ABI for the Arm® Architecture Advisory Note – SP must be 8-byte aligned on entry to AAPCS-conforming functions

2023Q3

Date of Issue: 6th October 2023

arm

1 Preamble

1.1 Abstract

This advisory note discusses a hitherto little noticed consequence of the ABI requirement for natural alignment for primitive data of size 1, 2, 4, and 8 bytes, and its implications for:

- Low level exception-handling code running on:
  - A and R profiles of version 7 of the Arm architecture.
  - Versions of the Arm architecture earlier than version 7.
- Code that might be entered directly through an Armv7M exception vector.
- Tool chains that generate such code.

1.2 Keywords

ABI for the Arm architecture, advisory note

1.3 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.

1.4 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.

1.5 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.

1.6 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 Licence section.

1.7 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.

1.8 Copyright

Copyright (c) 2006, 2007, 2009, 2018, 2021-2023, Arm Limited and its affiliates. All rights reserved.

Contents

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 Arm ABI 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 **Release** quality level.

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
0.01	28th February 2006	DRAFT for internal comment.
0.1	3rd March 2006	CONFIDENTIAL version for limited release.
1.0	20th March 2006	Open access version.
A	25th October 2007	Document renumbered (formerly GENC-007024 v1.0).
B	23rd October 2009	Updated the reference to the Arm ARM; reviewed use of terminology.
2018Q4	21st December 2018	Minor typographical fixes, updated links.
2021Q1	12th April 2021	• document released on Github • new Licence: CC-BY-SA-4.0 • new sections on Contributions, Trademark notice, and Copyright

2.2 References

This document refers to the following documents.

Ref	Document number / External URL	Title
AAPCS32		Procedure Call Standard for the Arm Architecture
Armv7ARM_M	Arm DDI 0403E	Arm DDI 0406: Arm Architecture Reference Manual Arm v7-A and Arm v7-R edition
Armv7ARM_AR	Arm DDI 0406C	Arm DDI 0403C: Armv7-M Architecture Reference Manual
Armv5ARM	Arm DDI 0100E, ISBN 0 201 737191	The Arm Architecture Reference Manual, 2nd edition, edited by David Seal, published by Addison-Wesley.

2.3 Terms and abbreviations

This advisory note uses the following terms and abbreviations.

AAPCS

Procedure Call Standard for the Arm Architecture

ABI

Application Binary Interface:

1. The specifications to which an executable must conform in order to execute in a specific execution environment. For example, the *Linux ABI for the Arm Architecture.*
2. A particular aspect of the specifications to which independently produced relocatable files must conform in order to be statically linkable and executable. For example, the [CPPABI32], the [RTABI32], the [CLIBABI32].

Q-o-I

Quality of Implementation – a quality, behavior, functionality, or mechanism not required by this standard, but which might be provided by systems conforming to it. Q-o-I is often used to describe the toolchain-specific means by which a standard requirement is met.

3 The Problem and How to Avoid it

3.1 The need to align SP to a multiple of 8 at conforming call sites

The Procedure Call Standard for the Arm Architecture [AAPCS32] requires primitive data types to be naturally aligned according to their sizes (for size = 1, 2, 4, 8 bytes). Doing otherwise creates more problems than it solves.

In return for preserving the natural alignment of data, conforming code is permitted to rely on that alignment. To support aligning data allocated on the stack, the stack pointer (SP) is required to be 8-byte aligned on entry to a conforming function. In practice this requirement is met if:

- At each call site, the current size of the calling function's stack frame is a multiple of 8 bytes.

  This places an obligation on compilers and assembly language programmers.

- SP is a multiple of 8 when control first enters a program.

  This places an obligation on authors of low level OS, RTOS, and runtime library code to align SP at all points at which control first enters a body of (AAPCS-conforming) code.

In turn, this requires the value of SP to be aligned to 0 modulo 8:

- By exception handlers, before calling AAPCS-conforming code.
- By OS/RTOS/run-time system code, before giving control to an application.

3.2 Possible consequences of SP misalignment

The possible consequences of not aligning SP properly depend on the architecture version and the characteristics of the code (and, hence on the behavior of the code generator). Possible consequences include:

- Alignment fault or UNPREDICTABLE behavior.
- Application failure.

3.2.1 Alignment fault or UNPREDICTABLE behavior

For architecture Armv5TE (in particular, for Intel XScale processors) and architecture Armv6 with CP15 register 1 A and U bits [Arm ARM, §G3.1, Unaligned access support] configured to emulate Armv5TE:

- An LDRD or STRD using a stack address presumed by a code generator to be 0 modulo 8, but actually 4 modulo 8, could cause an Alignment Fault or show UNPREDICTABLE behavior.

This failure cannot occur in code generated for architectures earlier than Armv5TE (no LDRD or STRD) or on processors conforming to architecture Armv7 or later (which cannot cause an alignment fault when the effective address of an LDRD or STRD is 4 modulo 8).

3.2.2 Application failure

An application failure might occur if SP is not 0 modulo 8 on entry to each AAPCS-conforming function and the program contains an interface such that:

- Code on one side of the interface evaluates the presumed alignment of an 8-byte aligned, stack allocated datum at compile time.
- Code on the other side of the interface evaluates the actual alignment of the datum at run time.

The interface defined by the C library's stdarg.h macros `va_start` and `va_arg` gives us a concrete example of how an application might fail.

