DWARF for the Arm® Architecture

2021Q1

Date of Issue: 12th April 2021

arm

1 Preamble

1.1 Abstract

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

1.2 Keywords

DWARF, DWARF 3.0, use of DWARF format

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) 2003-2007, 2012, 2018, 2020, 2021, 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
1.0	30th October 2003	First public release.
2.0	24th March 2005	Second public release.
2.01	6th October 2006	Added register numbers for VFP-v3 d0-d31 (DWARF register names).
2.02	5th May 2006	Minor corrections now that DWARF 3.0 is a standard; incompatible changes to the values of DW_AT_endianity (Describing other endian data) as a result.
A	25th October 2007	Document renumbered (formerly GENC-003533 v2.02).
B r2.09	30th November 2012	Common information entries: Clarify CIE descriptions of registers that are unused by intention of the user, for example as a consequence of the chosen procedure call standard.
2018Q4	21st December 2018	Minor typographical fixes, updated links.

Issue	Date	Change
2020Q4	21st December 2020	• document released on Github • new Licence: CC-BY-SA-4.0 • new sections on Contributions, Trademark notice, and Copyright • Add Thread ID register numbers
2021Q1	12th April 2021	• Delete duplicated TPIDRURO register entry in regiter number table. • Added PACBTI-M unwinding information.

2.2 References

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

Ref	External reference or URL	Title
AADWARF	Source for this document	DWARF for the Arm Architecture
BSABI32		ABI for the Arm Architecture (Base Standard)
GDWARF	http://dwarfstd.org/Dwarf3Std.php	DWARF 3.0, the generic debug table format.

2.3 Terms and abbreviations

The ABI for the Arm Architecture 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 AAELF32, RTABI32, ...

AEABI

(Embedded) ABI for the Arm architecture (this ABI...)

Arm-based

... based on the Arm architecture ...

core registers

The general purpose registers visible in the Arm architecture's programmer's model, typically r0-r12, SP, LR, PC, and CPSR.

EABI

An ABI suited to the needs of embedded, and deeply embedded (sometimes called free standing), applications.

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.

VFP

The Arm architecture's Floating Point architecture and instruction set. In this ABI, this abbreviation includes all floating point variants regardless of whether or not vector (V) mode is supported.

2.4 Acknowledgements

This specification has been developed with the active support of the following organizations. In alphabetical order: Arm, CodeSourcery, Intel, Metrowerks, Montavista, Nexus Electronics, PalmSource, Symbian, Texas Instruments, and Wind River.

3 Overview

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

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

- The enumeration of DWARF register-numbers for, use in .debug_frame sections (DWARF register names).
- How the machine state (Arm state versus Thumb state) is encoded in DWARF 3.0 line number tables (DWARF line number information (ISA field)).
- How to describe access to Arm architecture v6 other-endian data (Describing other endian data).
- The definition of Canonical Frame Address (CFA) used by this ABI (Canonical Frame Address).
- The generation and interpretation of debug frame Common Information Entries (Common information entries).

3.1 Miscellaneous obligations on producers of relocatable files

3.1.1 Support for stack unwinding

To support stack unwinding by debuggers, producers must always generate .debug_frame sections, even when:

- Not generating other debug tables.
- At high optimization levels.
- Assembling hand-written assembly language, if that code calls code compiled from C or C++.

3.1.2 The debugging illusion (not mandatory)

Ideally, a user of a C/C++ source language debugger would like the illusion of:

- Stepping through the source program sequence point (SP) by sequence point.
- Being able to inspect the program's state at any sequence point, and seeing there the state predicted by the source language semantics.

For the purpose of debugging illusion, we define an observation point (OP) to be a point at which a debugger may (meaningfully) inspect a program's state. Most sequence points are also observation points. In addition

- There is an OP just after each function call (at the pc value to which the call will return).
- There is no OP at the SP after evaluation of function arguments but before the function call.

A variable's scope extends from the point of declaration of the identifier to the end of the smallest enclosing block. A variable need not have a value everywhere in its scope – it may be initialized some way after its declaration.

When a user signals to a producer (by Q-o-I means) that it should favour quality of debugging over quality of generated code, the producer should strive (Q-o-I) to generate DWARF tables and code supporting this illusion. Specifically:

- A statement should describe the code between consecutive OPs.
- At each OP, every in-scope, initialized, source code variable should have a location (need not be in memory), and that location should hold the value predicted by the source language semantics.

It is not necessary to describe OPs in code the producer knows can never be executed (e.g. in `if(0){i++;}`).

