

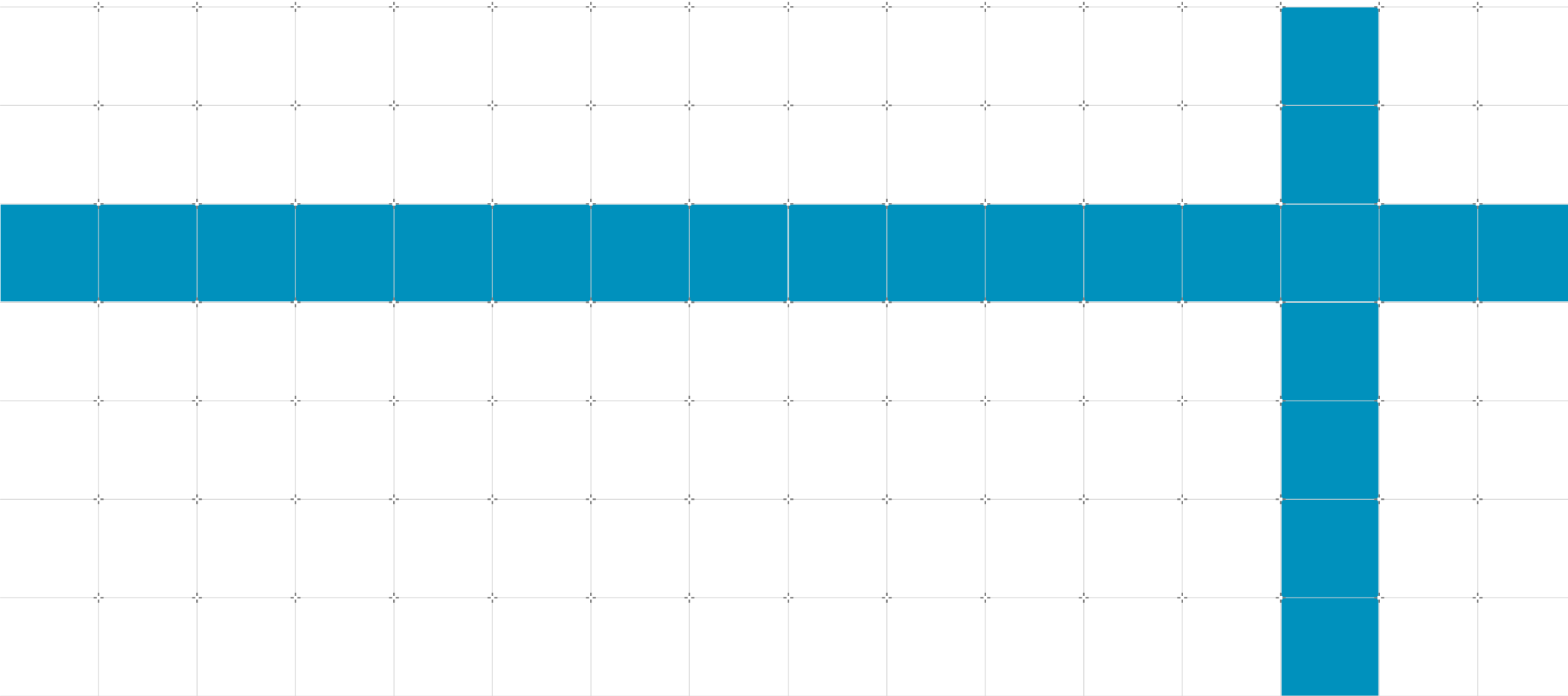


Arm Neon Intrinsics Reference

Release Date: 06 June 2025

Version: 2025Q2

Copyright: see section [Copyright](#).



Contents

1	Preface	12
1.1	Abstract	12
1.2	Latest release and defects report	12
1.3	License	12
1.3.1	About the license	12
1.4	Contributions	13
1.5	Trademark notice	13
1.6	Copyright	13
1.7	Document history	13
1.7.1	Changes between 2021Q2 and 2021Q3	14
1.7.2	Changes between 2021Q3 and 2021Q4	14
1.7.3	Changes between 2021Q4 and 2023Q2	15
1.7.4	Changes between 2023Q2 and 2024Q3	15
1.7.5	Changes between 2024Q3 and 2024Q4	15
1.7.6	Changes between 2024Q4 and 2025Q2	15
2	List of Intrinsic	16
2.1	Basic intrinsic	16
2.1.1	Vector arithmetic	16
2.1.1.1	Add	16
2.1.1.1.1	Addition	16
2.1.1.1.2	Widening addition	19
2.1.1.1.3	Narrowing addition	23
2.1.1.1.4	Saturating addition	31
2.1.1.2	Multiply	38
2.1.1.2.1	Multiplication	38
2.1.1.2.2	Multiply extended	41
2.1.1.2.3	Multiply-accumulate	44
2.1.1.2.4	Multiply-accumulate and widen	50
2.1.1.2.5	Fused multiply-accumulate	55
2.1.1.2.6	Saturating multiply	62
2.1.1.2.7	Saturating multiply-accumulate	65

2.1.1.2.8	Widening multiplication	72
2.1.1.2.9	Saturating multiply by scalar and widen	74
2.1.1.2.10	Saturating multiply-accumulate by scalar and widen	83
2.1.1.3	Polynomial	85
2.1.1.3.1	Polynomial multiply	85
2.1.1.4	Division	86
2.1.1.5	Subtract	86
2.1.1.5.1	Subtraction	86
2.1.1.5.2	Widening subtraction	90
2.1.1.5.3	Narrowing subtraction	93
2.1.1.5.4	Saturating subtract	99
2.1.1.6	Absolute	103
2.1.1.6.1	Absolute difference	103
2.1.1.6.2	Widening absolute difference	106
2.1.1.6.3	Absolute difference and accumulate	108
2.1.1.6.4	Widening absolute difference and accumulate	110
2.1.1.6.5	Absolute value	113
2.1.1.6.6	Saturating absolute value	115
2.1.1.6.7	Absolute minimum and maximum	116
2.1.1.7	Maximum	118
2.1.1.8	Minimum	121
2.1.1.9	Rounding	124
2.1.1.10	Reciprocal	128
2.1.1.10.1	Reciprocal estimate	128
2.1.1.10.2	Reciprocal square-root estimate	130
2.1.1.10.3	Reciprocal exponent	132
2.1.1.11	Square root	132
2.1.1.12	Pairwise arithmetic	133
2.1.1.12.1	Pairwise addition	133
2.1.1.12.2	Pairwise addition and widen	136
2.1.1.12.3	Pairwise maximum	139
2.1.1.12.4	Pairwise minimum	142
2.1.1.12.5	Pairwise maximum (IEEE754)	145
2.1.1.12.6	Pairwise minimum (IEEE754)	145
2.1.1.13	Across vector arithmetic	146
2.1.1.13.1	Addition across vector	146

2.1.1.13.2	Addition across vector widening	149
2.1.1.13.3	Maximum across vector	150
2.1.1.13.4	Minimum across vector	152
2.1.1.13.5	Maximum across vector (IEEE754)	154
2.1.1.13.6	Minimum across vector (IEEE754)	155
2.1.2	Compare	156
2.1.2.1	Bitwise equal	156
2.1.2.2	Bitwise equal to zero	160
2.1.2.3	Greater than or equal to	163
2.1.2.4	Less than or equal to	168
2.1.2.5	Greater than	174
2.1.2.6	Less than	179
2.1.2.7	Absolute greater than or equal to	184
2.1.2.8	Absolute less than or equal to	185
2.1.2.9	Absolute greater than	186
2.1.2.10	Absolute less than	187
2.1.2.11	Bitwise not equal to zero	188
2.1.3	Shift	192
2.1.3.1	Left	192
2.1.3.1.1	Vector shift left	192
2.1.3.1.2	Vector saturating shift left	197
2.1.3.1.3	Vector rounding shift left	206
2.1.3.1.4	Vector saturating rounding shift left	209
2.1.3.1.5	Vector shift left and widen	213
2.1.3.1.6	Vector shift left and insert	216
2.1.3.2	Right	221
2.1.3.2.1	Vector shift right	221
2.1.3.2.2	Vector rounding shift right	224
2.1.3.2.3	Vector shift right and accumulate	227
2.1.3.2.4	Vector rounding shift right and accumulate	230
2.1.3.2.5	Vector shift right and narrow	234
2.1.3.2.6	Vector saturating shift right and narrow	236
2.1.3.2.7	Vector saturating rounding shift right and narrow	241
2.1.3.2.8	Vector rounding shift right and narrow	245
2.1.3.2.9	Vector shift right and insert	248
2.1.4	Data type conversion	253

2.1.4.1	Conversions	253
2.1.4.2	Reinterpret casts	266
2.1.5	Move	316
2.1.5.1	Narrow	316
2.1.5.2	Widen	318
2.1.5.3	Saturating narrow	319
2.1.6	Scalar arithmetic	323
2.1.6.1	Vector multiply-accumulate by scalar	323
2.1.6.2	Vector multiply-subtract by scalar	332
2.1.6.3	Vector multiply by scalar	339
2.1.6.4	Vector multiply by scalar and widen	347
2.1.6.5	Vector multiply-accumulate by scalar and widen	351
2.1.6.6	Fused multiply-accumulate by scalar	357
2.1.7	Logical	358
2.1.7.1	Negate	358
2.1.7.2	Saturating Negate	360
2.1.7.3	Bitwise NOT	362
2.1.7.4	AND	363
2.1.7.5	OR	366
2.1.7.6	Exclusive OR	368
2.1.7.7	OR-NOT	371
2.1.8	Bit manipulation	373
2.1.8.1	Count leading sign bits	373
2.1.8.2	Count leading zeros	375
2.1.8.3	Population count	376
2.1.8.4	Bitwise clear	377
2.1.8.5	Bitwise select	380
2.1.9	Vector manipulation	385
2.1.9.1	Copy vector lane	385
2.1.9.2	Reverse bits within elements	397
2.1.9.3	Create vector	398
2.1.9.4	Set all lanes to the same value	400
2.1.9.5	Combine vectors	414
2.1.9.6	Split vectors	416
2.1.9.7	Extract one element from vector	420
2.1.9.8	Extract vector from a pair of vectors	428

2.1.9.9	Reverse elements	434
2.1.9.10	Zip elements	439
2.1.9.11	Unzip elements	449
2.1.9.12	Transpose elements	459
2.1.9.13	Set vector lane	468
2.1.9.14	Unzip elements'	474
2.1.10	Load	475
2.1.10.1	Stride	475
2.1.10.2	Load	563
2.1.11	Store	564
2.1.11.1	Stride	564
2.1.11.2	Store	637
2.1.12	Table lookup	637
2.1.12.1	Table lookup	637
2.1.12.2	Extended table lookup	648
2.1.12.3	Lookup table read with 2-bit indices	658
2.1.12.4	Lookup table read with 4-bit indices	665
2.2	Crypto	669
2.2.1	Cryptography	669
2.2.1.1	AES	669
2.2.1.2	SHA1	670
2.2.1.3	SHA256	671
2.2.2	Vector arithmetic	672
2.2.2.1	Polynomial	672
2.2.2.1.1	Polynomial multiply	672
2.2.2.1.2	Polynomial addition	672
2.3	CRC32	674
2.3.1	Cryptography	674
2.3.1.1	CRC32	674
2.4	sqrddmlah intrinsics (From ARMv8.1-A)	675
2.4.1	Vector arithmetic	675
2.4.1.1	Multiply	675
2.4.1.1.1	Saturating multiply-accumulate	675
2.4.1.1.2	Saturating multiply-accumulate by element	678
2.5	fp16 scalar intrinsics (available through <arm_fp16.h> from ARMv8.2-A)	683
2.5.1	Vector arithmetic	683

2.5.1.1	Absolute	683
2.5.1.1.1	Absolute value	683
2.5.1.1.2	Absolute difference	683
2.5.1.2	Reciprocal	684
2.5.1.2.1	Reciprocal estimate	684
2.5.1.2.2	Reciprocal square-root estimate	684
2.5.1.2.3	Reciprocal step	685
2.5.1.3	Rounding	685
2.5.1.4	Square root	686
2.5.1.5	Add	686
2.5.1.5.1	Addition	686
2.5.1.6	Division	687
2.5.1.7	Maximum	687
2.5.1.8	Minimum	688
2.5.1.9	Multiply	688
2.5.1.9.1	Multiplication	688
2.5.1.9.2	Multiply extended	689
2.5.1.9.3	Fused multiply-accumulate	689
2.5.1.10	Subtract	689
2.5.1.10.1	Subtraction	689
2.5.2	Compare	690
2.5.2.1	Bitwise equal to zero	690
2.5.2.2	Greater than or equal to zero	690
2.5.2.3	Greater than zero	690
2.5.2.4	Less than or equal to zero	691
2.5.2.5	Less than zero	691
2.5.2.6	Absolute greater than or equal to	691
2.5.2.7	Absolute greater than	692
2.5.2.8	Absolute less than or equal to	692
2.5.2.9	Absolute less than	692
2.5.2.10	Equal to	693
2.5.2.11	Greater than or equal to	693
2.5.2.12	Greater than	693
2.5.2.13	Less than or equal to	694
2.5.2.14	Less than	694
2.5.3	Data type conversion	694

2.5.3.1	Conversions	694
2.5.4	Logical	700
2.5.4.1	Negate	700
2.6	fp16 vector intrinsics (from ARMv8.2-A)	701
2.6.1	Vector arithmetic	701
2.6.1.1	Absolute	701
2.6.1.1.1	Absolute value	701
2.6.1.1.2	Absolute difference	701
2.6.1.2	Reciprocal	702
2.6.1.2.1	Reciprocal estimate	702
2.6.1.2.2	Reciprocal square-root estimate	702
2.6.1.2.3	Reciprocal step	703
2.6.1.3	Rounding	704
2.6.1.4	Square root	705
2.6.1.5	Add	706
2.6.1.5.1	Addition	706
2.6.1.6	Division	706
2.6.1.7	Maximum	707
2.6.1.8	Minimum	708
2.6.1.9	Multiply	709
2.6.1.9.1	Multiplication	709
2.6.1.9.2	Multiply extended	711
2.6.1.9.3	Fused multiply-accumulate	713
2.6.1.10	Pairwise arithmetic	717
2.6.1.10.1	Pairwise addition	717
2.6.1.10.2	Pairwise maximum	718
2.6.1.10.3	Pairwise minimum	719
2.6.1.11	Subtract	720
2.6.1.11.1	Subtraction	720
2.6.2	Compare	720
2.6.2.1	Bitwise equal to zero	720
2.6.2.2	Greater than or equal to zero	720
2.6.2.3	Greater than zero	721
2.6.2.4	Less than or equal to zero	721
2.6.2.5	Less than zero	722
2.6.2.6	Absolute greater than or equal to	722

