DWARF for the ARM® 64-bit Architecture (AArch64)

Document number: ARM IHI 0057B, current through AArch64 ABI release 1.0
Date of Issue: 22nd May 2013

Abstract

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

Keywords

DWARF, DWARF 3.0, use of DWARF format

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

Please check the *ARM Information Center* (http://infocenter.arm.com/) for a later release if your copy is more than 3 months old (navigate to the *Software Development Tools* section, *Application Binary Interface for the ARM Architecture* subsection).
Please report defects in this specification to *arm* dot *eabi* at *arm* dot *com*.

Licence

Proprietary notice

Contents

1 ABOUT THIS DOCUMENT

1.1 Change control

1.1.1 Current status and anticipated changes

This document's status is released. Clarifications, extensions and minor changes should be expected.

1.1.2 Change history

Issue	Date	By	Change
00bet3	16th December 2010	MGD	Beta release.
1.0	22nd May 2013	RE	First public release.

1.2 References

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

Ref	External reference or URL	Title
AADWARF	This document	DWARF for the ARM 64-bit Architecture (AArch64).
GDWARF	http://www.dwarfstd.org/	DWARF 3.0, the generic debug table format.

1.3 Terms and abbreviations

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

Term	Meaning
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.
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.
T32	The instruction set named *Thumb* in the ARMv7 architecture; T32 uses 16-bit and 32-bit instructions.

1.4 Your licence to use this specification

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 by applicable law.

ARM Contract reference LEC-ELA-00081 V2.0 AB/LS (9 March 2005)

1.5 Acknowledgements

2 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].

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 (§3.1).
- The definition of *Canonical Frame Address* (CFA) used by this ABI (§3.2).

3 ARM-SPECIFIC DWARF DEFINITIONS

3.1 DWARF register names

[GDWARF] §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.

DWARF register name	AArch64 register name	Description	See note
0-30	X0-X30	64-bit general registers	1
31	SP	64-bit stack pointer	
32	*Reserved*		
33	ELR_mode	The current mode exception link register	
34-63	*Reserved*		
64-95	V0-V31	128-bit FP/Advanced SIMD registers	2
96-127	*Reserved*		

Notes:

1. The size of 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. 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 the 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.

3.2 Canonical Frame Address

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

3.3 Common Information Entries

The DWARF virtual unwinding model is based, conceptually, on a tabular structure with one column for each target register ([GDWARF], *§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.

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.

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`.

  (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).

- 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`).