Release notes for the A64 Instruction Set Architecture for Arm A-profile Architecture

2023-09

Non-Confidential

Issue 01

109389_2023-09_01_en

Release notes for the A64 Instruction Set Architecture for Arm A-profile Architecture

Release information

Document history

Issue	Date	Confidentiality	Change
2023_09-01	29 September 2023	Non-Confidential	2023-09 release

Proprietary Notice

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 Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Release notes for the A64 Instruction Set Architecture for Arm A-profile Architecture (2023-09)

29 September 2023

Product Status

The information in this release covers multiple versions of the architecture. The content relating to different versions is given different quality ratings.

The information relating to the 2023 Extensions of the A-profile Architecture and FEAT_D128 of the 2022 Extensions is at Alpha quality. Alpha quality means that most major features of the specification are described in this release, but some features and details might be missing.

The information relating to the remainder of the 2022 Extensions of the A-profile Architecture and the rest of the Architecture is at Beta quality. Beta quality means that all major features of the specification are described, but some details might be missing.

Change history

This release introduces the 2023 Extensions of the A-profile Architecture.

The following changes are made to instruction descriptions:

- For the following instructions, the PSTATE.DIT Operational information is removed:
  - the SVE WHILE<cc> instructions.
  - the SME WHILE<cc> instructions.
  - the SME MOVT instructions that move data between general-purpose register and ZT0 instructions.
- The ST64B, ST64BV, and ST64BV0 instructions are renamed.
- The **CONSTRAINED UNPREDICTABLE** behavior of the following instructions is clarified:
  - LDCLRP, LDCLRPA, LDCLRPAL, LDCLRPL.
  - LDSETP, LDSETPA, LDSETPAL, LDSETPL.
  - RCWSCLRP, RCWSCLRPA, RCWSCLRPL, RCWSCLRPAL.
  - RCWCLRP, RCWCLRPA, RCWCLRPL, RCWCLRPAL.
  - RCWSSETP, RCWSSETPA, RCWSSETPL, RCWSSETPAL.
  - RCWSETP, RCWSETPA, RCWSETPL, RCWSETPAL.
  - RCWSSWPP, RCWSSWPPA, RCWSSWPPL, RCWSSWPPAL.
  - RCWSWPP, RCWSWPPA, RCWSWPPL, RCWSWPPAL.
  - SWPP, SWPPA, SWPPAL, SWPPL.

- The operation pseudocode for the LDIAPP instruction is corrected to call CreateAccDescLDAcqPC() instead of CreateAccDescAcqRel() to show that it has AcquirePC memory ordering semantics. WFI and WFE instructions are updated to be **CONSTRAINED UNPREDICTABLE** in Debug state.
- The titles of the GCSB DSYNC, PSB CSYNC, and TSB CSYNC instruction pages are changed.
- The DSB instruction descriptions of the behavior of variants with the nXS qualifier are removed. The DRPS instruction description is expanded.
- For the following instructions, the descriptions of the behavior with PSTATE.BTYPE values is made more consistent:
    - BRK.
    - BTI.
    - HLT.
    - PACIA, PACIA1716, PACIASP, PACIAZ, PACIZA.
    - PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZB.
- To improve clarity in operational pseudocode of Advanced SIMD and floating-point instructions, references to the FPCR[] accessor are replaced by the FPCR register name.