2.6.2.7	Absolute greater than	723
2.6.2.8	Absolute less than or equal to	723
2.6.2.9	Absolute less than	724
2.6.2.10	Equal to	724
2.6.2.11	Greater than or equal to	725
2.6.2.12	Greater than	725
2.6.2.13	Less than or equal to	726
2.6.2.14	Less than	726
2.6.3	Data type conversion	727
2.6.3.1	Conversions	727
2.6.4	Logical	731
2.6.4.1	Negate	731
2.7	Additional intrinsics added in ACLE 3.0 for data processing (Always available)	731
2.7.1	Bit manipulation	731
2.7.1.1	Bitwise select	731
2.7.2	Vector manipulation	732
2.7.2.1	Zip elements	732
2.7.2.2	Unzip elements	733
2.7.2.3	Transpose elements	734
2.7.2.4	Set all lanes to the same value	735
2.7.2.5	Extract vector from a pair of vectors	737
2.7.2.6	Reverse elements	737
2.7.3	Move	738
2.7.3.1	Vector move	738
2.8	Dot Product intrinsics added for ARMv8.2-a and newer. Requires the +dotprod architecture extension.	738
2.8.1	Vector arithmetic	738
2.8.1.1	Dot product	738
2.9	Armv8.4-a intrinsics.	741
2.9.1	Cryptography	741
2.9.1.1	SHA512	741
2.9.1.2	SM3	742
2.9.1.3	SM4	744
2.9.2	Logical	744
2.9.2.1	Exclusive OR	744
2.9.2.2	Rotate and exclusive OR	746
2.9.2.3	Exclusive OR and rotate	746

2.9.2.4 Bit clear and exclusive OR	747
2.10 FP16 Armv8.4-a	749
2.10.1 Vector arithmetic	749
2.10.1.1 Multiply	749
2.10.1.1.1 Fused multiply-accumulate	749
2.11 Complex operations from Armv8.3-a	754
2.11.1 Complex arithmetic	754
2.11.1.1 Complex addition	754
2.11.1.2 Complex multiply-accumulate	756
2.11.1.3 Complex multiply-accumulate by scalar	759
2.12 Floating-point rounding intrinsics from Armv8.5-A	767
2.12.1 Vector arithmetic	767
2.12.1.1 Rounding	767
2.13 Matrix multiplication intrinsics from Armv8.6-A	769
2.13.1 Vector arithmetic	769
2.13.1.1 Matrix multiply	769
2.13.1.2 Dot product	770
2.14 Bfloat16 intrinsics Requires the +bf16 architecture extension.	772
2.14.1 Vector manipulation	772
2.14.1.1 Create vector	772
2.14.1.2 Set all lanes to the same value	772
2.14.1.3 Combine vectors	774
2.14.1.4 Split vectors	774
2.14.1.5 Set vector lane	775
2.14.1.6 Copy vector lane	775
2.14.2 Load	776
2.14.2.1 Stride	776
2.14.3 Store	783
2.14.3.1 Stride	783
2.14.4 Data type conversion	788
2.14.4.1 Reinterpret casts	788
2.14.4.2 Conversions	795
2.14.5 Vector arithmetic	796
2.14.5.1 Dot product	796
2.14.5.2 Matrix multiply	797
2.14.5.3 Multiply	798

2.14.5.3.1	Multiply-accumulate	798
2.14.6	Scalar arithmetic	798
2.14.6.1	Vector multiply-accumulate by scalar	798
2.15	Modal 8-bit floating-point intrinsics	800
2.15.1	Data type conversion	800
2.15.1.1	Conversions	800
2.15.2	Vector arithmetic	802
2.15.2.1	Exponent	802
2.15.2.2	Dot product	803
2.15.2.3	Multiply	807
2.15.2.3.1	Multiply-accumulate and widen	807

1. Preface

1.1 Abstract

This document is complementary to the main Arm C Language Extensions (ACLE) specification, which can be found on the [ACLE project on GitHub](#).

1.2 Latest release and defects report

For the latest release of this document, see the [ACLE project on GitHub](#).

Please report defects in this specification to the [issue tracker page on GitHub](#).

1.3 License

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.

1.3.1 About the license

As identified more fully in the [License](#) 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 (for example, 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.

1.4 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 [License](#) section.

We do not require copyright assignment. The original contributor will retain the copyright.

1.5 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.

1.6 Copyright

- Copyright 2014-2024 Arm Limited and/or its affiliates open-source-office@arm.com
- Copyright 2021 Matt P. Dziubinski matdzb@gmail.com

1.7 Document history

Issue	Date	Change
A	09 May 2014	First release
B	24 March 2016	Updated for ARMv8.1
C	30 March 2019	Version ACLE Q1 2019

Copyright: see section [Copyright](#).

Issue	Date	Change
D	30 June 2019	Version ACLE Q2 2019
E	30 Sept 2019	Version ACLE Q3 2019
F	30 May 2020	Version ACLE Q2 2020
G	30 October 2020	Version ACLE Q3 2020
H	02 July 2021	2021Q2
I	30 September 2021	2021Q3
J	11 January 2022	2021Q4
K	04 August 2023	2023Q2
L	30 September 2024	2024Q3
M	21 February 2025	2024Q4
N	06 June 2025	2025Q2

1.7.1 Changes between 2021Q2 and 2021Q3

- Fixed the guard macro for the base intrinsics.
- Corrected sdot, udot and usdot specification on AArch32.

1.7.2 Changes between 2021Q3 and 2021Q4.

- Fixed typo in signature of vaddq_s16.
- Updated copyright statement in section [Copyright](#).

- Converted document sources from reStructuredText (.rst) to Markdown (.md). The tool [pandoc](#) is now used to render the PDF of the specs. The PDF is rendered using the standard layout used in Arm specifications.
- Added missing item for release 2021Q3 in the table with the list of versions in section [Document history](#).

1.7.3 Changes between 2021Q4 and 2023Q2

- Added support for FEAT_LRCPC3 LDAP1/STL1 intrinsics.

1.7.4 Changes between 2023Q2 and 2024Q3

- Fixed the range of the lane immediate argument for vst2q_lane_f64.

1.7.5 Changes between 2024Q3 and 2024Q4

- Added mf8 forms of the vbsl, vluti2 and vluti4 families of intrinsics.

1.7.6 Changes between 2024Q4 and 2025Q2

- Added fp8 version of the vget_lane intrinsic.

2. List of Intrinsics

2.1 Basic intrinsics

The intrinsics in this section are guarded by the macro `__ARM_NEON`.

2.1.1 Vector arithmetic

2.1.1.1 Add

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vadd_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ADD Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vaddq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ADD Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vadd_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>ADD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vaddq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ADD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vadd_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>ADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vaddq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vadd_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>ADD Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vaddq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ADD Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vadd_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ADD Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vaddq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ADD Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vadd_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>ADD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vaddq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ADD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vadd_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>ADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vaddq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vadd_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>ADD Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vaddq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ADD Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>float32x2_t vadd_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vaddq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x1_t vadd_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FADD Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vaddq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FADD Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int64_t vadd_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>ADD Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64_t vadd_u64(uint64_t a, uint64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>ADD Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64

2.1.1.1.1 Addition

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vaddl_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SADDL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vaddl_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SADDL Vd.4S, Vn.4H, Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vaddl_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SADDL Vd.2D, Vn.2S, Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint16x8_t vaddl_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UADDL Vd.8H, Vn.8B, Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vaddl_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UADDL Vd.4S, Vn.4H, Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vaddl_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UADDL Vd.2D, Vn.2S, Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int16x8_t vaddl_high_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SADDL2 Vd.8H, Vn.16B, Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vaddl_high_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SADDL2 Vd.4S, Vn.8H, Vm.8H</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vaddl_high_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SADDL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vaddl_high_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UADDL2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vaddl_high_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UADDL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vaddl_high_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UADDL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>int16x8_t vaddw_s8(int16x8_t a, int8x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8B</code>	<code>SADDW Vd.8H,Vn.8H,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vaddw_s16(int32x4_t a, int16x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4H</code>	<code>SADDW Vd.4S,Vn.4S,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vaddw_s32(int64x2_t a, int32x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2S</code>	<code>SADDW Vd.2D,Vn.2D,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vaddw_u8(uint16x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8B</code>	<code>UADDW Vd.8H,Vn.8H,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vaddw_u16(uint32x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4H</code>	<code>UADDW Vd.4S,Vn.4S,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vaddw_u32(uint64x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2S</code>	<code>UADDW Vd.2D,Vn.2D,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int16x8_t vaddw_high_s8(int16x8_t a, int8x16_t b)</code>	<code>a -> Vn.8H b -> Vm.16B</code>	<code>SADDW2 Vd.8H,Vn.8H,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vaddw_high_s16(int32x4_t a, int16x8_t b)</code>	<code>a -> Vn.4S b -> Vm.8H</code>	<code>SADDW2 Vd.4S,Vn.4S,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vaddw_high_s32(int64x2_t a, int32x4_t b)</code>	<code>a -> Vn.2D b -> Vm.4S</code>	<code>SADDW2 Vd.2D,Vn.2D,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vaddw_high_u8(uint16x8_t a, uint8x16_t b)</code>	<code>a -> Vn.8H b -> Vm.16B</code>	<code>UADDW2 Vd.8H,Vn.8H,Vm.16B</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vaddw_high_u16(uint32x4_t a, uint16x8_t b)</code>	<code>a -> Vn.4S b -> Vm.8H</code>	<code>UADDW2 Vd.4S,Vn.4S,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vaddw_high_u32(uint64x2_t a, uint32x4_t b)</code>	<code>a -> Vn.2D b -> Vm.4S</code>	<code>UADDW2 Vd.2D,Vn.2D,Vm.4S</code>	<code>Vd.2D -> result</code>	A64

2.1.1.1.2 Widening addition

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vhadd_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SHADD Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vhaddq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SHADD Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vhadd_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SHADD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vhaddq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SHADD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vhadd_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SHADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vhaddq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SHADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vhadd_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UHADD Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vhaddq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UHADD Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vhadd_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UHADD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vhaddq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UHADD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vhadd_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UHADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vhaddq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UHADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int8x8_t vrhadd_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SRHADD Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vrhaddq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SRHADD Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vrhadd_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SRHADD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vrhaddq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SRHADD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vrhadd_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SRHADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vrhaddq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SRHADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vrhadd_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>URHADD Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vrhaddq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>URHADD Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vrhadd_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>URHADD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vrhaddq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>URHADD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vrhadd_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>URHADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vrhaddq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>URHADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vaddhn_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ADDHN Vd.8B,Vn.8H,Vm.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vaddhn_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ADDHN Vd.4H,Vn.4S,Vm.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vaddhn_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ADDHN Vd.2S,Vn.2D,Vm.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vaddhn_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ADDHN Vd.8B,Vn.8H,Vm.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vaddhn_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ADDHN Vd.4H,Vn.4S,Vm.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vaddhn_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ADDHN Vd.2S,Vn.2D,Vm.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vaddhn_high_s16(int8x8_t r, int16x8_t a, int16x8_t b)</code>	<code>r -> Vd.8B a -> Vn.8H b -> Vm.8H</code>	<code>ADDHN2 Vd.16B,Vn.8H,Vm.8H</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vaddhn_high_s32(int16x4_t r, int32x4_t a, int32x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4S b -> Vm.4S</code>	<code>ADDHN2 Vd.8H,Vn.4S,Vm.4S</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vaddhn_high_s64(int32x2_t r, int64x2_t a, int64x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2D b -> Vm.2D</code>	<code>ADDHN2 Vd.4S,Vn.2D,Vm.2D</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vaddhn_high_u16(uint8x8_t r, uint16x8_t a, uint16x8_t b)</code>	<code>r -> Vd.8B a -> Vn.8H b -> Vm.8H</code>	<code>ADDHN2 Vd.16B,Vn.8H,Vm.8H</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vaddhn_high_u32(uint16x4_t r, uint32x4_t a, uint32x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4S b -> Vm.4S</code>	<code>ADDHN2 Vd.8H,Vn.4S,Vm.4S</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vaddhn_high_u64(uint32x2_t r, uint64x2_t a, uint64x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2D b -> Vm.2D</code>	<code>ADDHN2 Vd.4S,Vn.2D,Vm.2D</code>	<code>Vd.4S -> result</code>	A64
<code>int8x8_t vraddhn_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>RADDHN Vd.8B,Vn.8H,Vm.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vraddhn_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>RADDHN Vd.4H,Vn.4S,Vm.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vraddhn_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>RADDHN Vd.2S,Vn.2D,Vm.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vraddhn_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>RADDHN Vd.8B,Vn.8H,Vm.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vraddhn_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>RADDHN Vd.4H,Vn.4S,Vm.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vraddhn_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>RADDHN Vd.2S,Vn.2D,Vm.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8x16_t vraddhn_high_s16(int8x8_t r, int16x8_t a, int16x8_t b)</code>	<code>r -> Vd.8B a -> Vn.8H b -> Vm.8H</code>	<code>RADDHN2 Vd.16B,Vn.8H,Vm.8H</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vraddhn_high_s32(int16x4_t r, int32x4_t a, int32x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4S b -> Vm.4S</code>	<code>RADDHN2 Vd.8H,Vn.4S,Vm.4S</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vraddhn_high_s64(int32x2_t r, int64x2_t a, int64x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2D b -> Vm.2D</code>	<code>RADDHN2 Vd.4S,Vn.2D,Vm.2D</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vraddhn_high_u16(uint8x8_t r, uint16x8_t a, uint16x8_t b)</code>	<code>r -> Vd.8B a -> Vn.8H b -> Vm.8H</code>	<code>RADDHN2 Vd.16B,Vn.8H,Vm.8H</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vraddhn_high_u32(uint16x4_t r, uint32x4_t a, uint32x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4S b -> Vm.4S</code>	<code>RADDHN2 Vd.8H,Vn.4S,Vm.4S</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x4_t vraddhn_high_u64(uint32x2_t r, uint64x2_t a, uint64x2_t b)	r -> Vd.2S a -> Vn.2D b -> Vm.2D	RADDHN2 Vd.4S, Vn.2D, Vm.2D	Vd.4S -> result	A64

