly to the `libmie.so` instrumentation client:

- `-f-cpu` – show the CPU state on a capability fault. By default, this information is not shown.
- `-f-debug` – enter debug mode on a capability fault. Normally, the application stops immediately after a capability fault. This option allows the client to enter debug mode for gathering additional information about runtime context.
- `-break <pc>` – pause execution immediately before executing the instruction at the given `<pc>` address. If not specified explicitly, this option has no effect (there is no default value). Only a single instance of this option can be used on a command line.
- `-debug` – enable debug features. This option is automatically implied when using one of the `-f-debug` or `-break` options. If not specified explicitly, this option has no effect. This option must be used to enable tracking of instruction markers for the debugger (see *Marker for debugger* (page 9)).
- `-strict-a64-store` – track tags for capabilities in memory during AArch64 stores. This option is enabled by default. Disabling this option with `-no-strict-a64-store` stops invalidation of memory tags by AArch64 store operations but also increases the speed of emulation.
- `-strict-c64-mem` – check all memory operations in Purecap (C64) mode. This option is enabled by default. Disabling this option with `-no-strict-c64-mem` ignores capability faults that would be generated by AArch64 instructions but also increases the speed of emulation.
- `-verbose` – show extra diagnostic messages from the Morello emulation client.

Options to control tracer instrumentation client

The following options can be supplied directly to the `libtracer.so` instrumentation client:

- `-instr` – enable instruction trace.
- `-mem` – enable memory access trace.
- `-trace` – enable trace (instructions and memory), equivalent to using both `-instr -mem`.
- `-stat` – enable collection of statistics.
- `-cache` – enable cache model and collecting associated data.
- `-full` – show detailed results from statistics or cache model.
- `-verbose` – show extra diagnostic messages from the tracer instrumentation client.

Options to control scope of tracing

By default, the scope of tracing corresponds to everything executed from the start of `main` function up to and including the return from it. The exception is running stripped applications for which all code is traced. Use the following options to modify scope tracing:

- `-all` – trace all executed instructions, including the instructions outside the `main` function.
- `-roi` – trace only instructions from the region of interest defined by the tracer marker instructions (see *Markers for tracing* (page 9)).
- `-c64` – trace only instructions executed in Purecap (C64) mode.
- `-fr <pc1> -to <pc2>` – define the region of interest for tracing using addresses `<pc1>` and `<pc1>`.

Note: These options are mutually exclusive.

Options for cache model

The following options have no effect unless `-cache` option is used:

- `-l1-d-size` – L1 d-cache size (bytes), and size must be power of two. The default is 64 KiB.
- `-l1-i-size` – L1 i-cache size (bytes), and size must be power of two. Th default is 64 KiB.
- `-l2-size` – L2 cache size (bytes), and size must be power of two. The default is 512 KiB.
- `-cache-line-size` – Cache line size (bytes), and size must be power of two. The default is 64.
- `-l1-d-ways` – L1 d-cache associativity, must be power of two. The default is 4.
- `-l1-i-ways` – L1 i-cache associativity, must be power of two. The default is 4.
- `-l2-ways` – L2 cache associativity, must be power of two. The default is 8.
- `-cache-cores` – the number of cores in the cache model. The default is 1.

Miscellaneous options

- `-h`, `-help` – display command line options for the launcher and the instrumentation clients.
- `-version` – display version information and exit.

Note: When using `morelloie` frontend, if none of the tracer options are specified, the tracer client is not loaded. This improves the performance of the emulation. To disable loading of Morello emulation client `libmie.so`, use the option `-no-mie`. This option is only available for the `morelloie` frontend.

2.3.2 Launcher options

If `launcher` binary is used directly, the usage template is:

```
launcher [options] [-c /path/to/lib/libtracer.so [tracer-options]] \
[-c /path/to/lib/libmie.so [emulator-options]] -- application [application's argument(s)]
```

The general rule is that any options supplied before `-c` flag are consumed by the `launcher` itself. The `-c` flag starts command line for each instrumentation client. The flag must be followed by the path to the client's library and optionally by the arguments intended for the client. All flags between `-c` and either the next `-c` flag or the double hyphen delimiter `--` are part of the command line for the corresponding client library. Everything after the `--` delimiter forms the command line to start the process of the application.

Options and flags described in *Frontend options* (page 6) also apply here.

2.4 Inline control instructions

Morello IE supports special marker instructions that can be embedded in the source code of applications using the macros shown in the following section to control the scope of tracing or introducing breakpoints for the interactive Morello debugger.