- The compiler evaluates the presumed alignment of a parameter value passed to a variadic function at compile time. This determines whether to insert an additional padding word before an 8-byte aligned parameter value. Parameter values beyond the fourth word are passed to the callee via the stack and a variadic callee often pushes earlier parameter values onto the stack (to support uniform treatment of `va_list` types).
- Code generated by the `va_arg` macro evaluates the corresponding actual alignment at run time. This determines whether or not to skip a padding word preceding an 8-byte aligned parameter value.

A more cautious than usual implementation of `va_start` and `va_arg` can avoid this problem and operate correctly whether SP is 0 or 4 modulo 8 (§2.3.2.3).

In general, a compiler cannot detect whether similar code exists in an application. An application containing such code can fail if SP is not properly aligned.

3.3 Corrective steps

3.3.1 Operating systems and run-time environments

As stated in The need to align SP to a multiple of 8 at conforming call sites, operating systems and other run-time environments must ensure that SP is a multiple of 8 before calling AAPCS-conforming code. Alternatively the system must ensure that:

- The code it calls makes no use of 8-byte aligned, stack allocated data (see Safe option not to align SP).

  For example, an operating system might require that no 8-byte types be manipulated by exception handling code, and software development tools for that OS might support this proscription (Safe option not to align SP).
- If the architecture is V5TE or V6 configured to give V5TE alignment behavior, the compiler used to build the code must not have generated LDRD/STRD in place of a pair of LDR/STR to consecutive locations.

This requirement extends to operating systems and run-time code for all architecture versions prior to Armv7 and to the A, R and M architecture profiles thereafter. Special considerations associated with Armv7M are discussed in Special considerations for Cortex-M based applications.

3.3.2 Software development tools

3.3.2.1 Option to align SP on entry to designated functions

To support legacy execution environments in which SP is not properly aligned, compilers should offer an option to generate code to align SP to a multiple of 8 on entry to designated functions.

The means by which a function might be designated for special treatment is a quality of implementation (Q-o-I). Plausible means include the use of pseudo storage class specifiers like `__irq` or `__declspec(irq)`, or attributes like `__attribute__((irq))`, in a function's declaration.

3.3.2.2 Safe option not to align SP

To support safely not using the SP alignment option, compilers should offer an option (Q-o-I) to:

- Not generate LDRD/STRD.
- Fault the use of 8-byte aligned, stack allocated data.

  (8-byte aligned parameters to variadic functions need not be faulted if the toolchain implements the repair described in Repair of va_start and va_arg).
- Or, if that is too difficult, fault all uses of 8-byte data types.

A program that makes no us of LDRD/STRD cannot suffer the failure described in Alignment fault or UNPREDICTABLE behavior.

A program that makes no use of 8-byte aligned, stack allocated data cannot suffer the failure described in Application failure. And a program that makes no use 8-byte types certainly makes no use of 8-byte aligned, stack allocated data.

Assembly language programmers must, of course, certify the safety of their own code.

3.3.2.3 *Repair of* `va_start` *and* `va_arg`

To avoid injecting a fault into their users' programs in execution environments that do not correctly align SP, software development tools should offer an option (Q-o-I) to repair the C library's stdarg.h macros `va_start` and `va_arg`, as follows.

(We assume `va_start` expands to a call to the intrinsic function `__va_start`, and `va_arg` to a call to `__va_arg`. It is already very difficult – or impossible – to implement `va_start` and `va_arg` in a way that evaluates each argument only once – as required by the C standard – without the assistance of at least one intrinsic function).

`__va_start` should return a pointer value `ap` with bit[1] set if SP was 4 modulo 8 on entry to the containing function.

- The function containing the call to `__va_start` has the variadic parameter list allocated in the stack frame.
- Because arguments are guaranteed to be 4-byte aligned (by C's argument promotion rules and the AAPCS requirement that SP be 4-byte aligned at all instants), bits[1:0] of `ap` are otherwise 0.
- Coding the SP-misaligned case as `1` produces a `__va_start` compatible with an ordinary (not repaired) `__va_arg` in conforming environments in which SP is 0 modulo 8 at function entry.

If T is a data type requiring 8-byte alignment, `__va_arg(ap, T)` must increment the pointer it calculates by 4 bytes (to skip a padding word inserted at compile time) if:

```
(bit[1] of ap is 0 and bit[2] of ap is 1) or (bit[1] of ap is 1 and bit[2] of ap is 0).
```

Whatever the sort of T, `__va_arg(ap, T)` must clear bit 1 of the pointer it calculates before dereferencing it.

- This implementation of `__va_arg` is compatible with an ordinary (not repaired) `__va_start` in conforming environments in which SP is 0 modulo 8 at function entry and bit 1 of `ap` is always 0.

3.3.3 Special considerations for Cortex-M based applications

Armv7M is unique in making it possible (absent the problem discussed in this advisory note) to attach an AAPCS-conforming function directly to an exception vector.

(Under previous architecture versions and other architecture strands, some 'glue' code is required between an exception vector and an AAPCS-conforming function. Usually, an OS, RTOS, or run-time system provides this code. Considerations relating to such systems were discussed in Operating systems and run-time environments).

Cortex-M3 is the first implementation of Armv7M.

- Revision 0 of Cortex-M3 (CM3_r0) does not align SP to a multiple of 8 on entry to exceptions.

  Users of CM3_r0 must take appropriate precautions if the correctness of their software might depend on the alignment of stack-allocated data presumed by development tools to be 8-byte aligned.
- Revision 1 of Cortex-M3 will offer a configurable option to align SP to a multiple of 8 on entry to exceptions.
- A future revision of the M profile architecture will require SP to be 8-byte aligned on entry to exceptions.