2.1.1.1.3 Narrowing addition

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8x8_t vqadd_s8(int8x8_t a, int8x8_t b)	a -> Vn.8B b -> Vm.8B	SQADD Vd.8B, Vn.8B, Vm.8B	Vd.8B -> result	v7/A32/A64
int8x16_t vqaddq_s8(int8x16_t a, int8x16_t b)	a -> Vn.16B b -> Vm.16B	SQADD Vd.16B, Vn.16B, Vm.16B	Vd.16B -> result	v7/A32/A64
int16x4_t vqadd_s16(int16x4_t a, int16x4_t b)	a -> Vn.4H b -> Vm.4H	SQADD Vd.4H, Vn.4H, Vm.4H	Vd.4H -> result	v7/A32/A64
int16x8_t vqaddq_s16(int16x8_t a, int16x8_t b)	a -> Vn.8H b -> Vm.8H	SQADD Vd.8H, Vn.8H, Vm.8H	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vqadd_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SQADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqaddq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SQADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vqadd_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SQADD Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vqaddq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SQADD Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vqadd_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UQADD Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vqaddq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UQADD Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vqadd_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UQADD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vqaddq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UQADD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vqadd_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UQADD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vqaddq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UQADD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vqadd_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>UQADD Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vqaddq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UQADD Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int8_t vqaddb_s8(int8_t a, int8_t b)</code>	<code>a -> Bn b -> Bm</code>	<code>SQADD Bd,Bn,Bm</code>	<code>Bd -> result</code>	A64
<code>int16_t vqaddh_s16(int16_t a, int16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>SQADD Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t</code> <code>vqadds_s32</code> (<code>int32_t</code> <code>a</code> , <code>int32_t</code> <code>b</code>)	<code>a</code> -> <code>Sn</code> <code>b</code> -> <code>Sm</code>	<code>SQADD Sd,Sn,Sm</code>	<code>Sd</code> -> <code>result</code>	A64
<code>int64_t</code> <code>vqaddd_s64</code> (<code>int64_t</code> <code>a</code> , <code>int64_t</code> <code>b</code>)	<code>a</code> -> <code>Dn</code> <code>b</code> -> <code>Dm</code>	<code>SQADD Dd,Dn,Dm</code>	<code>Dd</code> -> <code>result</code>	A64
<code>uint8_t</code> <code>vqaddb_u8</code> (<code>uint8_t</code> <code>a</code> , <code>uint8_t</code> <code>b</code>)	<code>a</code> -> <code>Bn</code> <code>b</code> -> <code>Bm</code>	<code>UQADD Bd,Bn,Bm</code>	<code>Bd</code> -> <code>result</code>	A64
<code>uint16_t</code> <code>vqaddh_u16</code> (<code>uint16_t</code> <code>a</code> , <code>uint16_t</code> <code>b</code>)	<code>a</code> -> <code>Hn</code> <code>b</code> -> <code>Hm</code>	<code>UQADD Hd,Hn,Hm</code>	<code>Hd</code> -> <code>result</code>	A64
<code>uint32_t</code> <code>vqadds_u32</code> (<code>uint32_t</code> <code>a</code> , <code>uint32_t</code> <code>b</code>)	<code>a</code> -> <code>Sn</code> <code>b</code> -> <code>Sm</code>	<code>UQADD Sd,Sn,Sm</code>	<code>Sd</code> -> <code>result</code>	A64
<code>uint64_t</code> <code>vqaddd_u64</code> (<code>uint64_t</code> <code>a</code> , <code>uint64_t</code> <code>b</code>)	<code>a</code> -> <code>Dn</code> <code>b</code> -> <code>Dm</code>	<code>UQADD Dd,Dn,Dm</code>	<code>Dd</code> -> <code>result</code>	A64
<code>int8x8_t</code> <code>vuqadd_s8</code> (<code>int8x8_t</code> <code>a</code> , <code>uint8x8_t</code> <code>b</code>)	<code>a</code> -> <code>Vd.8B</code> <code>b</code> -> <code>Vn.8B</code>	<code>SUQADD Vd.8B,Vn.8B</code>	<code>Vd.8B</code> -> <code>result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vuqaddq_s8(int8x16_t a, uint8x16_t b)</code>	<code>a -> Vd.16B b -> Vn.16B</code>	<code>SUQADD Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int16x4_t vuqadd_s16(int16x4_t a, uint16x4_t b)</code>	<code>a -> Vd.4H b -> Vn.4H</code>	<code>SUQADD Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vuqaddq_s16(int16x8_t a, uint16x8_t b)</code>	<code>a -> Vd.8H b -> Vn.8H</code>	<code>SUQADD Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A64
<code>int32x2_t vuqadd_s32(int32x2_t a, uint32x2_t b)</code>	<code>a -> Vd.2S b -> Vn.2S</code>	<code>SUQADD Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vuqaddq_s32(int32x4_t a, uint32x4_t b)</code>	<code>a -> Vd.4S b -> Vn.4S</code>	<code>SUQADD Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A64
<code>int64x1_t vuqadd_s64(int64x1_t a, uint64x1_t b)</code>	<code>a -> Dd b -> Dn</code>	<code>SUQADD Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vuqaddq_s64(int64x2_t a, uint64x2_t b)</code>	<code>a -> Vd.2D b -> Vn.2D</code>	<code>SUQADD Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8_t vuqaddb_s8(int8_t a, uint8_t b)</code>	a -> Bd b -> Bn	SUQADD Bd,Bn	Bd -> result	A64
<code>int16_t vuqaddh_s16(int16_t a, uint16_t b)</code>	a -> Hd b -> Hn	SUQADD Hd,Hn	Hd -> result	A64
<code>int32_t vuqadds_s32(int32_t a, uint32_t b)</code>	a -> Sd b -> Sn	SUQADD Sd,Sn	Sd -> result	A64
<code>int64_t vuqaddd_s64(int64_t a, uint64_t b)</code>	a -> Dd b -> Dn	SUQADD Dd,Dn	Dd -> result	A64
<code>uint8x8_t vsqadd_u8(uint8x8_t a, int8x8_t b)</code>	a -> Vd.8B b -> Vn.8B	USQADD Vd.8B,Vn.8B	Vd.8B -> result	A64
<code>uint8x16_t vsqaddq_u8(uint8x16_t a, int8x16_t b)</code>	a -> Vd.16B b -> Vn.16B	USQADD Vd.16B,Vn.16B	Vd.16B -> result	A64
<code>uint16x4_t vsqadd_u16(uint16x4_t a, int16x4_t b)</code>	a -> Vd.4H b -> Vn.4H	USQADD Vd.4H,Vn.4H	Vd.4H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vsqaddq_u16(uint16x8_t a, int16x8_t b)</code>	<code>a -> Vd.8H b -> Vn.8H</code>	<code>USQADD Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x2_t vsqadd_u32(uint32x2_t a, int32x2_t b)</code>	<code>a -> Vd.2S b -> Vn.2S</code>	<code>USQADD Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vsqaddq_u32(uint32x4_t a, int32x4_t b)</code>	<code>a -> Vd.4S b -> Vn.4S</code>	<code>USQADD Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x1_t vsqadd_u64(uint64x1_t a, int64x1_t b)</code>	<code>a -> Dd b -> Dn</code>	<code>USQADD Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vsqaddq_u64(uint64x2_t a, int64x2_t b)</code>	<code>a -> Vd.2D b -> Vn.2D</code>	<code>USQADD Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint8_t vsqaddb_u8(uint8_t a, int8_t b)</code>	<code>a -> Bd b -> Bn</code>	<code>USQADD Bd,Bn</code>	<code>Bd -> result</code>	A64
<code>uint16_t vsqaddh_u16(uint16_t a, int16_t b)</code>	<code>a -> Hd b -> Hn</code>	<code>USQADD Hd,Hn</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32_t vsqadds_u32(uint32_t a, int32_t b)</code>	a -> Sd b -> Sn	USQADD Sd,Sn	Sd -> result	A64
<code>uint64_t vsqaddd_u64(uint64_t a, int64_t b)</code>	a -> Dd b -> Dn	USQADD Dd,Dn	Dd -> result	A64

2.1.1.1.4 Saturating addition

2.1.1.2 Multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vmul_s8(int8x8_t a, int8x8_t b)</code>	a -> Vn.8B b -> Vm.8B	MUL Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>int8x16_t vmulq_s8(int8x16_t a, int8x16_t b)</code>	a -> Vn.16B b -> Vm.16B	MUL Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>int16x4_t vmul_s16(int16x4_t a, int16x4_t b)</code>	a -> Vn.4H b -> Vm.4H	MUL Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vmulq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>MUL Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vmul_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>MUL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vmulq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>MUL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vmul_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>MUL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vmulq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>MUL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vmul_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>MUL Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vmulq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>MUL Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vmul_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>MUL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vmulq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>MUL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x2_t vmul_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FMUL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vmulq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FMUL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float64x1_t vmul_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FMUL Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vmulq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FMUL Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x2_t vmull_high_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UMULL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64

2.1.1.2.1 Multiplication

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vmulx_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FMULX Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vmulxq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FMULX Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float64x1_t vmulx_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FMULX Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vmulxq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FMULX Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32_t vmulxs_f32(float32_t a, float32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>FMULX Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>float64_t vmulxd_f64(float64_t a, float64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FMULX Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vmulx_lane_f32(float32x2_t a, float32x2_t v, const int lane)	a -> Vn.2S v -> Vm.2S 0 <= lane <= 1	FMULX Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64
float32x4_t vmulxq_lane_f32(float32x4_t a, float32x2_t v, const int lane)	a -> Vn.4S v -> Vm.2S 0 <= lane <= 1	FMULX Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
float64x1_t vmulx_lane_f64(float64x1_t a, float64x1_t v, const int lane)	a -> Dn v -> Vm.1D 0 <= lane <= 0	FMULX Dd,Dn,Vm.D[lane]	Dd -> result	A64
float64x2_t vmulxq_lane_f64(float64x2_t a, float64x1_t v, const int lane)	a -> Vn.2D v -> Vm.1D 0 <= lane <= 0	FMULX Vd.2D,Vn.2D,Vm.D[lane]	Vd.2D -> result	A64
float32_t vmulxs_lane_f32(float32_t a, float32x2_t v, const int lane)	a -> Sn v -> Vm.2S 0 <= lane <= 1	FMULX Sd,Sn,Vm.S[lane]	Sd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64_t vmulxd_lane_f64(float64_t a, float64x1_t v, const int lane)	a -> Dn v -> Vm.1D 0 <= lane <= 0	FMULX Dd,Dn,Vm.D[lane]	Dd -> result	A64
float32x2_t vmulx_laneq_f32(float32x2_t a, float32x4_t v, const int lane)	a -> Vn.2S v -> Vm.4S 0 <= lane <= 3	FMULX Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64
float32x4_t vmulxq_laneq_f32(float32x4_t a, float32x4_t v, const int lane)	a -> Vn.4S v -> Vm.4S 0 <= lane <= 3	FMULX Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
float64x1_t vmulx_laneq_f64(float64x1_t a, float64x2_t v, const int lane)	a -> Dn v -> Vm.2D 0 <= lane <= 1	FMULX Dd,Dn,Vm.D[lane]	Dd -> result	A64
float64x2_t vmulxq_laneq_f64(float64x2_t a, float64x2_t v, const int lane)	a -> Vn.2D v -> Vm.2D 0 <= lane <= 1	FMULX Vd.2D,Vn.2D,Vm.D[lane]	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32_t vmulxs_laneq_f32(float32_t a, float32x4_t v, const int lane)	a -> Sn v -> Vm.4S 0 <= lane <= 3	FMULX Sd,Sn,Vm.S[lane]	Sd -> result	A64
float64_t vmulxd_laneq_f64(float64_t a, float64x2_t v, const int lane)	a -> Dn v -> Vm.2D 0 <= lane <= 1	FMULX Dd,Dn,Vm.D[lane]	Dd -> result	A64