2.4.1 Markers for tracing

The following macros can be used to embed start and stop tracing markers. Every time the execution reaches these instructions, tracing is respectively enabled or disabled. To make use of these markers, specify `-roi` flag. Without this flag, marker instructions have no effect.

```
/* Start tracing */
#define __START_TRACE() asm volatile ("hint #0b1000000")

/* Stop tracing */
#define __STOP_TRACE()  asm volatile ("hint #0b1000001")
```

The resulting instructions are valid AArch64 instructions equivalents of `NOP` and do not affect the functionality of the application.

2.4.2 Marker for debugger

The following macro can be used to insert a breakpoint marker that is recognised by the Morello emulation client if debug mode is enabled. Use `-debug` flag to enable this mode.

```
/* Put breakpoint */
#define __MIE_DEBUG() asm volatile ("hint #0b1000100")
```

The resulting instruction is a valid AArch64 instruction equivalent of `NOP` and does not affect the functionality of the application.

2.5 Limitations and known issues

Morello Instruction Emulator is capable of running Purecap and Hybrid (see [Morello AAPCS] for more details) Morello userspace applications on non-Morello AArch64 Linux systems. However the following exceptions and limitations must be taken into account.

2.5.1 Running dynamically linked applications

On systems with no Morello-aware dynamic linker, dynamically linked Purecap Morello applications do not run because of the lack of support for Morello relocations. A workaround is to build statically-linked Purecap applications that can be executed under Morello IE.

However, both dynamically and statically linked Hybrid Morello applications do work.

2.5.2 Running stripped binaries

Stripped binaries do not run under Morello IE because of the lack of symbol names that are required for correct detection of C64 thunks. This is required for the C library emulation layer to function properly.

2.5.3 Use of non-Morello C library

On systems with no Morello C library, Hybrid and Purecap Morello applications can still be built and linked to an existing (non-Morello) C library. Using this method allows the creation of Morello applications that can be executed under Morello IE that emulate Morello-aware interface for an application.

Note: Functions from a non-Morello C library are executed in A64 mode and therefore do not have Morello functionality.

2.5.4 Inexact bounds

For a large allocation using `malloc` or a similar function, the returned capability may have bounds greater than the size requested, because Morello capabilities cannot exactly represent every possible combination of address and bounds.

Objects with inexact bounds allocated by the non-Morello `malloc` family of functions have their capability bounds set correctly and means that inexact bounds apply. However the size of the allocated memory is exactly as requested and the capability bounds might be wider than the allocated size. There is a risk that unrelated heap objects have overlapping bounds.

Use `-verbose` flag to display a message whenever inexact capability bounds are used.

2.5.5 Linking to Glibc with ifuncs

On systems with no Morello dynamic linker, Purecap applications linked statically to Glibc with ifuncs might not function properly. This is related to the lack of dynamic linker support for processing of relocation entries required for ifuncs to be loaded correctly. A workaround is to use another C library (for example, Musl libc[2]) instead of Glibc. See the `README.rst` in the `samples` directory.

2.5.6 CCTLR.SBL

Effects of `CCTLR.SBL = 1` are not fully supported in Purecap (C64) mode. The modifications to the AArch64 instructions that are related to `CCTLR.SBL` bit are not yet implemented. The behavior of the Morello instructions that are affected by the `CCTLR.SBL` bit value work as expected.

2.5.7 Multi-threaded applications

Support for multi-threaded applications based on `libpthread` is experimental.

[2] https://musl.libc.org/

CHAPTER

THREE

ADVANCED TOPICS

3.1 Building Morello applications

You can use the Morello LLVM toolchain to build Morello applications. See [Morello LLVM] for more details about using the toolchain. This section shows some examples of building and running such applications. More examples are available in `samples` directory (see `README.rst` in this folder).

The current version of the emulator can execute:

- statically linked Purecap Morello applications,
- statically and dynamically linked Hybrid Morello applications.

Note: See the [Morello AAPCS] for more details about the types of Morello applications and modes of execution.

Running an application under Morello IE generates normal application output to `stderr` and `stdout`, while the output of the emulator itself is always redirected to `stderr`.

In the following examples, the `clang` command refers to the C compiler from Morello LLVM toolchain.

3.1.1 Run a hello world example

The following example shows the simplest hello world example for a Morello application:

```
// hello.c
int printf(const char*, ...); // not including stdio.h
int main() {
    printf("hello Morello\n");
    return 0;
}
```

You can use most standard C library headers installed in your system to write source code of Morello applications. However, you must patch some of the headers (such as `libio.h`) to make them compatible with Morello.

