DWARF for the Arm 64-bit Architecture (AArch64)

Release 2020Q2

Document number: IHI0057_**E**

Arm Ltd

Jun 12, 2020

IHI0057_E_2020Q2_00_en

CONTENTS

IHI0057_E_2020Q2_00_en

Document number: IHI 0057_E

Release version: 2020Q2

CHAPTER

ONE

PREAMBLE

1.1 Abstract

This document describes the use of the DWARF debug table format in the Application Binary Interface (ABI) for the Arm 64-bit architecture.

1.2 Keywords

DWARF, DWARF 3.0, use of DWARF format

1.3 How to find the latest release of this specification or report a defect in it

Please check the Arm Developer site (https://developer.arm.com/architectures/system-architectures/software-standards/abi) for a later release if your copy is more than one year old.

Please report defects in this specification to arm.eabi@arm.com.

1.4 Licence

THE TERMS OF YOUR ROYALTY FREE LIMITED LICENCE TO USE THIS ABI SPECIFICATION ARE GIVEN IN *Your licence to use this specification* (page 7) (Arm contract reference LEC-ELA-00081 V2.0). PLEASE READ THEM CAREFULLY.

BY DOWNLOADING OR OTHERWISE USING THIS SPECIFICATION, YOU AGREE TO BE BOUND BY ALL OF ITS TERMS. IF YOU DO NOT AGREE TO THIS, DO NOT DOWNLOAD OR USE THIS SPECIFICATION. THIS ABI SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES (SEE *Your licence to use this specification* (page 7) FOR DETAILS).

1.5 Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.
LES-PRE-20349

1.6 Contents

Contents

CHAPTER

TWO

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.

Content relating to SVE and Pointer Authentication should be considered as having a **Beta** support level. This includes:

- DWARF register names marked as **Beta** in *DWARF register names* (page 10)
- Call frame instructions (*Call frame instructions (Beta)* (page 12))
- DWARF expression operations (*DWARF expression operations (Beta)* (page 12))

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

2.1.2 Change history

Issue	Date	By	Change
00bet3	16th December 2010	MGD	Beta release.
1.0	22nd May 2013	RE	First public release.
2018Q4	31st December 2018	OS	Add SVE and pointer authentication support.
2019Q4	30th January 2020	TS	Minor layout changes.
2020Q2	1st June 2020	TS	Add requirements for unwinding MTE tagged stack. Describe DWARF representation of SVE vector types.

2.2 References

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

Ref	External reference or URL	Title
AADWARF (page 1)		DWARF for the Arm 64-bit Architecture (AArch64).
GDWARF[1]	http://dwarfstd.org/Dwarf3Std.php	DWARF 3.0, the generic debug table format.

2.3 Terms and abbreviations

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

A32 The instruction set named Arm in the Armv7 architecture; A32 uses 32-bit fixed-length instructions.

A64 The instruction set available when in AArch64 state.

AAPCS64 Procedure Call Standard for the Arm 64-bit Architecture (AArch64).

AArch32 The 32-bit general-purpose register width state of the Armv8 architecture, broadly compatible with the Armv7-A architecture.

AArch64 The 64-bit general-purpose register width state of the Armv8 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 C++ ABI for the Arm Architecture, ELF for the Arm Architecture, . . .

Arm-based . . . based on the Arm architecture . . .

Floating point Depending on context floating point means or qualifies: (a) floating-point arithmetic conforming to IEEE 754 2008; (b) the Armv8 floating point instruction set; (c) the register set shared by (b) and the Armv8 SIMD instruction set.

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 tool-chain-specific means by which a standard requirement is met.

MTE Memory Tagging Extension.

PAC Pointer Authentication Code.

PAUTH Pointer Authentication Extension.

SIMD Single Instruction Multiple Data – A term denoting or qualifying: (a) processing several data items in parallel under the control of one instruction; (b) the Arm v8 SIMD instruction set: (c) the register set shared by (b) and the Armv8 floating point instruction set.

SIMD and floating point The Arm architecture's SIMD and Floating Point architecture comprising the floating point instruction set, the SIMD instruction set and the register set shared by them.

SVE Scalable Vector Extension.

T32 The instruction set named Thumb in the Armv7 architecture; T32 uses 16-bit and 32-bit instructions.

[1] http://dwarfstd.org/Dwarf3Std.php

2.4 Your licence to use this specification

IMPORTANT: THIS IS A LEGAL AGREEMENT ("LICENCE") BETWEEN YOU (AN INDIVIDUAL OR SINGLE ENTITY WHO IS RECEIVING THIS DOCUMENT DIRECTLY FROM ARM LIMITED) ("LICENSEE") AND ARM LIMITED ("ARM") FOR THE SPECIFICATION DEFINED IMMEDIATELY BELOW. BY DOWNLOADING OR OTHERWISE USING IT, YOU AGREE TO BE BOUND BY ALL OF THE TERMS OF THIS LICENCE. IF YOU DO NOT AGREE TO THIS, DO NOT DOWNLOAD OR USE THIS SPECIFICATION.