2.1.1.2.2 Multiply extended

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8x8_t vmla_s8(int8x8_t a, int8x8_t b, int8x8_t c)	a -> Vd.8B b -> Vn.8B c -> Vm.8B	MLA Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
int8x16_t vmlaq_s8(int8x16_t a, int8x16_t b, int8x16_t c)	a -> Vd.16B b -> Vn.16B c -> Vm.16B	MLA Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int16x4_t vmla_s16(int16x4_t a, int16x4_t b, int16x4_t c)	a -> Vd.4H b -> Vn.4H c -> Vm.4H	MLA Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	v7/A32/A64
int16x8_t vmlaq_s16(int16x8_t a, int16x8_t b, int16x8_t c)	a -> Vd.8H b -> Vn.8H c -> Vm.8H	MLA Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	v7/A32/A64
int32x2_t vmla_s32(int32x2_t a, int32x2_t b, int32x2_t c)	a -> Vd.2S b -> Vn.2S c -> Vm.2S	MLA Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	v7/A32/A64
int32x4_t vmlaq_s32(int32x4_t a, int32x4_t b, int32x4_t c)	a -> Vd.4S b -> Vn.4S c -> Vm.4S	MLA Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	v7/A32/A64
uint8x8_t vmla_u8(uint8x8_t a, uint8x8_t b, uint8x8_t c)	a -> Vd.8B b -> Vn.8B c -> Vm.8B	MLA Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint8x16_t vmlaq_u8(uint8x16_t a, uint8x16_t b, uint8x16_t c)	a -> Vd.16B b -> Vn.16B c -> Vm.16B	MLA Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
uint16x4_t vmla_u16(uint16x4_t a, uint16x4_t b, uint16x4_t c)	a -> Vd.4H b -> Vn.4H c -> Vm.4H	MLA Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	v7/A32/A64
uint16x8_t vmlaq_u16(uint16x8_t a, uint16x8_t b, uint16x8_t c)	a -> Vd.8H b -> Vn.8H c -> Vm.8H	MLA Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	v7/A32/A64
uint32x2_t vmla_u32(uint32x2_t a, uint32x2_t b, uint32x2_t c)	a -> Vd.2S b -> Vn.2S c -> Vm.2S	MLA Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	v7/A32/A64
uint32x4_t vmlaq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)	a -> Vd.4S b -> Vn.4S c -> Vm.4S	MLA Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vmla_f32(float32x2_t a, float32x2_t b, float32x2_t c)	N/A	RESULT[I] = a[i] + (b[i] * c[i]) for i = 0 to 1	N/A	v7/A32/A64
float32x4_t vmlaq_f32(float32x4_t a, float32x4_t b, float32x4_t c)	N/A	RESULT[I] = a[i] + (b[i] * c[i]) for i = 0 to 3	N/A	v7/A32/A64
float64x1_t vmla_f64(float64x1_t a, float64x1_t b, float64x1_t c)	N/A	RESULT[I] = a[i] + (b[i] * c[i]) for i = 0	N/A	A64
float64x2_t vmlaq_f64(float64x2_t a, float64x2_t b, float64x2_t c)	N/A	RESULT[I] = a[i] + (b[i] * c[i]) for i = 0 to 1	N/A	A64
int8x8_t vmls_s8(int8x8_t a, int8x8_t b, int8x8_t c)	a -> Vd.8B b -> Vn.8B c -> Vm.8B	MLS Vd.8B, Vn.8B, Vm.8B	Vd.8B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vmlsq_s8(int8x16_t a, int8x16_t b, int8x16_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>MLS Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	<code>v7/A32/A64</code>
<code>int16x4_t vmls_s16(int16x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.4H</code>	<code>MLS Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>int16x8_t vmlsq_s16(int16x8_t a, int16x8_t b, int16x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.8H</code>	<code>MLS Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	<code>v7/A32/A64</code>
<code>int32x2_t vmls_s32(int32x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.2S</code>	<code>MLS Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>int32x4_t vmlsq_s32(int32x4_t a, int32x4_t b, int32x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S</code>	<code>MLS Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	<code>v7/A32/A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vmls_u8(uint8x8_t a, uint8x8_t b, uint8x8_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>MLS Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>uint8x16_t vmlsq_u8(uint8x16_t a, uint8x16_t b, uint8x16_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>MLS Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	<code>v7/A32/A64</code>
<code>uint16x4_t vmls_u16(uint16x4_t a, uint16x4_t b, uint16x4_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.4H</code>	<code>MLS Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>uint16x8_t vmlsq_u16(uint16x8_t a, uint16x8_t b, uint16x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.8H</code>	<code>MLS Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	<code>v7/A32/A64</code>
<code>uint32x2_t vmls_u32(uint32x2_t a, uint32x2_t b, uint32x2_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.2S</code>	<code>MLS Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x4_t vmlsq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)	a -> Vd.4S b -> Vn.4S c -> Vm.4S	MLS Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	v7/A32/A64
float32x2_t vmls_f32(float32x2_t a, float32x2_t b, float32x2_t c)	N/A	RESULT[I] = a[i] - (b[i] * c[i]) for i = 0 to 1	N/A	v7/A32/A64
float32x4_t vmlsq_f32(float32x4_t a, float32x4_t b, float32x4_t c)	N/A	RESULT[I] = a[i] - (b[i] * c[i]) for i = 0 to 3	N/A	v7/A32/A64
float64x1_t vmls_f64(float64x1_t a, float64x1_t b, float64x1_t c)	N/A	RESULT[I] = a[i] - (b[i] * c[i]) for i = 0	N/A	A64
float64x2_t vmlsq_f64(float64x2_t a, float64x2_t b, float64x2_t c)	N/A	RESULT[I] = a[i] - (b[i] * c[i]) for i = 0 to 1	N/A	A64

2.1.1.2.3 Multiply-accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vmlal_s8(int16x8_t a, int8x8_t b, int8x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8B c -> Vm.8B</code>	<code>SMLAL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vmlal_s16(int32x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.4H</code>	<code>SMLAL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vmlal_s32(int64x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.2D b -> Vn.2S c -> Vm.2S</code>	<code>SMLAL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint16x8_t vmlal_u8(uint16x8_t a, uint8x8_t b, uint8x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8B c -> Vm.8B</code>	<code>UMLAL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vmlal_u16(uint32x4_t a, uint16x4_t b, uint16x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.4H</code>	<code>UMLAL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint64x2_t vmlal_u32(uint64x2_t a, uint32x2_t b, uint32x2_t c)	a -> Vd.2D b -> Vn.2S c -> Vm.2S	UMLAL Vd.2D,Vn.2S,Vm.2S	Vd.2D -> result	v7/A32/A64
int16x8_t vmlal_high_s8(int16x8_t a, int8x16_t b, int8x16_t c)	a -> Vd.8H b -> Vn.16B c -> Vm.16B	SMLAL2 Vd.8H,Vn.16B,Vm.16B	Vd.8H -> result	A64
int32x4_t vmlal_high_s16(int32x4_t a, int16x8_t b, int16x8_t c)	a -> Vd.4S b -> Vn.8H c -> Vm.8H	SMLAL2 Vd.4S,Vn.8H,Vm.8H	Vd.4S -> result	A64
int64x2_t vmlal_high_s32(int64x2_t a, int32x4_t b, int32x4_t c)	a -> Vd.2D b -> Vn.4S c -> Vm.4S	SMLAL2 Vd.2D,Vn.4S,Vm.4S	Vd.2D -> result	A64
uint16x8_t vmlal_high_u8(uint16x8_t a, uint8x16_t b, uint8x16_t c)	a -> Vd.8H b -> Vn.16B c -> Vm.16B	UMLAL2 Vd.8H,Vn.16B,Vm.16B	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vmlal_high_u16(uint32x4_t a, uint16x8_t b, uint16x8_t c)</code>	<code>a -> Vd.4S b -> Vn.8H c -> Vm.8H</code>	<code>UMLAL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vmlal_high_u32(uint64x2_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vd.2D b -> Vn.4S c -> Vm.4S</code>	<code>UMLAL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>int16x8_t vmlsl_s8(int16x8_t a, int8x8_t b, int8x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8B c -> Vm.8B</code>	<code>SMLSL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vmlsl_s16(int32x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.4H</code>	<code>SMLSL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vmlsl_s32(int64x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.2D b -> Vn.2S c -> Vm.2S</code>	<code>SMLSL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vmlsl_u8(uint16x8_t a, uint8x8_t b, uint8x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8B c -> Vm.8B</code>	<code>UMLSL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vmlsl_u16(uint32x4_t a, uint16x4_t b, uint16x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.4H</code>	<code>UMLSL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vmlsl_u32(uint64x2_t a, uint32x2_t b, uint32x2_t c)</code>	<code>a -> Vd.2D b -> Vn.2S c -> Vm.2S</code>	<code>UMLSL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int16x8_t vmlsl_high_s8(int16x8_t a, int8x16_t b, int8x16_t c)</code>	<code>a -> Vd.8H b -> Vn.16B c -> Vm.16B</code>	<code>SMLSL2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vmlsl_high_s16(int32x4_t a, int16x8_t b, int16x8_t c)</code>	<code>a -> Vd.4S b -> Vn.8H c -> Vm.8H</code>	<code>SMLSL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vmlsl_high_s32(int64x2_t a, int32x4_t b, int32x4_t c)</code>	<code>a -> Vd.2D b -> Vn.4S c -> Vm.4S</code>	<code>SMLS2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vmlsl_high_u8(uint16x8_t a, uint8x16_t b, uint8x16_t c)</code>	<code>a -> Vd.8H b -> Vn.16B c -> Vm.16B</code>	<code>UMLS2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vmlsl_high_u16(uint32x4_t a, uint16x8_t b, uint16x8_t c)</code>	<code>a -> Vd.4S b -> Vn.8H c -> Vm.8H</code>	<code>UMLS2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vmlsl_high_u32(uint64x2_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vd.2D b -> Vn.4S c -> Vm.4S</code>	<code>UMLS2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64

2.1.1.2.4 Multiply-accumulate and widen

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vfma_f32(float32x2_t a, float32x2_t b, float32x2_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.2S</code>	<code>FMLA Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vfmaq_f32(float32x4_t a, float32x4_t b, float32x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S</code>	<code>FMLA Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float64x1_t vfma_f64(float64x1_t a, float64x1_t b, float64x1_t c)</code>	<code>b -> Dn c -> Dm a -> Da</code>	<code>FMADD Dd,Dn,Dm,Da</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vfmaq_f64(float64x2_t a, float64x2_t b, float64x2_t c)</code>	<code>a -> Vd.2D b -> Vn.2D c -> Vm.2D</code>	<code>FMLA Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32x2_t vfma_lane_f32(float32x2_t a, float32x2_t b, float32x2_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>FMLA Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vfmaq_lane_f32(float32x4_t a, float32x4_t b, float32x2_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.2S 0 <= lane <= 1	FMLA Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
float64x1_t vfma_lane_f64(float64x1_t a, float64x1_t b, float64x1_t v, const int lane)	a -> Dd b -> Dn v -> Vm.1D 0 <= lane <= 0	FMLA Dd,Dn,Vm.D[lane]	Dd -> result	A64
float64x2_t vfmaq_lane_f64(float64x2_t a, float64x2_t b, float64x1_t v, const int lane)	a -> Vd.2D b -> Vn.2D v -> Vm.1D 0 <= lane <= 0	FMLA Vd.2D,Vn.2D,Vm.D[lane]	Vd.2D -> result	A64
float32_t vfmas_lane_f32(float32_t a, float32_t b, float32x2_t v, const int lane)	a -> Sd b -> Sn v -> Vm.2S 0 <= lane <= 1	FMLA Sd,Sn,Vm.S[lane]	Sd -> result	A64
float64_t vfmad_lane_f64(float64_t a, float64_t b, float64x1_t v, const int lane)	a -> Dd b -> Dn v -> Vm.1D 0 <= lane <= 0	FMLA Dd,Dn,Vm.D[lane]	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vfma_laneq_f32(float32x2_t a, float32x2_t b, float32x4_t v, const int lane)	a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	FMLA Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64
float32x4_t vfmaq_laneq_f32(float32x4_t a, float32x4_t b, float32x4_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.4S 0 <= lane <= 3	FMLA Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
float64x1_t vfma_laneq_f64(float64x1_t a, float64x1_t b, float64x2_t v, const int lane)	a -> Dd b -> Dn v -> Vm.2D 0 <= lane <= 1	FMLA Dd,Dn,Vm.D[lane]	Dd -> result	A64
float64x2_t vfmaq_laneq_f64(float64x2_t a, float64x2_t b, float64x2_t v, const int lane)	a -> Vd.2D b -> Vn.2D v -> Vm.2D 0 <= lane <= 1	FMLA Vd.2D,Vn.2D,Vm.D[lane]	Vd.2D -> result	A64
float32_t vfmas_laneq_f32(float32_t a, float32_t b, float32x4_t v, const int lane)	a -> Sd b -> Sn v -> Vm.4S 0 <= lane <= 3	FMLA Sd,Sn,Vm.S[lane]	Sd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64_t vfmad_laneq_f64(float64_t a, float64_t b, float64x2_t v, const int lane)	a -> Dd b -> Dn v -> Vm.2D 0 <= lane <= 1	FMLA Dd,Dn,Vm.D[lane]	Dd -> result	A64
float32x2_t vfms_f32(float32x2_t a, float32x2_t b, float32x2_t c)	a -> Vd.2S b -> Vn.2S c -> Vm.2S	FMLS Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	v7/A32/A64
float32x4_t vfmsq_f32(float32x4_t a, float32x4_t b, float32x4_t c)	a -> Vd.4S b -> Vn.4S c -> Vm.4S	FMLS Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	v7/A32/A64
float64x1_t vfms_f64(float64x1_t a, float64x1_t b, float64x1_t c)	b -> Dn c -> Dm a -> Da	FMSUB Dd,Dn,Dm,Da	Dd -> result	A64
float64x2_t vfmsq_f64(float64x2_t a, float64x2_t b, float64x2_t c)	a -> Vd.2D b -> Vn.2D c -> Vm.2D	FMLS Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vfms_lane_f32(float32x2_t a, float32x2_t b, float32x2_t v, const int lane)	a -> Vd.2S b -> Vn.2S v -> Vm.2S 0 <= lane <= 1	FMLS Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64
float32x4_t vfmsq_lane_f32(float32x4_t a, float32x4_t b, float32x2_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.2S 0 <= lane <= 1	FMLS Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
float64x1_t vfms_lane_f64(float64x1_t a, float64x1_t b, float64x1_t v, const int lane)	a -> Dd b -> Dn v -> Vm.1D 0 <= lane <= 0	FMLS Dd,Dn,Vm.D[lane]	Dd -> result	A64
float64x2_t vfmsq_lane_f64(float64x2_t a, float64x2_t b, float64x1_t v, const int lane)	a -> Vd.2D b -> Vn.2D v -> Vm.1D 0 <= lane <= 0	FMLS Vd.2D,Vn.2D,Vm.D[lane]	Vd.2D -> result	A64
float32_t vfmss_lane_f32(float32_t a, float32_t b, float32x2_t v, const int lane)	a -> Sd b -> Sn v -> Vm.2S 0 <= lane <= 1	FMLS Sd,Sn,Vm.S[lane]	Sd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64_t vfmsd_lane_f64(float64_t a, float64_t b, float64x1_t v, const int lane)	a -> Dd b -> Dn v -> Vm.1D 0 <= lane <= 0	FMLS Dd,Dn,Vm.D[lane]	Dd -> result	A64
float32x2_t vfms_laneq_f32(float32x2_t a, float32x2_t b, float32x4_t v, const int lane)	a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	FMLS Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64
float32x4_t vfmsq_laneq_f32(float32x4_t a, float32x4_t b, float32x4_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.4S 0 <= lane <= 3	FMLS Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
float64x1_t vfms_laneq_f64(float64x1_t a, float64x1_t b, float64x2_t v, const int lane)	a -> Dd b -> Dn v -> Vm.2D 0 <= lane <= 1	FMLS Dd,Dn,Vm.D[lane]	Dd -> result	A64
float64x2_t vfmsq_laneq_f64(float64x2_t a, float64x2_t b, float64x2_t v, const int lane)	a -> Vd.2D b -> Vn.2D v -> Vm.2D 0 <= lane <= 1	FMLS Vd.2D,Vn.2D,Vm.D[lane]	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32_t vfmss_laneq_f32(float32_t a, float32_t b, float32x4_t v, const int lane)	a -> Sd b -> Sn v -> Vm.4S 0 <= lane <= 3	FMLS Sd,Sn,Vm.S[lane]	Sd -> result	A64
float64_t vfmsd_laneq_f64(float64_t a, float64_t b, float64x2_t v, const int lane)	a -> Dd b -> Dn v -> Vm.2D 0 <= lane <= 1	FMLS Dd,Dn,Vm.D[lane]	Dd -> result	A64