You can work around this issue by not including headers that contain references to the files requiring modification (for example, `stdio.h`). Instead, you can declare the functions explicitly as shown in the example for the `printf` function.

To compile and link the hello world example, use the following command:

```
clang -march=morello+c64 -mabi=purecap -fuse-ld=lld -Wl,--morello-c64-plt hello.c -o hello -static
```

To run this example, use:

```
morelloie -- ./hello
```

Alternatively, you can run an application under Morello IE by using `launcher` binary explicitly:

```
launcher -c /path/to/lib/libmie.so -- ./hello
```

Note:

- When using Purecap (C64) mode, to enable Morello support, provide the options `-march=morello+c64` and `-mabi=purecap` for compilation and `-Wl,--morello-c64-plt` for linking.
- When using Hybrid (A64) mode, to enable Morello support, provide the option `-march=morello` for compilation and `-Wl,--morello-c64-plt` for linking.
- To link to the binary for Morello application, use the LLVM linker from the Morello toolchain: provide `-fuse-ld=lld` to invoke LLVM linker.
- To indicate that the binary must be statically linked, provide the `-static` option.

3.1.2 Stack corruption example

Out of bound writes to memory, allocated on stack, can modify content that is referenced by another variable. This process is also known as stack corruption. The following example demonstrates how a Morello application behaves when stack corruption is about to happen.

```
// stack.c
void fun(int *data) {
    data[3] = 3; // <--- access outside object bounds
}
int main() {
    int x = 0;
    int data[3] = {0, 1, 2};
    fun(data);
    return data[0];
}
```

Try running this example:

```
morelloie -- ./stack
Simulated capability fault: out of bounds access at 2342d4
Range [0000ffffdf2a81e8, +4] is not in bounds [0000ffffdf2a81dc, 0000ffffdf2a81e8]
Fault capability: 0x1:ffffc000:41e881dc:0000ffff:df2a81e8
         tag: true
       value: 0x 0000ffffdf2a81e8
      offset: 0x000000000000000c
        base: 0x00000ffffdf2a81dc
       limit: 0x00000ffffdf2a81e8
       valid: true
   in bounds: false
      length: 12
      offset: 12
 permissions: ...
      sealed: false
       flags: 0x 0000000000000000
Segmentation fault (core dumped)
```

This interrupts the execution and delivers `SIGSEGV` to the application due to out of bounds access.

```
2342d0: 68 00 80 52                  mov     w8, #3
2342d4: 08 0c 00 b9                  str     w8, [c0, #12] // <---
```

Note: Run this example with `-f-debug` flag to enter debug mode when out of bounds fault occurs to explore runtime context.

3.1.3 Out of bounds access example

The following example shows the behavior of a Morello application when out of bounds access occurs for dynamically allocated block of memory.

```
// heap.c
void *malloc(unsigned long size);
void free(void* ptr);
int main() {
    int *data = (int *)malloc(sizeof(int) * 3);
    int x = data[3]; // <--- reading outside of bounds
    free(data);
    return x;
}
```

To compile and link this example:

```
clang -march=morello+c64 -mabi=purecap -fuse-ld=lld heap.c -o heap -static
```

To run the example:

```
morelloie -- ./heap
Simulated capability fault: out of bounds access at 2342f8
Range [00000000002f0c8c, +4] is not in bounds [00000000002f0c80, 00000000002f0c8c]
Fault capability: 0x1:dc100000:4c8c0c80:00000000:002f0c8c
         tag: true
       value: 0x 00000000002f0c8c
      offset: 0x000000000000000c
        base: 0x000000000002f0c80
       limit: 0x000000000002f0c8c
       valid: true
   in bounds: false
...
Segmentation fault (core dumped)
```

3.1.4 Coremark and Dhrystone

For information on how to build Coremark and Dhrystone benchmarks, refer to the `samples` folder and the `README.rst` file in the distribution bundle. You must patch both applications for Morello to avoid including certain C library headers and to use the correct size of `void *` type.

The bundle includes the patches and a makefile which contains correct compilation and linking commands.

3.2 Capability faults

If a capability fault occurs during execution, the emulator prints:

- information about the fault
- the PC value where the fault occurred
- details about faulty capability.

Morello IE terminates the application with an appropriate OS signal. Use the `llvm-objdump` tool from the Morello toolchain to disassemble the binary and locate the instruction that caused the fault by using the reported PC value. Alternatively, use the interactive Morello debugger.