4 Arm-specific DWARF definitions

4.1 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. Numbers 0-127 encode in 1 byte, 128-16383 in 2 bytes.

Mapping from DWARF register number to Arm architecture register number

DWARF register number	Arm core or co-processor registers	Description
0–15	R0–R15	Arm core integer registers
16–63	None	Obsolescent: 16–47 were previously used for both FPA and VFP registers (Note 1)
64–95	S0–S31	Legacy VFP-v2 use: D0–D15 alias S0, S2, … S30 (Note 1, Note 4)
96–103	F0–F7	Obsolescent: FPA registers 0-7 (Note 1)
104–111	wCGR0–wCGR7	Intel wireless MMX general purpose registers 0-7
	ACC0-ACC7	XScale accumulator register 0-7 (Note 2)
112–127	wR0–wR15	Intel wireless MMX data registers 0–15
128	SPSR	Current SPSR register
129	SPSR_FIQ	FIQ-mode SPSR
130	SPSR_IRQ	IRQ-mode SPSR
131	SPSR_ABT	ABT-mode SPSR

DWARF register number	Arm core or co-processor registers	Description
132	SPSR_UND	UND-mode SPSR
133	SPSR_SVC	SVC-mode SPSR
134–142	None	Reserved for future allocation
143	RA_AUTH_CODE	Return Address Authentication Code
144–150	R8_USR–R14_USR	User mode registers
151–157	R8_FIQ–R14_FIQ	Banked FIQ registers
158–159	R13_IRQ–R14_IRQ	Banked IRQ registers
160–161	R13_ABT–R14_ABT	Banked ABT registers
162–163	R13_UND–R14_UND	Banked UND registers
164–165	R13_SVC–R14_SVC	Banked SVC registers
166–191	None	Reserved for future allocation
192–199	wC0–wC7	Intel wireless MMX control register in co-processor 0–7
200–255	None	Reserved for future allocation
256-287	VFP-v3/Neon D0-D31	VFP-v3/Neon 64-bit register file (Note 4)
288-319	None	Reserved to VFP/Neon
320	TPIDRURO	PL0 Read-Only Software Thread ID register
321	TPIDRURW	PL0 Read/Write Software Thread ID register
322	TPIDPR	PL1 Software Thread ID register
323	HTPIDPR	Hyp Software Thread ID register
324-8191	None	Reserved for future allocation
8192–16383	Vendor co-processor	Unspecified vendor co-processor register (Note 3)

Note

1. In ADS toolkits, DWARF names 16–23 were used to represent FPA registers F0–F7 and 16-47 were used to represent VFP registers S0–S31. No application needs to use both numberings simultaneously but it can complicate decoding, so in RVDS new, non-overlapping, numbers 64-95 were allocated to VFP S0-S31. Debuggers that need to support legacy objects may need to handle both mappings.

2. Current implementations of the version 1 XScale Architecture specification implement only acc0, though eight such registers (acc0–acc7) are defined architecturally in co-processor 0. The version 2 specification defines the Wireless MMX co-processor in Arm co-processor slots 0 and 1. No system can contain both acc0 and MMX, so these numberings can overlap.

3. The vendor co-processor space is not specified by this ABI and should be used when there is unlikely to be a requirement for multiple vendors to support debugging such code. By using numbers in this space vendors can be sure that they will not conflict with future ABI allocations. If a set of co-processor registers is likely to be used directly from a high-level language and to require support of multiple toolkit vendors, then an application should be made to Arm for an allocation of a numbering in the reserved space.

4. The VFP-v3 and Neon architectures extend the register file to 32 64-bit registers, posing significant difficulties to extending the ABI v2.0 VFP encodings. There is no simple scheme using 1-byte register numbers that is compatible with the legacies. We have, therefore, introduced a new, simple, more precisely specified scheme using 2-byte register numbers. The new numbering scheme should also be used for VFP-v2.

The CPSR, VFP and FPA control registers are not allocated a numbering above. It is considered unlikely that these will be needed for producing a stack back-trace in a debugger.

4.1.1 VFP-v3 and Neon register descriptions

Architecturally, VFP-v3 and the Neon SIMD unit share a register file comprising 32 64-bit registers, D0-D31. Registers D0-D31 are described by DWARF register numbers 256-287. Register numbers 288-319 are reserved in case of future register file expansion.

DWARF registers 64-95 are obsolescent (and will become obsolete in the next major revision of the ABI for the Arm Architecture).

In DWARF terms:

- Register Dx is described as DW_OP_regx(256+x).
- Q registers Q0-Q15 are described by composing two D registers together.