2.1.1.2.5 Fused multiply-accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int16x4_t vqdmulh_s16(int16x4_t a, int16x4_t b)	a -> Vn.4H b -> Vm.4H	SQDMULH Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	v7/A32/A64
int16x8_t vqdmulhq_s16(int16x8_t a, int16x8_t b)	a -> Vn.8H b -> Vm.8H	SQDMULH Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vqdmulh_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SQDMULH Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqdmulhq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SQDMULH Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int16_t vqdmulhh_s16(int16_t a, int16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>SQDMULH Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64
<code>int32_t vqdmulhs_s32(int32_t a, int32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>SQDMULH Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>int16x4_t vqrdmulh_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SQRDMULH Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vqrdmulhq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SQRDMULH Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vqrdmulh_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SQRDMULH Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqrdmulhq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SQRDMULH Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int16_t vqrdmulhh_s16(int16_t a, int16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>SQRDMULH Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64
<code>int32_t vqrdmulhs_s32(int32_t a, int32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>SQRDMULH Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>int32x4_t vqdmull_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SQDMULL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vqdmull_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SQDMULL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int32_t vqdmullh_s16(int16_t a, int16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>SQDMULL Sd,Hn,Hm</code>	<code>Sd -> result</code>	A64
<code>int64_t vqdmulls_s32(int32_t a, int32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>SQDMULL Dd,Sn,Sm</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmull_high_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SQDMULL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vqdmull_high_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SQDMULL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64

2.1.1.2.6 Saturating multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmmlal_s16(int32x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.4H</code>	<code>SQDMLAL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vqdmmlal_s32(int64x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.2D b -> Vn.2S c -> Vm.2S</code>	<code>SQDMLAL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int32_t vqdmmlalh_s16(int32_t a, int16_t b, int16_t c)</code>	<code>a -> Sd b -> Hn c -> Hm</code>	<code>SQDMLAL Sd,Hn,Hm</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vqdmmlals_s32(int64_t a, int32_t b, int32_t c)</code>	a -> Dd b -> Sn c -> Sm	SQDMLAL Dd,Sn,Sm	Dd -> result	A64
<code>int32x4_t vqdmmlal_high_s16(int32x4_t a, int16x8_t b, int16x8_t c)</code>	a -> Vd.4S b -> Vn.8H c -> Vm.8H	SQDMLAL2 Vd.4S,Vn.8H,Vm.8H	Vd.4S -> result	A64
<code>int64x2_t vqdmmlal_high_s32(int64x2_t a, int32x4_t b, int32x4_t c)</code>	a -> Vd.2D b -> Vn.4S c -> Vm.4S	SQDMLAL2 Vd.2D,Vn.4S,Vm.4S	Vd.2D -> result	A64
<code>int32x4_t vqdmmlsl_s16(int32x4_t a, int16x4_t b, int16x4_t c)</code>	a -> Vd.4S b -> Vn.4H c -> Vm.4H	SQDMLSL Vd.4S,Vn.4H,Vm.4H	Vd.4S -> result	v7/A32/A64
<code>int64x2_t vqdmmlsl_s32(int64x2_t a, int32x2_t b, int32x2_t c)</code>	a -> Vd.2D b -> Vn.2S c -> Vm.2S	SQDMLSL Vd.2D,Vn.2S,Vm.2S	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t vqdmulsh_s16(int32_t a, int16_t b, int16_t c)</code>	a -> Sd b -> Hn c -> Hm	SQDMLSL Sd,Hn,Hm	Sd -> result	A64
<code>int64_t vqdmulsls_s32(int64_t a, int32_t b, int32_t c)</code>	a -> Dd b -> Sn c -> Sm	SQDMLSL Dd,Sn,Sm	Dd -> result	A64
<code>int32x4_t vqdmulsl_high_s16(int32x4_t a, int16x8_t b, int16x8_t c)</code>	a -> Vd.4S b -> Vn.8H c -> Vm.8H	SQDMLSL2 Vd.4S,Vn.8H,Vm.8H	Vd.4S -> result	A64
<code>int64x2_t vqdmulsl_high_s32(int64x2_t a, int32x4_t b, int32x4_t c)</code>	a -> Vd.2D b -> Vn.4S c -> Vm.4S	SQDMLSL2 Vd.2D,Vn.4S,Vm.4S	Vd.2D -> result	A64
<code>int32x4_t vqdmulal_lane_s16(int32x4_t a, int16x4_t b, int16x4_t v, const int lane)</code>	a -> Vd.4S b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	SQDMLAL Vd.4S,Vn.4H,Vm.H[lane]	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vqdmmlal_lane_s32(int64x2_t a, int32x2_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMLAL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int32_t vqdmmlalh_lane_s16(int32_t a, int16_t b, int16x4_t v, const int lane)</code>	<code>a -> Sd b -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMLAL Sd,Hn,Vm.H[lane]</code>	<code>Sd -> result</code>	A64
<code>int64_t vqdmmlals_lane_s32(int64_t a, int32_t b, int32x2_t v, const int lane)</code>	<code>a -> Dd b -> Sn v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMLAL Dd,Sn,Vm.S[lane]</code>	<code>Dd -> result</code>	A64
<code>int32x4_t vqdmmlal_high_lane_s16(int32x4_t a, int16x8_t b, int16x4_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMLAL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vqdmmlal_high_lane_s32(int64x2_t a, int32x4_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMLAL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmmlal_laneq_s16(int32x4_t a, int16x4_t b, int16x8_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMLAL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vqdmmlal_laneq_s32(int64x2_t a, int32x2_t b, int32x4_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQDMLAL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>int32_t vqdmmlalh_laneq_s16(int32_t a, int16_t b, int16x8_t v, const int lane)</code>	<code>a -> Sd b -> Hn v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMLAL Sd,Hn,Vm.H[lane]</code>	<code>Sd -> result</code>	A64
<code>int64_t vqdmmlals_laneq_s32(int64_t a, int32_t b, int32x4_t v, const int lane)</code>	<code>a -> Dd b -> Sn v -> Vm.4S 0 <= lane <= 3</code>	<code>SQDMLAL Dd,Sn,Vm.S[lane]</code>	<code>Dd -> result</code>	A64
<code>int32x4_t vqdmmlal_high_laneq_s16(int32x4_t a, int16x8_t b, int16x8_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMLAL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vqdmmlal_high_laneq_s32(int64x2_t a, int32x4_t b, int32x4_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQDMLAL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>int32x4_t vqdmmlsl_lane_s16(int32x4_t a, int16x4_t b, int16x4_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMLSL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vqdmmlsl_lane_s32(int64x2_t a, int32x2_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMLSL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int32_t vqdmmlslh_lane_s16(int32_t a, int16_t b, int16x4_t v, const int lane)</code>	<code>a -> Sd b -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMLSL Sd,Hn,Vm.H[lane]</code>	<code>Sd -> result</code>	A64
<code>int64_t vqdmmlsls_lane_s32(int64_t a, int32_t b, int32x2_t v, const int lane)</code>	<code>a -> Dd b -> Sn v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMLSL Dd,Sn,Vm.S[lane]</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmmlsl_high_lane_s16(int32x4_t a, int16x8_t b, int16x4_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMLSL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vqdmmlsl_high_lane_s32(int64x2_t a, int32x4_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMLSL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>int32x4_t vqdmmlsl_laneeq_s16(int32x4_t a, int16x4_t b, int16x8_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMLSL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vqdmmlsl_laneeq_s32(int64x2_t a, int32x2_t b, int32x4_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQDMLSL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>int32_t vqdmmlslh_laneeq_s16(int32_t a, int16_t b, int16x8_t v, const int lane)</code>	<code>a -> Sd b -> Hn v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMLSL Sd,Hn,Vm.H[lane]</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vqdmmlsl_laneq_s32(int64_t a, int32_t b, int32x4_t v, const int lane)</code>	a -> Dd b -> Sn v -> Vm.4S 0 <= lane <= 3	SQDMLSL Dd,Sn,Vm.S[lane]	Dd -> result	A64
<code>int32x4_t vqdmmlsl_high_laneq_s16(int32x4_t a, int16x8_t b, int16x8_t v, const int lane)</code>	a -> Vd.4S b -> Vn.8H v -> Vm.8H 0 <= lane <= 7	SQDMLSL2 Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A64
<code>int64x2_t vqdmmlsl_high_laneq_s32(int64x2_t a, int32x4_t b, int32x4_t v, const int lane)</code>	a -> Vd.2D b -> Vn.4S v -> Vm.4S 0 <= lane <= 3	SQDMLSL2 Vd.2D,Vn.4S,Vm.S[lane]	Vd.2D -> result	A64

2.1.1.2.7 Saturating multiply-accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vmull_s8(int8x8_t a, int8x8_t b)</code>	a -> Vn.8B b -> Vm.8B	SMULL Vd.8H,Vn.8B,Vm.8B	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vmull_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SMULL Vd.4S, Vn.4H, Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vmull_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SMULL Vd.2D, Vn.2S, Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint16x8_t vmull_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UMULL Vd.8H, Vn.8B, Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vmull_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UMULL Vd.4S, Vn.4H, Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vmull_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UMULL Vd.2D, Vn.2S, Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int16x8_t vmull_high_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SMULL2 Vd.8H, Vn.16B, Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vmull_high_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SMULL2 Vd.4S, Vn.8H, Vm.8H</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vmull_high_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SMULL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vmull_high_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UMULL2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vmull_high_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UMULL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64

2.1.1.2.8 Widening multiplication

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmull_n_s16(int16x4_t a, int16_t b)</code>	<code>a -> Vn.4H b -> Vm.H[0]</code>	<code>SQDMULL Vd.4S,Vn.4H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vqdmull_n_s32(int32x2_t a, int32_t b)</code>	<code>a -> Vn.2S b -> Vm.S[0]</code>	<code>SQDMULL Vd.2D,Vn.2S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmull_high_n_s16(int16x8_t a, int16_t b)</code>	<code>a -> Vn.8H b -> Vm.H[0]</code>	<code>SQDMULL2 Vd.4S,Vn.8H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vqdmull_high_n_s32(int32x4_t a, int32_t b)</code>	<code>a -> Vn.4S b -> Vm.S[0]</code>	<code>SQDMULL2 Vd.2D,Vn.4S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	A64
<code>int32x4_t vqdmull_lane_s16(int16x4_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMULL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vqdmull_lane_s32(int32x2_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMULL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int32_t vqdmullh_lane_s16(int16_t a, int16x4_t v, const int lane)</code>	<code>a -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMULL Sd,Hn,Vm.H[lane]</code>	<code>Sd -> result</code>	A64
<code>int64_t vqdmulls_lane_s32(int32_t a, int32x2_t v, const int lane)</code>	<code>a -> Sn v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMULL Dd,Sn,Vm.S[lane]</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmull_high_lane_s16(int16x8_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMULL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vqdmull_high_lane_s32(int32x4_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMULL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>int32x4_t vqdmull_laneq_s16(int16x4_t a, int16x8_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMULL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vqdmull_laneq_s32(int32x2_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQDMULL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>int32_t vqdmullh_laneq_s16(int16_t a, int16x8_t v, const int lane)</code>	<code>a -> Hn v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMULL Sd,Hn,Vm.H[lane]</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vqdmulls_laneq_s32(int32_t a, int32x4_t v, const int lane)</code>	a -> Sn v -> Vm.4S 0 <= lane <= 3	SQDMULL Dd,Sn,Vm.S[lane]	Dd -> result	A64
<code>int32x4_t vqdmull_high_laneq_s16(int16x8_t a, int16x8_t v, const int lane)</code>	a -> Vn.8H v -> Vm.8H 0 <= lane <= 7	SQDMULL2 Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A64
<code>int64x2_t vqdmull_high_laneq_s32(int32x4_t a, int32x4_t v, const int lane)</code>	a -> Vn.4S v -> Vm.4S 0 <= lane <= 3	SQDMULL2 Vd.2D,Vn.4S,Vm.S[lane]	Vd.2D -> result	A64
<code>int16x4_t vqdmulh_n_s16(int16x4_t a, int16_t b)</code>	a -> Vn.4H b -> Vm.H[0]	SQDMULH Vd.4H,Vn.4H,Vm.H[0]	Vd.4H -> result	v7/A32/A64
<code>int16x8_t vqdmulhq_n_s16(int16x8_t a, int16_t b)</code>	a -> Vn.8H b -> Vm.H[0]	SQDMULH Vd.8H,Vn.8H,Vm.H[0]	Vd.8H -> result	v7/A32/A64
<code>int32x2_t vqdmulh_n_s32(int32x2_t a, int32_t b)</code>	a -> Vn.2S b -> Vm.S[0]	SQDMULH Vd.2S,Vn.2S,Vm.S[0]	Vd.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmulhq_n_s32(int32x4_t a, int32_t b)</code>	<code>a -> Vn.4S b -> Vm.S[0]</code>	<code>SQDMULH Vd.4S,Vn.4S,Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int16x4_t vqdmulh_lane_s16(int16x4_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMULH Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vqdmulhq_lane_s16(int16x8_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMULH Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vqdmulh_lane_s32(int32x2_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMULH Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqdmulhq_lane_s32(int32x4_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMULH Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int16_t vqdmulhh_lane_s16(int16_t a, int16x4_t v, const int lane)</code>	<code>a -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>SQDMULH Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t vqdmulhs_lane_s32(int32_t a, int32x2_t v, const int lane)</code>	<code>a -> Sn v -> Vm.2S 0 <= lane <= 1</code>	<code>SQDMULH Sd,Sn,Vm.H[lane]</code>	<code>Sd -> result</code>	A64
<code>int16x4_t vqdmulh_laneq_s16(int16x4_t a, int16x8_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMULH Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vqdmulhq_laneq_s16(int16x8_t a, int16x8_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMULH Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>int32x2_t vqdmulh_laneq_s32(int32x2_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQDMULH Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vqdmulhq_laneq_s32(int32x4_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQDMULH Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16_t vqdmulhh_laneq_s16(int16_t a, int16x8_t v, const int lane)</code>	<code>a -> Hn v -> Vm.8H 0 <= lane <= 7</code>	<code>SQDMULH Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64
<code>int32_t vqdmulhs_laneq_s32(int32_t a, int32x4_t v, const int lane)</code>	<code>a -> Sn v -> Vm.4S 0 <= lane <= 3</code>	<code>SQDMULH Sd,Sn,Vm.H[lane]</code>	<code>Sd -> result</code>	A64
<code>int16x4_t vqrdmulh_n_s16(int16x4_t a, int16_t b)</code>	<code>a -> Vn.4H b -> Vm.H[0]</code>	<code>SQRDMULH Vd.4H,Vn.4H,Vm.H[0]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vqrdmulhq_n_s16(int16x8_t a, int16_t b)</code>	<code>a -> Vn.8H b -> Vm.H[0]</code>	<code>SQRDMULH Vd.8H,Vn.8H,Vm.H[0]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vqrdmulh_n_s32(int32x2_t a, int32_t b)</code>	<code>a -> Vn.2S b -> Vm.S[0]</code>	<code>SQRDMULH Vd.2S,Vn.2S,Vm.S[0]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqrdmulhq_n_s32(int32x4_t a, int32_t b)</code>	<code>a -> Vn.4S b -> Vm.S[0]</code>	<code>SQRDMULH Vd.4S,Vn.4S,Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vqrdmulh_lane_s16(int16x4_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQRDMULH Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vqrdmulhq_lane_s16(int16x8_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQRDMULH Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vqrdmulh_lane_s32(int32x2_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQRDMULH Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqrdmulhq_lane_s32(int32x4_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQRDMULH Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int16_t vqrdmulhh_lane_s16(int16_t a, int16x4_t v, const int lane)</code>	<code>a -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>SQRDMULH Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t vqrdmulhs_lane_s32(int32_t a, int32x2_t v, const int lane)</code>	<code>a -> Sn v -> Vm.2S 0 <= lane <= 1</code>	<code>SQRDMULH Sd,Sn,Vm.S[lane]</code>	<code>Sd -> result</code>	A64
<code>int16x4_t vqrdmulh_laneq_s16(int16x4_t a, int16x8_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQRDMULH Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vqrdmulhq_laneq_s16(int16x8_t a, int16x8_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQRDMULH Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>int32x2_t vqrdmulh_laneq_s32(int32x2_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQRDMULH Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vqrdmulhq_laneq_s32(int32x4_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQRDMULH Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16_t vqrdmulhh_laneq_s16(int16_t a, int16x8_t v, const int lane)</code>	a -> Hn v -> Vm.8H 0 <= lane <= 7	SQRDMULH Hd,Hn,Vm.H[lane]	Hd -> result	A64
<code>int32_t vqrdmulhs_laneq_s32(int32_t a, int32x4_t v, const int lane)</code>	a -> Sn v -> Vm.4S 0 <= lane <= 3	SQRDMULH Sd,Sn,Vm.S[lane]	Sd -> result	A64