The following table shows which signals are emulated for each type of capability faults:

Fault	Signal
Capability tag not set	SIGABRT
Capability is sealed	SIGBUS
Incorrect capability permissions	SIGKILL
Access out of capability bounds	SIGSEGV
Anything else	SIGBUS

You can also provide `-f-debug` option to pause execution immediately after the fault to gather additional information (see *Interactive debugger* (page 15) about how to use interactive debugger).

Use `-f-cpu` option to show information about emulated CPU state when a capability fault occurs.

3.3 Instruction and memory trace

The tracer instrumentation client included in Morello IE allows the generation of runtime traces with information about executed instructions and memory accesses. This feature can be enabled by using `-trace` option or by providing `-mem` for only memory trace and `-instr` for just instruction trace.

By default, code that is invoked directly or indirectly from `main` function is traced when tracing is switched on. In case of a stripped binary, entire application code is traced.

Note: To ensure correctness of execution, code inside a region defined by load-exclusive and store-exclusive instructions is not instrumented and therefore will not be traced.

To change the scope of tracing, use one of the following mutually-exclusive options:

- To force tracing of all application code, use the `-all` flag.
- To enable use of tracer marker instructions (see *Markers for tracing* (page 9)), use the `-roi` flag.
- To trace only Purecap code (C64 code), use the `-c64` flag.
- To trace only code executed from the instruction at address `<pc1>` to the instruction at address `<pc2>` (inclusive), use a pair of options `-fr <pc1> -to <pc2>`.

Note: Provide the `<pc>` values as hexadecimal integer literals with `0x` prefix.

Tracing can be enabled and disabled using marker instructions (see *Markers for tracing* (page 9)):

```
#define __START_TRACE() asm volatile ("hint #0b1000000")
#define __STOP_TRACE()  asm volatile ("hint #0b1000001")
```

The following example shows trace output that is generated by Morello IE:

```
#    Seq PC     Mode   Encoding Opcode Operands
M      1 2342ad (C64): 028343ff sub    Cd=31 Cn=31 imm12=208 CapSubImm_LSL0
M      2 2342b1 (C64): 4285fbfd stp    Ct=29 Ct2=30 Rn=31 imm7=11 CapStorePairImmPre
#                      Transfer size   Address          Data
M        2342b1 (C64): MW32            0000ffffcdeb1f70 ffffc0000001 ... fffcdeb1f90
M        2342b1 (C64): CW02            0000ffffcdeb1f70 10
#        ...
M     15 2342e5 (C64): c2c838a5 scbnds Cd=5 Cn=5 imm6=16 CapSetBoundsImmNoS
A     16 2342e9 (C64): 2a1f03e8 orr    %wzr %wzr lsl $0x00 -> %w8
A     17 2342ed (C64): b9000008 str    %w8 -> (%x0)[4byte]
A        2342ed (C64): MW04            0000ffffcdeb1f6c 00000000
```

Every instruction is shown with a sequential number (seq), address (pc), mode of execution (A64 or C64), 32-bit encoding value, opcode, and instruction operands.

Instruction trace entry is printed before instruction is executed.

For memory accesses, additional information is provided when memory tracing is enabled:

- Write or read: `MW` or `MR` correspondingly.
- Transfer size in bytes: for example, `MW32` means write of 32 bytes and `MR04` means read of 4 bytes.
- Address used for memory access (64-bit value).
- Data which is loaded or stored.
- For loading and storing capabilities, tags are also shown in binary format. For example, `CW02` with data `10` means that 2 tags are written, one tag is `1` and the other tag is `0`.

A memory trace entry always follows the corresponding instruction trace entry. When an instruction fails to execute, a memory trace entry is not shown.

3.4 Interactive debugger

Morello IE includes a simple interactive debugger that provides access to emulated state and runtime context of the executed application.

3.4.1 Entering debug mode

To use the interactive debugger, the Morello emulation client must enter debug mode. There are several ways to do this.

A breakpoint can be inserted from command line by using `-break` option with the value of the PC address at which the breakpoint must be inserted.

```
morelloie -break 0x2342d4 -- ./app
```

Note: Provide the <pc> value for `-break` option as a hexadecimal integer literal with `0x` prefix.

During runtime instrumentation Morello IE inserts code to pause the execution and enter debug mode in which a user can submit commands to request information about the current execution context. When execution flow is about to reach an instruction at given address, the application stops and waits for user commands.