The following changes are made to the Shared Pseudocode

- The FPDot() pseudocode function is clarified to remove a duplicate code snippet.
- In the function AArch64.S1Walk(), a check is added to detect if the RegularStartLevel + SKL is greater than 3 and report a translation fault at level 0.
- The pseudocode function AArch64.S2CheckPermissions() is corrected to test the SCTLR_ELx.TCF field while checking for Permission fault due to the NoTagAccess attribute when FEAT_MTE_PERM is implemented.
- The functions AArch64.ReservedBreakpointType() and AArch32.ReservedBreakpointType() are updated to account for the changes in FEAT_Debugv8p9 to allow for more than 16 breakpoints when calculating whether the given breakpoint is context aware. The pseudocode function IsContextMatchingBreakpoint() is renamed to IsContextAwareBreakpoint(), and calls updated accordingly.
- The pseudocode function AddPAC() is updated to ignore bits <63:56> when TBI is set.
- The accessibility pseudocode for PMZR_EL0 is clarified by using the new ZeroPMUCounters(X[t,64]) function. A new pseudocode function ZeroPMUCounters() is defined to represent the action of a write to the PMZR_EL0 register. New functions {{GetPMUWriteMask()}} and {{GetPMUReadMask()}} are introduced, to return a mask of the PMU counters that are writable or readable at the current Exception level.
- The pseudocode function AArch64.VAIsOutOfRange() is updated to no longer apply the effect of walkparams.mtx to instruction addresses.
- IMPLEMENTATION_DEFINED string in the case of registers with op0=3 and CRn=11,15 is added in pseudocode function CheckTransactionalSystemAccess().
- The AddPac(), Auth(), and Strip() functions are corrected to account for the impact of FEAT_LVA3 on bit 55.
- The pseudocode for the FEAT_MOPS instructions has been substantially refactored.

Many simple clarifications and corrections are also present, but are too small to be listed here. Some minor formatting changes are suppressed and not highlighted in the diff output.

Known issues

All issues identified in the below list will be fixed in a future release.

- References to PSTATE.SPSel in MSR (immediate), SMSTART, and SMSTOP instructions will be corrected to PSTATE.SP.
- In SVE instructions PMOV (to vector) and PMOV (to predicate), the portion index being optional will be shown in the assembler syntax, and the description of will state that the portion index defaults to zero.
- For the following SME and SVE2.1 instructions, the feature conditions stated in the pseudocode are correct, but the feature conditions stated above the encoding diagrams are incomplete:
    - PSEL.
    - REVD.
    - BFMLSLB, BFMLSLT.
    - FDOT, SDOT (two-way), UDOT (two-way).
    - FCLAMP, SCLAMP, UCLAMP.
    - SQCVTN, SQCVTUN, SQRSHRN, SQRSHRUN, UQCVTN, UQRSHRN.
    - WHILEGE, WHILEGT, WHILEHI, WHILEHS, WHILELE, WHILELO, WHILELS, WHILELT (predicate pair).
- In the following SME2 instructions, the feature conditions stated in the pseudocode are correct, but the feature conditions stated above the encoding diagrams are incomplete:
    - SMLALL (multiple vectors), SMLALL (multiple and single vector).
    - SMLSLL (multiple vectors), SMLSLL (multiple and single vector).
    - UMLALL (multiple vectors), UMLALL (multiple and single vector).
    - UMLSLL (multiple vectors), UMLSLL (multiple and single vector).
    - SDOT (4-way, multiple and single vector), SDOT (4-way, multiple vectors).
    - UDOT (4-way, multiple and single vector), UDOT (4-way, multiple vectors).
    - FMLA (multiple and single vector), FMLA (multiple vectors).
    - FMLS (multiple and single vector), FMLS (multiple vectors).
    - ADD (array results, multiple and single vector), ADD (array results, multiple vectors), ADD (array accumulators).
    - SUB (array results, multiple and single vector), SUB (array results, multiple vectors), SUB (array accumulators).
    - FADD.
    - FSUB.
- The effect on the S2AP[1] bit in the translation table after the generation of a Permission fault under certain circumstances will be clarified.

- The position of the SynchronizeContext() call in the DRPSInstruction(), AArch64.ExceptionReturn(), and AArch64.TakeException() pseudocode functions will be corrected.
- The setting of FPEXC.DEX and FPEXC.TFV bits for an invalid FPSCR.Len and FPSCR.Stride, for an allocated CP10 or CP11 instruction is missing from pseudocode.
- The assembler syntax for MRS, MRRS, MSR, and MSRR instructions with unnamed registers will be clarified.

Limitations of Arm pseudocode

The pseudocode statements IMPLEMENTATION_DEFINED, SEE, **UNDEFINED**, and **UNPREDICTABLE** indicate behavior that differs from that indicated by the pseudocode being executed. If one of them is encountered:

- Earlier behavior indicated by the pseudocode is only specified as occurring to the extent required to determine that the statement is executed.
- No subsequent behavior indicated by the pseudocode occurs.

For more information, see Special statements in the Arm® Architecture Reference Manual for A-profile architecture (ARM DDI 0487).