"Specification" means, and is limited to, the version of the specification for the Applications Binary Interface for the Arm Architecture comprised in this document. Notwithstanding the foregoing, "Specification" shall not include (i) the implementation of other published specifications referenced in this Specification; (ii) any enabling technologies that may be necessary to make or use any product or portion thereof that complies with this Specification, but are not themselves expressly set forth in this Specification (e.g. compiler front ends, code generators, back ends, libraries or other compiler, assembler or linker technologies; validation or debug software or hardware; applications, operating system or driver software; RISC architecture; processor microarchitecture); (iii) maskworks and physical layouts of integrated circuit designs; or (iv) RTL or other high level representations of integrated circuit designs.

Use, copying or disclosure by the US Government is subject to the restrictions set out in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software – Restricted Rights at 48 C.F.R. 52.227-19, as applicable.

This Specification is owned by Arm or its licensors and is protected by copyright laws and international copyright treaties as well as other intellectual property laws and treaties. The Specification is licensed not sold.

1. Subject to the provisions of Clauses 2 and 3, Arm hereby grants to LICENSEE, under any intellectual property that is (i) owned or freely licensable by Arm without payment to unaffiliated third parties and (ii) either embodied in the Specification or Necessary to copy or implement an applications binary interface compliant with this Specification, a perpetual, non-exclusive, non-transferable, fully paid, worldwide limited licence (without the right to sublicense) to use and copy this Specification solely for the purpose of developing, having developed, manufacturing, having manufactured, offering to sell, selling, supplying or otherwise distributing products which comply with the Specification.
2. THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF SATISFACTORY QUALITY, MERCHANTABILITY, NONINFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE. THE SPECIFICATION MAY INCLUDE ERRORS. Arm RESERVES THE RIGHT TO INCORPORATE MODIFICATIONS TO THE SPECIFICATION IN LATER REVISIONS OF IT, AND TO MAKE IMPROVEMENTS OR CHANGES IN THE SPECIFICATION OR THE PRODUCTS OR TECHNOLOGIES DESCRIBED THEREIN AT ANY TIME.
3. This Licence shall immediately terminate and shall be unavailable to LICENSEE if LICENSEE or any party affiliated to LICENSEE asserts any patents against Arm, Arm affiliates, third parties who have a valid licence from Arm for the Specification, or any customers or distributors of any of them based upon a claim that a LICENSEE (or LICENSEE affiliate) patent is Necessary to implement the Specification. In this Licence; (i) "affiliate" means any entity controlling, controlled by or under common control with a party (in fact or in law, via voting securities, management control or otherwise) and "affiliated" shall be construed accordingly; (ii) "assert" means to allege infringement in legal or administrative proceedings, or proceedings before any other competent trade, arbitral or international authority; (iii) "Necessary" means with respect to any claims of any patent, those claims which, without the appropriate permission of the patent owner, will be infringed when implementing the Specification because no alternative, commercially reasonable, non-infringing way of implementing the Specification is known; and (iv) English law and the jurisdiction of the English courts shall apply to all aspects of this Licence, its interpretation and enforcement. The total liability of Arm and any of its suppliers and licensors under or in relation to this Licence shall be limited to the greater of the amount actually paid by LICENSEE for the Specification or US$10.00. The limitations, exclusions and disclaimers in this Licence shall apply to the maximum extent allowed

CHAPTER

THREE

OVERVIEW

The ABI for the Arm 64-bit architecture specifies the use of DWARF 3.0 format debugging data. For details of the base standard see GDWARF[2].

The ABI for the Arm 64-bit architecture gives additional rules for how DWARF 3.0 should be used, and how it is extended in ways specific to the Arm 64-bit architecture. The following topics are covered in detail:

- The enumeration of DWARF register numbers for using in `.debug_frame` and `.debug_info` sections (*DWARF register names* (page 10)).
- The definition of *Canonical Frame Address* (CFA) used by this ABI (*Canonical frame address* (page 11)).
- The definition of *Common Information Entries* (CIE) used by this ABI (*Common information entries* (page 11)).
- The definition of *Call Frame Instructions* (CFI) used by this ABI (*Call frame instructions (Beta)* (page 12)).
- The definition of DWARF Expression Operations used by this ABI (*DWARF expression operations (Beta)* (page 12)).