If the `-f-debug` options is used, every capability fault automatically results in entering debug mode. In this case, a faulty instruction has been attempted but has failed. A user can obtain information about current execution context, but execution is interrupted with an appropriate OS signal sent to the application as soon as the emulator leaves the debug mode.

Finally, a breakpoint marker instructions can be inserted in source code using this macro:

```
#define __MIE_DEBUG() asm volatile ("hint #0b1000100")
```

During runtime instrumentation for every such instruction Morello IE inserts code to pause the execution and enter debug mode. To enable this, use `-debug` flag on command line.

Note: The `-debug` flag is implied when either `-f-debug` or `-break` options are used.

3.4.2 Debugger commands

The interactive debugger has limited functionality with most of the supported commands similar to LLDB. In the following commands, <pc> refers to a PC address in form of a hexadecimal literal with or without `0x` prefix, <addr> refers to memory address in form of a hexadecimal literal with or without `0x` prefix, <reg> is a register name, and <sz> is the number of bytes to read from memory (a decimal integer literal).

- `m` – print help message.
- `q` – terminate application and exit.
- `c` – disable debugging and continue execution. Any existing breakpoints are ignored, and execution continues until the process exits or a capability fault occurs.
- `r` – run process until next breakpoint.
- `n` or `s` – step to the next instruction (step in).
- `finish` – run until exit from current function (step out).
- `until` <pc> – set new breakpoint address to <pc> and run until it.
- `bt` – show backtrace (last call shown first, see *Backtrace* (page 17) for details).
- `where` – show information about current and previous PC addresses.
- `cpu` – print CPU state (emulated and hardware registers).
- `p` <reg> – print current value of a register: `XSP`, `CSP`, `CLR`, `PCC`, `DDC`, `X0` to `X30`, `C0` to `C30`.
- `mem` <sz> <addr> – read <sz> bytes from memory address <addr> (1, 2, 4, 8 or 16 bytes only).
- `tag` <addr> – read memory tag for memory address <addr>.
- `cap` <addr> – display capability that can be loaded from address <addr>.
- `br list` – list breakpoints.
- `br set` <pc> – create a new breakpoint at <pc>.
- `br del` <pc> – delete existing breakpoint at <pc>.
- `br clear` – delete all breakpoints.
- `emul` – show emulation code for the current instruction.

3.4.3 Backtrace

Backtrace is a sequence of function calls with information about caller address, callee address, start address of the calling function, mode of execution (A64 or C64), and frame pointer. When available, a name of a function is also provided.

Consider the following example:

```
# Caller Mode  Frame pointer    Calling function
- 2342c0 (C64) 0000ffffc168df60 2342ac:main
- 234590 (A64) 0000ffffc168e0b0 2343b0:__libc_start_main
- 234188 (A64) 0000ffffc168e0b0 234160:_start
```

This backtrace shows three function calls: `_start` calls `__libc_start_main` which in turn calls `main`. Start addresses of the functions are `234160`, `2343b0`, and `2342ac`, respectively. In function `_start`, the call to `__libc_start_main` occurs at address `234188`, and the call of `main` from `__libc_start_main` occurs at `234590`. The current PC value is `2342c0` (which is the top-most element of the call stack). Finally, `_start` and `__libc_start_main` are executed in A64 mode, because they are part of non-Morello C library. However, `main` is executed in C64 mode, because it is part of the Purecap Morello application.

3.4.4 CPU state

The `cpu` debugger command shows current emulated and real CPU state which includes all general purpose capability registers and aliased AArch64 registers, some system registers available at EL0, and process ID and current thread ID information.

```
 # Reg   Emulated value                          Hardware value     Sync
 - C00 = 0x1:ffffc000:5f40df3c:0000ffff:c168df3c (0000ffffc168df3c) [=]
 - C01 = 0x1:97ffc000:40200000:00000000:492e0000 (00000000492e0000) [=]
 - C02 = 0x0:00000000:00000000:00000000:00000000 (0000ffffc168e0c8) [ ]
 - C03 = 0x0:00000000:00000000:00000000:00000000 (00000000002342ad) [ ]
...
 - C30 = 0x1:ffffc000:00010005:00000000:00234594 (0000000000234594) [=]
 - CSP = 0x1:ffffc000:00010005:0000ffff:c168de90 (0000ffffc168de90) [=]
 - PCC = 0x1:ffffc000:00010005:00000000:002342c1
 - CID_EL0         = 0x0:00000000:00000000:00000000:00000000
 - CTPIDR_EL0      = 0x0:00000000:00000000:00000000:00000000
 - CTPIDRRO_EL0    = 0x0:00000000:00000000:00000000:00000000
 - PSTATE.C64      = 1
 - PSTATE.NZCV     = 0000 (0110)
 - CCTLR.SBL       = 0
 - CCTLR.ADRDPB    = 0
 - CCTLR.DDCBO     = 0
 - CCTLR.PCCBO     = 0
 - <process id>    = 81557
 - <thread id>     = 81557 (main thread)
```