  `Qx = DW_OP_regx(256+2x) DW_OP_piece(8) DW_OP_regx(256+2x+1) [DW_OP_piece(8)]`

  (Note that the final DW_OP_piece(8) can be omitted because the whole register is used. It is left in above for expositional clarity).
- S registers are described as bit-pieces of a register
  - `S[2x] = DW_OP_regx(256 + (x >> 1)) DW_OP_bit_piece(32, 0)`
  - `S[2x+1] = DW_OP_regx(256 + (x >> 1)) DW_OP_bit_piece(32, 32)`
- Neon Half-word lanes and byte lanes are described in a similar way to S registers.

Producers should use this new numbering scheme for VFP-v2 before the ABI-v2.0 scheme (S0-S31 → 64-95) is declared obsolete. Consumers should accept both numberings for as long as there are legacy binaries.

4.2 DWARF line number information (ISA field)

GDWARF §6.2.5.2 Standard Opcodes, item 12, DW_LNS_set_isa, describes a single unsigned LEB128 operand that denotes the instruction set architecture (ISA) at the location identified by the line number table entry. The value of the operand is determined by the ABI for the architecture (this specification).

Under the Arm architecture there are many instruction set versions and variants, but few instruction set states. Under this ABI, the ISA field corresponding to a particular program address denotes the instruction set state encoded by the CPSR when the pc contains that address.

DW_LNS_set_isa values for the Arm Architecture

Name	Value	Meaning
DW_ISA_UNKNOWN	0	I-set state not available or not recorded.
DW_ISA_ARM_thumb	1	T-bit will be set in the CPSR when pc contains this code address.
DW_ISA_ARM_arm	2	T-bit will be clear in the CPSR when pc contains this code address.
	Other	Reserved to the ABI for the Arm architecture.

4.3 Describing other endian data

Arm architecture version 6 allows programs to access data stored in the other byte order, either by executing REV* instructions, or by juggling the E bit in the PSR. Consequently, there is a need to describe in DWARF tables data that has been statically declared with a particular byte order.

This ABI mandates no particular way to describe the byte order of data manipulated by a programming language, but one could imagine a simple language extension like the following, or use of #pragma:

```
extern __big_endian T bx;     // bx contains big-endian data
extern __little_endian T lx;  // lx contains little-endian data
```

Usually, all data has the same byte order and this is recorded in the EI_DATA field of the header of the ELF file (as the value ELFDATA2MSB or ELFDATA2LSB).

To describe data that is explicitly declared big-endian or little-endian (by whatever means), use the DWARF 3.0 attribute `DW_AT_endianity` (0x65). It takes a single LEB128 constant argument value that is one of the following:

```
DW_END_default (= 0)
DW_END_big (= 1)              (Was 0 prior to the DWARF 3.0 standard)
DW_END_little (= 2)           (Was 1 prior to the DWARF 3.0 standard)
```

By default the Arm architecture is little endian, so `DW_END_default` should be interpreted as `DW_END_little`.

The `DW_AT_endianity` attributes can be attached to type entries as follows.

- Attached to a base type (GDWARF, §5.1, Base Type Entries), this attribute gives the byte order of the data described by the base type.

  If this order differs from the default byte order recorded in the containing ELF file, a debugger should reverse the order of the bytes it fetches or stores when accessing values of that base type.

- Attached to any other type (GDWARF, §5, Type Entries), this attribute indicates that the type was labeled explicitly (in some way) with the given byte order.

  When representing such a type across its user interface, a debugger should label the representation in some way that indicates it was declared with an explicit byte order. Some possible labels for big-endian follow.

  ```
  __big_endian T X;
  __declspec(big_endian) T X;
  T X __attribute__("big endian");
  #pragma arm_big_endian
  struct BigT { ... };
  #pragma no_arm_big_endian
  BigT X;
  ```

  Any such representation by a debugger is entirely quality of implementation.

4.4 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 (r13) at the call site in the previous frame.

4.5 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 processors 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 targets with VFP and another for targets 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).

4.6 Return Address Authentication Code

Functions, compiled with Return Address Authentication will need to keep a pointer authentication code, used to validate integrity of the return address upon function exit. The pseudo-register `RA_AUTH_CODE` records the authentication code. It is an unsigned integer with the same size as a general-purpose register. If the value of `RA_AUTH_CODE` is used to authenticate a return address, the authentication shall use CFA as a modifier. The DWARF Call Frame Information table (GDWARF, §6.4.1, Structure of Call Frame Information) can be extended with a column for RA_AUTH_CODE, as needed.