[2] http://dwarfstd.org/Dwarf3Std.php

CHAPTER

FOUR

ARM-SPECIFIC DWARF DEFINITIONS

4.1 DWARF register names

GDWARF[3], §2.6.1, Register Name Operators, suggests that the mapping from a DWARF register name to a target register number should be defined by the ABI for the target architecture. DWARF register names are encoded as unsigned LEB128 integers.

Table 4.1: Mapping from DWARF register numbers to Arm 64-bit architecture registers

DWARF register number	AArch64 register name	Description
0–30	X0–X30	64-bit general registers (*Note 1* (page 10))
31	SP	64-bit stack pointer
32	Reserved	-
33	ELR_mode	The current mode exception link register
34	RA_SIGN_STATE (**Beta**)	Return address signed state pseudo-register (*Note 8* (page 11))
37-45	Reserved	-
46	VG (**Beta**)	64-bit SVE vector granule pseudo-register (*Note 2* (page 10), *Note 3* (page 10))
47	FFR (**Beta**)	VG×8-bit SVE first fault register (*Note 4* (page 10))
48-63	P0-P15 (**Beta**)	VG×8-bit SVE predicate registers (*Note 4* (page 10))
64-95	V0-V31	128-bit FP/Advanced SIMD registers (*Note 5* (page 10), *Note 7* (page 11))
96-127	Z0-Z31 (**Beta**)	VG×64-bit SVE vector registers (*Note 6* (page 11), *Note 7* (page 11))

Notes

1. The size of a general register is to be taken from context. For instance in a `.debug_info` section if the `DW_AT_location` attribute of a variable is `DW_OP_reg0` then the number of significant bits in the register is determined by the variable's `DW_AT_type` attribute. If no context is available (for example in `.debug_frame` or `.eh_frame` sections) then the register number refers to a 64-bit register.
2. The value of the SVE vector granule pseudo-register is an even integer in the range 2 to 32. The value of the register is the available size in bits of the SVE vector registers in the current call frame divided by 64.
3. The SVE vector granule pseudo-register enables the construction of DWARF expressions that require the use of the current vector length, such as the location of saved SVE predicate and vector registers on the stack using the DWARF stack frame operator `DW_CFA_expression`.
4. The available size of a SVE predicate register and the first fault register is VG×8-bits.

[3] http://dwarfstd.org/Dwarf3Std.php

5. In a similar manner to the general register file the size of an FP/Advanced SIMD register is taken from some external context to the register number. If no context is available then only the least significant 64 bits of the register are referenced. In particular this means that the most significant part of a SIMD register is unrecoverable by frame unwinding.
6. The available size of the SVE vector registers is VG×64-bits.
7. The architecture defines that the FP/Advanced SIMD registers (V registers) overlap with the SVE vector registers (Z registers). A given V register is mapped to the low 128-bits of the corresponding Z register.

   The DWARF call frame instructions do not explicitly specify the size of a register; this is implicit in the definition of the register. As a consequence the V registers and Z registers have been allocated separate DWARF register number ranges which have their own definition for the size of these registers.

   When searching the call frame information table for either a V register or a Z register a consumer must take into account the aliasing between the V and Z registers.
8. The RA_SIGN_STATE pseudo-register records whether the return address has been signed with a PAC. This information can be used when unwinding. It is an unsigned integer with the same size as a general register. Only bit[0] is meaningful and is initialized to zero. A value of 0 indicates the return address has not been signed. A value of 1 indicates the return address has been signed.

4.2 Canonical frame address

The term Canonical Frame Address (CFA) is defined in GDWARF[4], §6.4, Call Frame Information.

This ABI adopts the typical definition of CFA given there:

> The CFA is the value of the stack pointer (sp) at the call site in the previous frame.

4.3 Common information entries

The DWARF virtual unwinding model is based, conceptually, on a tabular structure with one column for each target register (GDWARF[5], §6.4.1, Structure of Call Frame Information). A `.debug_frame` Common Information Entry (CIE) specifies the initial values (on entry to an associated function) of each register.

The variability of execution environments conforming to the Arm architecture creates a problem for this model. A producer cannot reliably enumerate all the registers in the target. For example, an integer-only function might be included in one executable file for use in execution environments with floating-point and another for use in environments without. In effect, it must be acceptable for a producer not to initialize, in a CIE, registers it does not know about. In turn this generates an obligation on consuming debuggers to default missing initial values.

This generates the following obligations on producers and consumers of CIEs:

1. Consumers must default the CIE initial value of any target register not mentioned explicitly in the CIE.
   - Callee-saved registers (and registers intentionally unused by the program, for example as a consequence of the procedure call standard) should be initialized as if by `DW_CFA_same_value`, other registers as if by `DW_CFA_undefined`.