2.1.1.2.9 Saturating multiply by scalar and widen

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmulal_n_s16(int32x4_t a, int16x4_t b, int16_t c)</code>	a -> Vd.4S b -> Vn.4H c -> Vm.H[0]	SQDMLAL Vd.4S,Vn.4H,Vm.H[0]	Vd.4S -> result	v7/A32/A64
<code>int64x2_t vqdmulal_n_s32(int64x2_t a, int32x2_t b, int32_t c)</code>	a -> Vd.2D b -> Vn.2S c -> Vm.S[0]	SQDMLAL Vd.2D,Vn.2S,Vm.S[0]	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqdmmlal_high_n_s16(int32x4_t a, int16x8_t b, int16_t c)</code>	a -> Vd.4S b -> Vn.8H c -> Vm.H[0]	SQDMLAL2 Vd.4S,Vn.8H,Vm.H[0]	Vd.4S -> result	A64
<code>int64x2_t vqdmmlal_high_n_s32(int64x2_t a, int32x4_t b, int32_t c)</code>	a -> Vd.2D b -> Vn.4S c -> Vm.S[0]	SQDMLAL2 Vd.2D,Vn.4S,Vm.S[0]	Vd.2D -> result	A64
<code>int32x4_t vqdmmlsl_n_s16(int32x4_t a, int16x4_t b, int16_t c)</code>	a -> Vd.4S b -> Vn.4H c -> Vm.H[0]	SQDMLSL Vd.4S,Vn.4H,Vm.H[0]	Vd.4S -> result	v7/A32/A64
<code>int64x2_t vqdmmlsl_n_s32(int64x2_t a, int32x2_t b, int32_t c)</code>	a -> Vd.2D b -> Vn.2S c -> Vm.S[0]	SQDMLSL Vd.2D,Vn.2S,Vm.S[0]	Vd.2D -> result	v7/A32/A64
<code>int32x4_t vqdmmlsl_high_n_s16(int32x4_t a, int16x8_t b, int16_t c)</code>	a -> Vd.4S b -> Vn.8H c -> Vm.H[0]	SQDMLSL2 Vd.4S,Vn.8H,Vm.H[0]	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vqdmmlsl_high_n_s32(int64x2_t a, int32x4_t b, int32_t c)</code>	<code>a -> Vd.2D b -> Vn.4S c -> Vm.S[0]</code>	<code>SQDMLSL2 Vd.2D,Vn.4S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	A64

2.1.1.2.10 Saturating multiply-accumulate by scalar and widen**2.1.1.3 Polynomial**

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x8_t vmul_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>PMUL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x16_t vmulq_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>PMUL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x8_t vmull_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>PMULL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>poly16x8_t vmull_high_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>PMULL2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64

2.1.1.3.1 Polynomial multiply

2.1.1.4 Division

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vdiv_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FDIV Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vdivq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FDIV Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float64x1_t vdiv_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FDIV Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vdivq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FDIV Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64

2.1.1.5 Subtract

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vsub_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SUB Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vsubq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SUB Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vsub_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SUB Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vsubq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SUB Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vsub_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SUB Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vsubq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SUB Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vsub_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SUB Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vsubq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SUB Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vsub_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SUB Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vsubq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SUB Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vsub_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SUB Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vsubq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SUB Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vsub_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SUB Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vsubq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SUB Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vsub_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SUB Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vsubq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SUB Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>float32x2_t vsub_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FSUB Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vsubq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FSUB Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float64x1_t vsub_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FSUB Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vsubq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FSUB Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int64_t vsubd_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SUB Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vsubd_u64(uint64_t a, uint64_t b)</code>	a -> Dn b -> Dm	SUB Dd,Dn,Dm	Dd -> result	A64

2.1.1.5.1 Subtraction

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vsubl_s8(int8x8_t a, int8x8_t b)</code>	a -> Vn.8B b -> Vm.8B	SSUBL Vd.8H,Vn.8B,Vm.8B	Vd.8H -> result	v7/A32/A64
<code>int32x4_t vsubl_s16(int16x4_t a, int16x4_t b)</code>	a -> Vn.4H b -> Vm.4H	SSUBL Vd.4S,Vn.4H,Vm.4H	Vd.4S -> result	v7/A32/A64
<code>int64x2_t vsubl_s32(int32x2_t a, int32x2_t b)</code>	a -> Vn.2S b -> Vm.2S	SSUBL Vd.2D,Vn.2S,Vm.2S	Vd.2D -> result	v7/A32/A64
<code>uint16x8_t vsubl_u8(uint8x8_t a, uint8x8_t b)</code>	a -> Vn.8B b -> Vm.8B	USUBL Vd.8H,Vn.8B,Vm.8B	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsubl_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>USUBL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vsubl_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>USUBL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int16x8_t vsubl_high_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SSUBL2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vsubl_high_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SSUBL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vsubl_high_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SSUBL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vsubl_high_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>USUBL2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vsubl_high_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>USUBL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vsubl_high_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>USUBL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>int16x8_t vsubw_s8(int16x8_t a, int8x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8B</code>	<code>SSUBW Vd.8H,Vn.8H,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vsubw_s16(int32x4_t a, int16x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4H</code>	<code>SSUBW Vd.4S,Vn.4S,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vsubw_s32(int64x2_t a, int32x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2S</code>	<code>SSUBW Vd.2D,Vn.2D,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint16x8_t vsubw_u8(uint16x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8B</code>	<code>USUBW Vd.8H,Vn.8H,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vsubw_u16(uint32x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4H</code>	<code>USUBW Vd.4S,Vn.4S,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vsubw_u32(uint64x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2S</code>	<code>USUBW Vd.2D,Vn.2D,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vsubw_high_s8(int16x8_t a, int8x16_t b)</code>	<code>a -> Vn.8H b -> Vm.16B</code>	<code>SSUBW2 Vd.8H,Vn.8H,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vsubw_high_s16(int32x4_t a, int16x8_t b)</code>	<code>a -> Vn.4S b -> Vm.8H</code>	<code>SSUBW2 Vd.4S,Vn.4S,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vsubw_high_s32(int64x2_t a, int32x4_t b)</code>	<code>a -> Vn.2D b -> Vm.4S</code>	<code>SSUBW2 Vd.2D,Vn.2D,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vsubw_high_u8(uint16x8_t a, uint8x16_t b)</code>	<code>a -> Vn.8H b -> Vm.16B</code>	<code>USUBW2 Vd.8H,Vn.8H,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vsubw_high_u16(uint32x4_t a, uint16x8_t b)</code>	<code>a -> Vn.4S b -> Vm.8H</code>	<code>USUBW2 Vd.4S,Vn.4S,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vsubw_high_u32(uint64x2_t a, uint32x4_t b)</code>	<code>a -> Vn.2D b -> Vm.4S</code>	<code>USUBW2 Vd.2D,Vn.2D,Vm.4S</code>	<code>Vd.2D -> result</code>	A64

2.1.1.5.2 Widening subtraction

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vhsb_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SHSUB Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vhsbq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SHSUB Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vhsb_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SHSUB Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vhsbq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SHSUB Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vhsb_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SHSUB Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vhsbq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SHSUB Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vhsb_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UHSUB Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vhsbq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UHSUB Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vhsb_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UHSUB Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vhsbq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UHSUB Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vhsb_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UHSUB Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vhsbq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UHSUB Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int8x8_t vsubhn_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SUBHN Vd.8B,Vn.8H,Vm.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vsubhn_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SUBHN Vd.4H,Vn.4S,Vm.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vsubhn_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SUBHN Vd.2S,Vn.2D,Vm.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vsubhn_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SUBHN Vd.8B,Vn.8H,Vm.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vsubhn_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SUBHN Vd.4H,Vn.4S,Vm.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vsubhn_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SUBHN Vd.2S,Vn.2D,Vm.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8x16_t vsubhn_high_s16(int8x8_t r, int16x8_t a, int16x8_t b)</code>	<code>r -> Vd.8B a -> Vn.8H b -> Vm.8H</code>	<code>SUBHN2 Vd.16B,Vn.8H,Vm.8H</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vsubhn_high_s32(int16x4_t r, int32x4_t a, int32x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4S b -> Vm.4S</code>	<code>SUBHN2 Vd.8H,Vn.4S,Vm.4S</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vsubhn_high_s64(int32x2_t r, int64x2_t a, int64x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2D b -> Vm.2D</code>	<code>SUBHN2 Vd.4S,Vn.2D,Vm.2D</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vsubhn_high_u16(uint8x8_t r, uint16x8_t a, uint16x8_t b)</code>	<code>r -> Vd.8B a -> Vn.8H b -> Vm.8H</code>	<code>SUBHN2 Vd.16B,Vn.8H,Vm.8H</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vsubhn_high_u32(uint16x4_t r, uint32x4_t a, uint32x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4S b -> Vm.4S</code>	<code>SUBHN2 Vd.8H,Vn.4S,Vm.4S</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vsubhn_high_u64(uint32x2_t r, uint64x2_t a, uint64x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2D b -> Vm.2D</code>	<code>SUBHN2 Vd.4S,Vn.2D,Vm.2D</code>	<code>Vd.4S -> result</code>	A64
<code>int8x8_t vrsubhn_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>RSUBHN Vd.8B,Vn.8H,Vm.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vrsubhn_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>RSUBHN Vd.4H,Vn.4S,Vm.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vrsubhn_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>RSUBHN Vd.2S,Vn.2D,Vm.2D</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>uint8x8_t vrsubhn_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>RSUBHN Vd.8B,Vn.8H,Vm.8H</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>uint16x4_t vrsubhn_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>RSUBHN Vd.4H,Vn.4S,Vm.4S</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>uint32x2_t vrsubhn_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>RSUBHN Vd.2S,Vn.2D,Vm.2D</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>int8x16_t vrsubhn_high_s16(int8x8_t r, int16x8_t a, int16x8_t b)</code>	<code>r -> Vd.8B a -> Vn.8H b -> Vm.8H</code>	<code>RSUBHN2 Vd.16B,Vn.8H,Vm.8H</code>	<code>Vd.16B -> result</code>	<code>A64</code>
<code>int16x8_t vrsubhn_high_s32(int16x4_t r, int32x4_t a, int32x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4S b -> Vm.4S</code>	<code>RSUBHN2 Vd.8H,Vn.4S,Vm.4S</code>	<code>Vd.8H -> result</code>	<code>A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vrsubhn_high_s64(int32x2_t r, int64x2_t a, int64x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2D b -> Vm.2D</code>	<code>RSUBHN2 Vd.4S,Vn.2D,Vm.2D</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vrsubhn_high_u16(uint8x8_t r, uint16x8_t a, uint16x8_t b)</code>	<code>r -> Vd.8B a -> Vn.8H b -> Vm.8H</code>	<code>RSUBHN2 Vd.16B,Vn.8H,Vm.8H</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vrsubhn_high_u32(uint16x4_t r, uint32x4_t a, uint32x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4S b -> Vm.4S</code>	<code>RSUBHN2 Vd.8H,Vn.4S,Vm.4S</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vrsubhn_high_u64(uint32x2_t r, uint64x2_t a, uint64x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2D b -> Vm.2D</code>	<code>RSUBHN2 Vd.4S,Vn.2D,Vm.2D</code>	<code>Vd.4S -> result</code>	A64

2.1.1.5.3 Narrowing subtraction

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vqsub_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SQSUB Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vqsubq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SQSUB Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vqsub_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SQSUB Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vqsubq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SQSUB Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vqsub_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SQSUB Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqsubq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SQSUB Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vqsub_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SQSUB Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vqsubq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SQSUB Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vqsub_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UQSUB Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vqsubq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UQSUB Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vqsub_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UQSUB Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vqsubq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UQSUB Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vqsub_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UQSUB Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vqsubq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UQSUB Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vqsub_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>UQSUB Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vqsubq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UQSUB Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int8_t vqsubb_s8(int8_t a, int8_t b)</code>	<code>a -> Bn b -> Bm</code>	<code>SQSUB Bd,Bn,Bm</code>	<code>Bd -> result</code>	A64
<code>int16_t vqsubh_s16(int16_t a, int16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>SQSUB Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64
<code>int32_t vqsubs_s32(int32_t a, int32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>SQSUB Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>int64_t vqsubd_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SQSUB Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint8_t vqsubb_u8(uint8_t a, uint8_t b)</code>	<code>a -> Bn b -> Bm</code>	<code>UQSUB Bd,Bn,Bm</code>	<code>Bd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vqsubh_u16(uint16_t a, uint16_t b)</code>	a -> Hn b -> Hm	UQSUB Hd,Hn,Hm	Hd -> result	A64
<code>uint32_t vqsubs_u32(uint32_t a, uint32_t b)</code>	a -> Sn b -> Sm	UQSUB Sd,Sn,Sm	Sd -> result	A64
<code>uint64_t vqsubd_u64(uint64_t a, uint64_t b)</code>	a -> Dn b -> Dm	UQSUB Dd,Dn,Dm	Dd -> result	A64