Both emulated and real values are displayed for registers `C0` to `C30` and `CSP`. The real 64-bit hardware value is shown in brackets after emulated 129-bit value. When lower 64 bits of emulated capability are equal to the real hardware value of the aliased register, the `[=]` symbol is shown on the right. This means that emulated and real values have been synchronised. However, this synchronisation might not always occur because synchronisation happens only when required for the next instruction to be executed.

If a capability register is used in the next instruction as a source register, its value shown by the debugger is architecturally valid and the lower 64 bits are the same as 64 bits of the value of the corresponding hardware register. This is indicated with `[=]` symbol. However, if a capability register has not yet been used as part of a destination or source operand, the emulated CPU state contains its initial value while the real hardware register might have a different value previously set by a non-Morello instruction. Capability registers that are used as destination operands get an architecturally correct value immediately after execution of the instruction and lower 64 bits of its value are immediately propagated to the corresponding hardware register.

More detailed information about each register can be printed using `p <reg>` command. For capability registers this gives detailed information about capability, for example:

```
C01 = 0x1:97ffc000:40200000:00000000:492e0000
         tag: true
       value: 0x 00000000492e0000
      offset: 0x0000000000000000
        base: 0x000000000492e0000
       limit: 0x000000000492e0020
       valid: true
   in bounds: true
      length: 32
      offset: 0
 permissions: global ... load
      sealed: false
      flags: 0x 0000000000000000
```

For hardware registers, the 64-bit value is printed.

3.5 Statistics and cache model

3.5.1 Statistics

The tracer instrumentation client included in Morello IE can collect runtime information about executed instructions and memory accesses including Morello-specific data such as memory tag operations:

- *Total instructions executed* (including *Morello* instructions and *C64* instructions).
- *Total writes to memory* (including *Morello* writes and *C64* writes, also including writes of each transfer size counted independently for 1, 2, 4, 8, 16, and 32 bytes).
- *Total reads from memory* (including *Morello* reads and *C64* reads, also including reads of each transfer size counted independently for 1, 2, 4, 8, 16, and 32 bytes).
- *Tag writes* of tags for capabilities in memory.
- *Tag reads* of tags for capabilities in memory.

Use `-full` flag to display all collected information. Without this flag a brief version of data is displayed.

Statistics gathering module can also be used to analyse non-Morello AArch64 applications.

Note: Use of flags controlling the scope of tracing, such as `-roi` or `-c64`, also affects gathering of statistics.

3.5.2 Cache model

The cache model included in Morello IE is part of the tracer instrumentation client. It can be used to collect memory accesses and gather data related to use of CPU cache. The cache model is based on `drcachesim` tool from DynamoRIO and has limited configurability. It has two levels with independent instruction and data cache at level 1. Size of each cache, size of cache lines, associativity of L1 and L2 caches can be adjusted using command line options, for example:

```
morelloie -cache \
-l1-i-size 1024 -l1-d-size $((32*1024)) -l2-size $((128*1024)) -- ./app
```

This example sets L1 i-cache size to 1 KiB, L1 d-cache size to 32 KiB and L2 cache size to 12b KiB (see *Options for cache model* (page 8) for all options controlling cache model).

Use `-full` flag to display all collected information. Without this flag a brief version of data is displayed.

The data returned form the cache model includes information such as about hits and misses at L1 and L2 levels, miss rates, and number of executed instructions. When running Morello applications, both AArch64 and Morello instructions and memory access are taken into account. The cache model can also be used to analyse non-Morello AArch64 applications.

Note: Use of flags controlling the scope of tracing, such as `-roi` or `-c64`, also affects data fed into cache model.

3.6 Verbose logging

When flag `-verbose` is used, Morello IE produces extra diagnostic messages that are redirected to `stderr`. This information includes:

- Processed capability relocations (`CapDesc` entries).
- Detected `C64` thunks which are necessary for calling A64 code from C64 code.
- Inexact bounds used for dynamic memory allocations.
- Relocated and cleared memory tags (for example, when using `realloc`, `memmove` or `free`).
- Various warnings.