     A debugger can use built-in knowledge of the procedure call standard or can deduce which registers are callee-saved by scanning all CIEs.
   - The VG pseudo-register should be initialized as if by `DW_CFA_same_value`.

[4] http://dwarfstd.org/Dwarf3Std.php
[5] http://dwarfstd.org/Dwarf3Std.php

  - The RA_SIGN_STATE pseudo-register should be initialized as described in *Section 3.3* (page 11).

2. To allow consumers to reliably default the initial values of missing entries by scanning a program's CIEs, without recourse to built-in knowledge, producers must identify registers not preserved by callees, as follows:

   - If a function uses any register from a particular hardware register class (e.g. Arm core registers), its associated CIE must initialize all the registers of that class that are not callee-saved to DW_CFA_undefined.

   - If a function uses a callee-saved register R, its associated CIE must initialize R using one of the defined value methods (not DW_CFA_undefined).

   (As an optimization, a producer need not initialize registers it can prove cannot be used by any associated functions and their descendants. Although these are not callee-saved, they are not callee-used either.)

This ABI defines two CIE augmentation characters that may appear as part of a CIE augmentation string.

1. The character 'B' indicates that associated frames are using the B key for return address signing.
2. The character 'G' indicates that associated frames may modify MTE tags on the stack space they use.

Notes

1. The mark on a frame recording that it may have set MTE tags other than the stack background is information which can be used when unwinding.

4.4 Call frame instructions (Beta)

This ABI defines one vendor call frame instruction DW_CFA_AARCH64_negate_ra_state.

Table 4.2: AArch64 vendor CFA operations

Instruction	High 2 bits	Low 6 bits	Operand 1	Operand 2
DW_CFA_AARCH64_negate_ra_state	0	0x2D	-	-

The DW_CFA_AARCH64_negate_ra_state operation negates bit[0] of the RA_SIGN_STATE pseudo-register. It does not take any operands.

4.5 DWARF expression operations (Beta)

This ABI defines one vendor DWARF expression operation DW_OP_AARCH64_operation.

Table 4.3: AArch64 vendor DWARF expression operations

Operation	Code
DW_OP_AARCH64_operation	0xea

The DW_OP_AARCH64_operation takes one mandatory operand encoded as an unsigned LEB128. Bits[6:0] of this value specify an AArch64 DWARF Expression sub-operation. The remaining operands and the action performed are as specified by the sub-operation. The DW_OP_AARCH64_operation allows this ABI to define operations specific to the Arm 64-bit architecture outside the encoding space of DWARF expression operations.

Table 4.4: AArch64 DWARF expression sub-operations

Sub-operation	Code
DW_SUB_OP_AARCH64_sign	0x00

The DW_SUB_OP_AARCH64_sign sub-operation takes a single operand encoded as an unsigned LEB128 operand. This value specifies a pointer key signing operation given in *AArch64 DWARF pointer signing operations* (page 13). The top two stack stack entries are popped, the first is treated as an 8-byte address value to be signed and the second is treated as an 8-byte salt. The key signing operation is performed on the address value using the salt, and the result is pushed to the stack.

Table 4.5: AArch64 DWARF pointer signing operations

Code	Operation
0x0	Sign Instruction address with Key A
0x1	Sign Instruction address with Key B
0x2	Sign data address with Key A
0x3	Sign data address with Key B
0x4	Sign address with Generic key

4.6 Vector types (Beta)

The recommended way of describing an Advanced SIMD or SVE vector type is to use an array type (DW_TAG_array_type) that has the GNU vector type attribute (DW_AT_GNU_vector, code 0x2107). The array index for these vectors has a lower bound of zero. For variable-length SVE vectors, the upper bound (DW_AT_upper_bound) or element count (DW_AT_count) is an expression based on the VG pseudo-register. For Advanced SIMD vectors and fixed-length SVE vectors, the upper bound or element count is constant.

For example, the recommended representation of the SVE type svfloat32_t is:

```
DW_TAG_array_type
  DW_AT_name("...")
  DW_AT_GNU_vector
  DW_AT_type(reference to float)
  DW_TAG_subrange_type
    DW_AT_upper_bound(expression=
      DW_OP_bregx(46, 0)
      DW_OP_lit2
      DW_OP_mul
      DW_OP_lit1
      DW_OP_minus)
```

if using DW_AT_upper_bound and:

```
DW_TAG_array_type
  DW_AT_name("...")
  DW_AT_GNU_vector
  DW_AT_type(reference to float)
  DW_TAG_subrange_type
    DW_AT_count(expression=
      DW_OP_bregx(46, 0)
      DW_OP_lit2
      DW_OP_mul)
```

if using DW_AT_count. Note that the zero lower bound is implicit for C and C++.