2.1.1.5.4 Saturating subtract

2.1.1.6 Absolute

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vabd_s8(int8x8_t a, int8x8_t b)</code>	a -> Vn.8B b -> Vm.8B	SABD Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>int8x16_t vabdq_s8(int8x16_t a, int8x16_t b)</code>	a -> Vn.16B b -> Vm.16B	SABD Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vabd_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SABD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vabdq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SABD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vabd_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SABD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vabdq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SABD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vabd_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UABD Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vabdq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UABD Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vabd_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UABD Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vabdq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UABD Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vabd_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UABD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vabdq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UABD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x2_t vabd_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FABD Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vabdq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FABD Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float64x1_t vabd_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FABD Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vabdq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FABD Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32_t vabds_f32(float32_t a, float32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>FABD Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>float64_t vabdd_f64(float64_t a, float64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FABD Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64

2.1.1.6.1 Absolute difference

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vabdl_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SABDL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vabdl_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SABDL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vabdl_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SABDL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vabdl_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UABDL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vabdl_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UABDL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vabdl_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UABDL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int16x8_t vabdl_high_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SABDL2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vabdl_high_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SABDL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vabdl_high_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SABDL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vabdl_high_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UABDL2 Vd.8H,Vn.16B,Vm.16B</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vabdl_high_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UABDL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vabdl_high_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UABDL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64

2.1.1.6.2 Widening absolute difference

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vaba_s8(int8x8_t a, int8x8_t b, int8x8_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>SABA Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vabaq_s8(int8x16_t a, int8x16_t b, int8x16_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>SABA Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vaba_s16(int16x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.4H</code>	<code>SABA Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vabaq_s16(int16x8_t a, int16x8_t b, int16x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.8H</code>	<code>SABA Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vaba_s32(int32x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.2S</code>	<code>SABA Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vabaq_s32(int32x4_t a, int32x4_t b, int32x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S</code>	<code>SABA Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vaba_u8(uint8x8_t a, uint8x8_t b, uint8x8_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>UABA Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vabaq_u8(uint8x16_t a, uint8x16_t b, uint8x16_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>UABA Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vaba_u16(uint16x4_t a, uint16x4_t b, uint16x4_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.4H</code>	<code>UABA Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vabaq_u16(uint16x8_t a, uint16x8_t b, uint16x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.8H</code>	<code>UABA Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vaba_u32(uint32x2_t a, uint32x2_t b, uint32x2_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.2S</code>	<code>UABA Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vabaq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S</code>	<code>UABA Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

2.1.1.6.3 Absolute difference and accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vabal_s8(int16x8_t a, int8x8_t b, int8x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8B c -> Vm.8B</code>	<code>SABAL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vabal_s16(int32x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.4H</code>	<code>SABAL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vabal_s32(int64x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.2D b -> Vn.2S c -> Vm.2S</code>	<code>SABAL Vd.2D,Vn.2S,Vm.2S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint16x8_t vabal_u8(uint16x8_t a, uint8x8_t b, uint8x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8B c -> Vm.8B</code>	<code>UABAL Vd.8H,Vn.8B,Vm.8B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vabal_u16(uint32x4_t a, uint16x4_t b, uint16x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.4H</code>	<code>UABAL Vd.4S,Vn.4H,Vm.4H</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint64x2_t vabal_u32(uint64x2_t a, uint32x2_t b, uint32x2_t c)	a -> Vd.2D b -> Vn.2S c -> Vm.2S	UABAL Vd.2D,Vn.2S,Vm.2S	Vd.2D -> result	v7/A32/A64
int16x8_t vabal_high_s8(int16x8_t a, int8x16_t b, int8x16_t c)	a -> Vd.8H b -> Vn.16B c -> Vm.16B	SABAL2 Vd.8H,Vn.16B,Vm.16B	Vd.8H -> result	A64
int32x4_t vabal_high_s16(int32x4_t a, int16x8_t b, int16x8_t c)	a -> Vd.4S b -> Vn.8H c -> Vm.8H	SABAL2 Vd.4S,Vn.8H,Vm.8H	Vd.4S -> result	A64
int64x2_t vabal_high_s32(int64x2_t a, int32x4_t b, int32x4_t c)	a -> Vd.2D b -> Vn.4S c -> Vm.4S	SABAL2 Vd.2D,Vn.4S,Vm.4S	Vd.2D -> result	A64
uint16x8_t vabal_high_u8(uint16x8_t a, uint8x16_t b, uint8x16_t c)	a -> Vd.8H b -> Vn.16B c -> Vm.16B	UABAL2 Vd.8H,Vn.16B,Vm.16B	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vabal_high_u16(uint32x4_t a, uint16x8_t b, uint16x8_t c)</code>	<code>a -> Vd.4S b -> Vn.8H c -> Vm.8H</code>	<code>UABAL2 Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vabal_high_u32(uint64x2_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vd.2D b -> Vn.4S c -> Vm.4S</code>	<code>UABAL2 Vd.2D,Vn.4S,Vm.4S</code>	<code>Vd.2D -> result</code>	A64

2.1.1.6.4 Widening absolute difference and accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vabs_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>ABS Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vabsq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>ABS Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vabs_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>ABS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vabsq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>ABS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vabs_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>ABS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vabsq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>ABS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x2_t vabs_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FABS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vabsq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FABS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vabs_s64(int64x1_t a)</code>	<code>a -> Dn</code>	<code>ABS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64_t vabsd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>ABS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vabsq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>ABS Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float64x1_t vabs_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FABS Dd,Dn</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x2_t vabsq_f64(float64x2_t a)	a -> Vn.2D	FABS Vd.2D,Vn.2D	Vd.2D -> result	A64

2.1.1.6.5 Absolute value

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8x8_t vqabs_s8(int8x8_t a)	a -> Vn.8B	SQABS Vd.8B,Vn.8B	Vd.8B -> result	v7/A32/A64
int8x16_t vqabsq_s8(int8x16_t a)	a -> Vn.16B	SQABS Vd.16B,Vn.16B	Vd.16B -> result	v7/A32/A64
int16x4_t vqabs_s16(int16x4_t a)	a -> Vn.4H	SQABS Vd.4H,Vn.4H	Vd.4H -> result	v7/A32/A64
int16x8_t vqabsq_s16(int16x8_t a)	a -> Vn.8H	SQABS Vd.8H,Vn.8H	Vd.8H -> result	v7/A32/A64
int32x2_t vqabs_s32(int32x2_t a)	a -> Vn.2S	SQABS Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
int32x4_t vqabsq_s32(int32x4_t a)	a -> Vn.4S	SQABS Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1_t vqabs_s64(int64x1_t a)</code>	<code>a -> Dn</code>	<code>SQABS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vqabsq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>SQABS Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int8_t vqabsb_s8(int8_t a)</code>	<code>a -> Bn</code>	<code>SQABS Bd,Bn</code>	<code>Bd -> result</code>	A64
<code>int16_t vqabsh_s16(int16_t a)</code>	<code>a -> Hn</code>	<code>SQABS Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int32_t vqabss_s32(int32_t a)</code>	<code>a -> Sn</code>	<code>SQABS Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>int64_t vqabsd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>SQABS Dd,Dn</code>	<code>Dd -> result</code>	A64

2.1.1.6.6 Saturating absolute value

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vamax_f16(float16x4_t vn, float16x4_t vm)	vn -> Vn.4H vm -> Vm.4H	FAMAX Vd.4H, Vn.4H, Vm.4H	Vd.4H -> result	A64
float16x8_t vamaxq_f16(float16x8_t vn, float16x8_t vm)	vn -> Vn.8H vm -> Vm.8H	FAMAX Vd.8H, Vn.8H, Vm.8H	Vd.8H -> result	A64
float32x2_t vamax_f32(float32x2_t vn, float32x2_t vm)	vn -> Vn.2S vm -> Vm.2S	FAMAX Vd.2S, Vn.2S, Vm.2S	Vd.2S -> result	A64
float32x4_t vamaxq_f32(float32x4_t vn, float32x4_t vm)	vn -> Vn.4S vm -> Vm.4S	FAMAX Vd.4S, Vn.4S, Vm.4S	Vd.4S -> result	A64
float64x2_t vamaxq_f64(float64x2_t vn, float64x2_t vm)	vn -> Vn.2D vm -> Vm.2D	FAMAX Vd.2D, Vn.2D, Vm.2D	Vd.2D -> result	A64
float16x4_t vamin_f16(float16x4_t vn, float16x4_t vm)	vn -> Vn.4H vm -> Vm.4H	FAMIN Vd.4H, Vn.4H, Vm.4H	Vd.4H -> result	A64
float16x8_t vaminq_f16(float16x8_t vn, float16x8_t vm)	vn -> Vn.8H vm -> Vm.8H	FAMIN Vd.8H, Vn.8H, Vm.8H	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vamin_f32(float32x2_t vn, float32x2_t vm)</code>	<code>vn -> Vn.2S vm -> Vm.2S</code>	<code>FAMIN Vd.2S, Vn.2S, Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vaminq_f32(float32x4_t vn, float32x4_t vm)</code>	<code>vn -> Vn.4S vm -> Vm.4S</code>	<code>FAMIN Vd.4S, Vn.4S, Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float64x2_t vaminq_f64(float64x2_t vn, float64x2_t vm)</code>	<code>vn -> Vn.2D vm -> Vm.2D</code>	<code>FAMIN Vd.2D, Vn.2D, Vm.2D</code>	<code>Vd.2D -> result</code>	A64

2.1.1.6.7 Absolute minimum and maximum

2.1.1.7 Maximum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vmax_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SMAX Vd.8B, Vn.8B, Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vmaxq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SMAX Vd.16B, Vn.16B, Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vmax_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SMAX Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vmaxq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SMAX Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vmax_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SMAX Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vmaxq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SMAX Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vmax_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UMAX Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vmaxq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UMAX Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vmax_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UMAX Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vmaxq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UMAX Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vmax_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UMAX Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vmaxq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UMAX Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x2_t vmax_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FMAX Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vmaxq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FMAX Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float64x1_t vmax_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FMAX Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vmaxq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FMAX Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64

2.1.1.8 Minimum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vmin_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SMIN Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vminq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SMIN Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vmin_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SMIN Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vminq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SMIN Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vmin_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SMIN Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vminq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SMIN Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vmin_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UMIN Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vminq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UMIN Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vmin_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UMIN Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vminq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UMIN Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vmin_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UMIN Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vminq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UMIN Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x2_t vmin_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FMIN Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vminq_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	FMIN Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	v7/A32/A64
float64x1_t vmin_f64(float64x1_t a, float64x1_t b)	a -> Dn b -> Dm	FMIN Dd,Dn,Dm	Dd -> result	A64
float64x2_t vminq_f64(float64x2_t a, float64x2_t b)	a -> Vn.2D b -> Vm.2D	FMIN Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64
float32x2_t vmaxnm_f32(float32x2_t a, float32x2_t b)	a -> Vn.2S b -> Vm.2S	FMAXNM Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	A32/A64
float32x4_t vmaxnmq_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	FMAXNM Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	A32/A64
float64x1_t vmaxnm_f64(float64x1_t a, float64x1_t b)	a -> Dn b -> Dm	FMAXNM Dd,Dn,Dm	Dd -> result	A64
float64x2_t vmaxnmq_f64(float64x2_t a, float64x2_t b)	a -> Vn.2D b -> Vm.2D	FMAXNM Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vminnm_f32(float32x2_t a, float32x2_t b)	a -> Vn.2S b -> Vm.2S	FMINNM Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	A32/A64
float32x4_t vminnmq_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	FMINNM Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	A32/A64
float64x1_t vminnm_f64(float64x1_t a, float64x1_t b)	a -> Dn b -> Dm	FMINNM Dd,Dn,Dm	Dd -> result	A64
float64x2_t vminnmq_f64(float64x2_t a, float64x2_t b)	a -> Vn.2D b -> Vm.2D	FMINNM Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64

2.1.1.9 Rounding

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vrnd_f32(float32x2_t a)	a -> Vn.2S	FRINTZ Vd.2S,Vn.2S	Vd.2S -> result	A32/A64
float32x4_t vrndq_f32(float32x4_t a)	a -> Vn.4S	FRINTZ Vd.4S,Vn.4S	Vd.4S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x1_t vrnd_f64(float64x1_t a)	a -> Dn	FRINTZ Dd,Dn	Dd -> result	A64
float64x2_t vrndq_f64(float64x2_t a)	a -> Vn.2D	FRINTZ Vd.2D,Vn.2D	Vd.2D -> result	A64
float32x2_t vrndn_f32(float32x2_t a)	a -> Vn.2S	FRINTN Vd.2S,Vn.2S	Vd.2S -> result	A32/A64
float32x4_t vrndhq_f32(float32x4_t a)	a -> Vn.4S	FRINTN Vd.4S,Vn.4S	Vd.4S -> result	A32/A64
float64x1_t vrndn_f64(float64x1_t a)	a -> Dn	FRINTN Dd,Dn	Dd -> result	A32/A64
float64x2_t vrndhq_f64(float64x2_t a)	a -> Vn.2D	FRINTN Vd.2D,Vn.2D	Vd.2D -> result	A32/A64
float32_t vrndns_f32(float32_t a)	a -> Sn	FRINTN Sd,Sn	Sd -> result	A32/A64
float32x2_t vrndm_f32(float32x2_t a)	a -> Vn.2S	FRINTM Vd.2S,Vn.2S	Vd.2S -> result	A32/A64
float32x4_t vrndmq_f32(float32x4_t a)	a -> Vn.4S	FRINTM Vd.4S,Vn.4S	Vd.4S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x1_t vrndm_f64(float64x1_t a)	a -> Dn	FRINTM Dd,Dn	Dd -> result	A64
float64x2_t vrndmq_f64(float64x2_t a)	a -> Vn.2D	FRINTM Vd.2D,Vn.2D	Vd.2D -> result	A64
float32x2_t vrndp_f32(float32x2_t a)	a -> Vn.2S	FRINTP Vd.2S,Vn.2S	Vd.2S -> result	A32/A64
float32x4_t vrndpq_f32(float32x4_t a)	a -> Vn.4S	FRINTP Vd.4S,Vn.4S	Vd.4S -> result	A32/A64
float64x1_t vrndp_f64(float64x1_t a)	a -> Dn	FRINTP Dd,Dn	Dd -> result	A64
float64x2_t vrndpq_f64(float64x2_t a)	a -> Vn.2D	FRINTP Vd.2D,Vn.2D	Vd.2D -> result	A64
float32x2_t vrnda_f32(float32x2_t a)	a -> Vn.2S	FRINTA Vd.2S,Vn.2S	Vd.2S -> result	A32/A64
float32x4_t vrndaq_f32(float32x4_t a)	a -> Vn.4S	FRINTA Vd.4S,Vn.4S	Vd.4S -> result	A32/A64
float64x1_t vrnda_f64(float64x1_t a)	a -> Dn	FRINTA Dd,Dn	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2_t vrndaq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FRINTA Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32x2_t vrndi_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FRINTI Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float32x4_t vrndiq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FRINTI Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float64x1_t vrndi_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FRINTI Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vrndiq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FRINTI Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32x2_t vrndx_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FRINTX Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float32x4_t vrndxq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FRINTX Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float64x1_t vrndx_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FRINTX Dd,Dn</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x2_t vrndxq_f64(float64x2_t a)	a -> Vn.2D	FRINTX Vd.2D,Vn.2D	Vd.2D -> result	A64