The pseudocode descriptions have several limitations. These are mainly since, for clarity and brevity, the pseudocode is a sequential and mostly deterministic language.

These limitations include:

- Pseudocode does not describe the ordering requirements when an instruction generates multiple memory accesses. For a description of the ordering requirements on memory accesses, see External ordering constraints in the Arm® Architecture Reference Manual for A-profile architecture (ARM DDI 0487).
- Pseudocode does not describe the exact ordering requirements when a single floating-point instruction generates more than one floating-point exception and one or more of those floating-point exceptions is trapped. Combinations of floating-point exceptions in the Arm® Architecture Reference Manual for A-profile architecture (ARM DDI 0487) describes the exact rules.
- Note: There is no limitation in the case where all the floating-point exceptions are untrapped, because the pseudocode specifies the same behavior as the cross-referenced section.
- When a vector operation or another operation can perform via a concurrent set of operations that are not architecturally ordered, the pseudocode still presents a sequential deterministic detail.
- An exception can be taken during execution of the pseudocode for an instruction, either explicitly as a result of the execution of a pseudocode function such as Abort(), or implicitly, for example if an interrupt is taken during execution of an LDM instruction, pair load-store instructions, SVE vector instructions, Memory copy and memory set instructions etc.. If this happens, the pseudocode does not describe the extent to which the normal behavior of the instruction occurs. To determine that, see the descriptions of the exceptions in Handling exceptions that are taken to an Exception level using AArch32 and Definition of a precise exception and imprecise exception in the Arm® Architecture Reference Manual for A-profile architecture (ARM DDI 0487).

- Pseudocode does not describe the exact rules when an AArch32 instruction that generates any of the following fails its condition code check:
    - **UNDEFINED** instruction
    - Hyp trap.
    - Monitor trap.
    - Trap AArch64 exception.

In such cases, the **UNDEFINED** pseudocode statement or call to the applicable trap function lies inside the if ConditionPassed() then ... structure, either directly or in the EncodingSpecificOperations() function call, and so the pseudocode indicates that the instruction executes as a **NOP**.

- For the exact rules, see:
    - Conditional execution of undefined instructions.
    - EL2 configurable controls
    - EL3 configurable controls
    - Configurable instruction controls
- Where a significant aspect of the behavior is **IMPLEMENTATION DEFINED**, pseudocode may present only the declarations of the functions - the details of these functions is implementation defined.
- Pseudocode does not present the possible observability due to speculative execution.
- Pseudocode presents various details of the architecture - but does not show how the details can be combined to form a single definition. There are various aspects of such a detail that are also not presented. Notably:
    - Pseudocode does not show the details of fetching, decoding, and linking to instruction execution.
    - Pseudocode for combining the instruction shared decode, decode, and operation is not shown.
- Pseudocode presents all the architectural state as global state. The possible implications of multiple PEs or other components is not shown.
- Below details are either not shown or have noted limitations in the pseudocode:
    - Self-hosted trace and external trace
    - Pseudocode for modeling the register state or side effects. The accessibility details of a direct or external access such as traps etc are shown.
    - Generation of all architectural and micro-architectural Performance Monitoring Events. Note: Some architectural event generation is shown.
    - Construction of Statistical Profiling Extension records.
    - Behavior of instructions in Debug state when the behavior is **UNPREDICTABLE** is presented as if the instruction is executed identical to how it is when not in Debug state.
    - Activity Monitor Events and Counters.
    - System timer functionality.

    - Generic Interrupt Controller functionality.
    - External memory system.
    - External agents such as debugger.
    - PE behaviors that would lead to unrecoverable or uncontainable errors.
    - Where the behavior is **IMPLEMENTATION DEFINED** or **CONSTRAINED UNPREDICTABLE**, not all possibilities may be shown. Sometimes the pseudocode may present a simplified, but architecturally compatible view. In some situations, the possible behaviors may be outlined in a comment.
- The following architectural features are at Alpha quality and are above the limitations described below due to recency and completion of validation:
    - Halting Debug
    - Statistical Profiling Extension.
    - Perfomance Monitoring Events.

Potential Upcoming Changes

Potential upcoming improvements to the pseudocode are as follows:

- Clarify when pseudocode integer divide operator (DIV) is exact or if there is a remainder, how the rounding is done.