2.1.1.10 Reciprocal

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x2_t vrecpe_u32(uint32x2_t a)	a -> Vn.2S	URECPE Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
uint32x4_t vrecpeq_u32(uint32x4_t a)	a -> Vn.4S	URECPE Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
float32x2_t vrecpe_f32(float32x2_t a)	a -> Vn.2S	FRECPE Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
float32x4_t vrecpeq_f32(float32x4_t a)	a -> Vn.4S	FRECPE Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
float64x1_t vrecpe_f64(float64x1_t a)	a -> Dn	FRECPE Dd,Dn	Dd -> result	A64
float64x2_t vrecpeq_f64(float64x2_t a)	a -> Vn.2D	FRECPE Vd.2D,Vn.2D	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32_t vrecpes_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FRECPE Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>float64_t vrecped_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FRECPE Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>float32x2_t vrecps_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FRECPS Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vrecpsq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FRECPS Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float64x1_t vrecps_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FRECPS Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vrecpsq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FRECPS Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32_t vrecpss_f32(float32_t a, float32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>FRECPS Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64_t vrecpsd_f64(float64_t a, float64_t b)	a -> Dn b -> Dm	FRECPS Dd,Dn,Dm	Dd -> result	A64

2.1.1.10.1 Reciprocal estimate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x2_t vrsqrte_u32(uint32x2_t a)	a -> Vn.2S	URSQRTE Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
uint32x4_t vrsqrteq_u32(uint32x4_t a)	a -> Vn.4S	URSQRTE Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
float32x2_t vrsqrte_f32(float32x2_t a)	a -> Vn.2S	FRSQRTE Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
float32x4_t vrsqrteq_f32(float32x4_t a)	a -> Vn.4S	FRSQRTE Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
float64x1_t vrsqrte_f64(float64x1_t a)	a -> Dn	FRSQRTE Dd,Dn	Dd -> result	A64
float64x2_t vrsqrteq_f64(float64x2_t a)	a -> Vn.2D	FRSQRTE Vd.2D,Vn.2D	Vd.2D -> result	A64

Copyright: see section [Copyright](#).

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32_t vrsqrtes_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FRSQRTE Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>float64_t vrsqrtd_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FRSQRTE Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>float32x2_t vrsqrts_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FRSQRTS Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vrsqrtsq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FRSQRTS Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float64x1_t vrsqrts_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FRSQRTS Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vrsqrtsq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FRSQRTS Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32_t vrsqrtss_f32(float32_t a, float32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>FRSQRTS Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64_t vrsqrtsd_f64(float64_t a, float64_t b)	a -> Dn b -> Dm	FRSQRTS Dd,Dn,Dm	Dd -> result	A64

2.1.1.10.2 Reciprocal square-root estimate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32_t vrecpxs_f32(float32_t a)	a -> Sn	FRECPX Sd,Sn	Sd -> result	A64
float64_t vrecpxd_f64(float64_t a)	a -> Dn	FRECPX Dd,Dn	Dd -> result	A64

2.1.1.10.3 Reciprocal exponent

2.1.1.11 Square root

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vsqrt_f32(float32x2_t a)	a -> Vn.2S	FSQRT Vd.2S,Vn.2S	Vd.2S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vsqrtq_f32(float32x4_t a)	a -> Vn.4S	FSQRT Vd.4S,Vn.4S	Vd.4S -> result	A64
float64x1_t vsqrt_f64(float64x1_t a)	a -> Dn	FSQRT Dd,Dn	Dd -> result	A64
float64x2_t vsqrtq_f64(float64x2_t a)	a -> Vn.2D	FSQRT Vd.2D,Vn.2D	Vd.2D -> result	A64

2.1.1.12 Pairwise arithmetic

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8x8_t vpadd_s8(int8x8_t a, int8x8_t b)	a -> Vn.8B b -> Vm.8B	ADDP Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
int16x4_t vpadd_s16(int16x4_t a, int16x4_t b)	a -> Vn.4H b -> Vm.4H	ADDP Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	v7/A32/A64
int32x2_t vpadd_s32(int32x2_t a, int32x2_t b)	a -> Vn.2S b -> Vm.2S	ADDP Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vpadu8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ADDP Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vpadu16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>ADDP Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vpadu32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>ADDP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x2_t vpadf32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FADDP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8x16_t vpaddqs8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ADDP Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vpaddqs16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ADDP Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vpaddqs32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ADDP Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vpaddq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ADDP Vd.2D, Vn.2D, Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint8x16_t vpaddq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ADDP Vd.16B, Vn.16B, Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vpaddq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ADDP Vd.8H, Vn.8H, Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vpaddq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ADDP Vd.4S, Vn.4S, Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vpaddq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ADDP Vd.2D, Vn.2D, Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32x4_t vpaddq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FADDP Vd.4S, Vn.4S, Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float64x2_t vpaddq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FADDP Vd.2D, Vn.2D, Vm.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vpaddd_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>ADDP Dd,Vn.2D</code>	<code>Dd -> result</code>	A64
<code>uint64_t vpaddd_u64(uint64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>ADDP Dd,Vn.2D</code>	<code>Dd -> result</code>	A64
<code>float32_t vpadds_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FADDP Sd,Vn.2S</code>	<code>Sd -> result</code>	A64
<code>float64_t vpaddd_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FADDP Dd,Vn.2D</code>	<code>Dd -> result</code>	A64

2.1.1.12.1 Pairwise addition

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vpaddl_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>SADDLP Vd.4H,Vn.8B</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vpaddlq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>SADDLP Vd.8H,Vn.16B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vpaddl_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>SADDLP Vd.2S,Vn.4H</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vpaddlq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SADDLP Vd.4S,Vn.8H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vpaddl_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>SADDLP Vd.1D,Vn.2S</code>	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x2_t vpaddlq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>SADDLP Vd.2D,Vn.4S</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint16x4_t vpaddl_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>UADDLP Vd.4H,Vn.8B</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vpaddlq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>UADDLP Vd.8H,Vn.16B</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vpaddl_u16(uint16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>UADDLP Vd.2S,Vn.4H</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vpaddlq_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>UADDLP Vd.4S,Vn.8H</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vpaddl_u32(uint32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>UADDLP Vd.1D,Vn.2S</code>	<code>Vd.1D -> result</code>	v7/A32/A64
<code>uint64x2_t vpaddlq_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>UADDLP Vd.2D,Vn.4S</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vpadal_s8(int16x4_t a, int8x8_t b)</code>	<code>a -> Vd.4H b -> Vn.8B</code>	SADALP Vd.4H,Vn.8B	Vd.4H -> result	v7/A32/A64
<code>int16x8_t vpadalq_s8(int16x8_t a, int8x16_t b)</code>	<code>a -> Vd.8H b -> Vn.16B</code>	SADALP Vd.8H,Vn.16B	Vd.8H -> result	v7/A32/A64
<code>int32x2_t vpadal_s16(int32x2_t a, int16x4_t b)</code>	<code>a -> Vd.2S b -> Vn.4H</code>	SADALP Vd.2S,Vn.4H	Vd.2S -> result	v7/A32/A64
<code>int32x4_t vpadalq_s16(int32x4_t a, int16x8_t b)</code>	<code>a -> Vd.4S b -> Vn.8H</code>	SADALP Vd.4S,Vn.8H	Vd.4S -> result	v7/A32/A64
<code>int64x1_t vpadal_s32(int64x1_t a, int32x2_t b)</code>	<code>a -> Vd.1D b -> Vn.2S</code>	SADALP Vd.1D,Vn.2S	Vd.1D -> result	v7/A32/A64
<code>int64x2_t vpadalq_s32(int64x2_t a, int32x4_t b)</code>	<code>a -> Vd.2D b -> Vn.4S</code>	SADALP Vd.2D,Vn.4S	Vd.2D -> result	v7/A32/A64
<code>uint16x4_t vpadal_u8(uint16x4_t a, uint8x8_t b)</code>	<code>a -> Vd.4H b -> Vn.8B</code>	UADALP Vd.4H,Vn.8B	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vpadalq_u8(uint16x8_t a, uint8x16_t b)</code>	<code>a -> Vd.8H b -> Vn.16B</code>	UADALP Vd.8H,Vn.16B	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vpadal_u16(uint32x2_t a, uint16x4_t b)</code>	<code>a -> Vd.2S b -> Vn.4H</code>	UADALP Vd.2S,Vn.4H	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vpadalq_u16(uint32x4_t a, uint16x8_t b)</code>	<code>a -> Vd.4S b -> Vn.8H</code>	UADALP Vd.4S,Vn.8H	Vd.4S -> result	v7/A32/A64
<code>uint64x1_t vpadal_u32(uint64x1_t a, uint32x2_t b)</code>	<code>a -> Vd.1D b -> Vn.2S</code>	UADALP Vd.1D,Vn.2S	Vd.1D -> result	v7/A32/A64
<code>uint64x2_t vpadalq_u32(uint64x2_t a, uint32x4_t b)</code>	<code>a -> Vd.2D b -> Vn.4S</code>	UADALP Vd.2D,Vn.4S	Vd.2D -> result	v7/A32/A64

2.1.1.12.2 Pairwise addition and widen

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vmax_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>SMAXP Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vmax_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SMAXP Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vmax_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SMAXP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vmax_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UMAXP Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vmax_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UMAXP Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vmax_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UMAXP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x2_t vmax_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FMAXP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vpmasq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SMAXP Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vpmasq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SMAXP Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vpmasq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SMAXP Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vpmasq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UMAXP Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vpmasq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UMAXP Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vpmasq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UMAXP Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float32x4_t vpmasq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FMAXP Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x2_t vpmaxq_f64(float64x2_t a, float64x2_t b)	a -> Vn.2D b -> Vm.2D	FMAXP Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64
float32_t vpmaxs_f32(float32x2_t a)	a -> Vn.2S	FMAXP Sd,Vn.2S	Sd -> result	A64
float64_t vpmaxqd_f64(float64x2_t a)	a -> Vn.2D	FMAXP Dd,Vn.2D	Dd -> result	A64
float32_t vpmaxnms_f32(float32x2_t a)	a -> Vn.2S	FMAXNMP Sd,Vn.2S	Sd -> result	A64
float64_t vpmaxnmqd_f64(float64x2_t a)	a -> Vn.2D	FMAXNMP Dd,Vn.2D	Dd -> result	A64

2.1.1.12.3 Pairwise maximum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8x8_t vpmín_s8(int8x8_t a, int8x8_t b)	a -> Vn.8B b -> Vm.8B	SMINP Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vpmín_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SMINP Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vpmín_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SMINP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vpmín_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UMINP Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vpmín_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UMINP Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vpmín_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UMINP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x2_t vpmín_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FMINP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8x16_t vpmínq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SMINP Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vpminq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SMINP Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vpminq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SMINP Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vpminq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UMINP Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vpminq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UMINP Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vpminq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UMINP Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float32x4_t vpminq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FMINP Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float64x2_t vpminq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FMINP Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32_t vpmins_f32(float32x2_t a)	a -> Vn.2S	FMINP Sd,Vn.2S	Sd -> result	A64

2.1.1.12.4 Pairwise minimum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vpmaxnm_f32(float32x2_t a, float32x2_t b)	a -> Vn.2S b -> Vm.2S	FMAXNMP Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	A64
float32x4_t vpmaxnmq_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	FMAXNMP Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	A64
float64x2_t vpmaxnmq_f64(float64x2_t a, float64x2_t b)	a -> Vn.2D b -> Vm.2D	FMAXNMP Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64

2.1.1.12.5 Pairwise maximum (IEEE754)

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vpminnm_f32(float32x2_t a, float32x2_t b)	a -> Vn.2S b -> Vm.2S	FMINNMP Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	A64
float32x4_t vpminnmq_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	FMINNMP Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	A64
float64x2_t vpminnmq_f64(float64x2_t a, float64x2_t b)	a -> Vn.2D b -> Vm.2D	FMINNMP Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64
float64_t vpminqd_f64(float64x2_t a)	a -> Vn.2D	FMINP Dd,Vn.2D	Dd -> result	A64
float32_t vpminnms_f32(float32x2_t a)	a -> Vn.2S	FMINNMP Sd,Vn.2S	Sd -> result	A64
float64_t vpminnmqd_f64(float64x2_t a)	a -> Vn.2D	FMINNMP Dd,Vn.2D	Dd -> result	A64

2.1.1.12.6 Pairwise minimum (IEEE754)

2.1.1.13 Across vector arithmetic

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8_t vaddv_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>ADDV Bd,Vn.8B</code>	<code>Bd -> result</code>	A64
<code>int8_t vaddvq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>ADDV Bd,Vn.16B</code>	<code>Bd -> result</code>	A64
<code>int16_t vaddv_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>ADDV Hd,Vn.4H</code>	<code>Hd -> result</code>	A64
<code>int16_t vaddvq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>ADDV Hd,Vn.8H</code>	<code>Hd -> result</code>	A64
<code>int32_t vaddv_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code> <code>a -> Vm.2S</code>	<code>ADDP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.S[0] -> result</code>	A64
<code>int32_t vaddvq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>ADDV Sd,Vn.4S</code>	<code>Sd -> result</code>	A64
<code>int64_t vaddvq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>ADDP Dd,Vn.2D</code>	<code>Dd -> result</code>	A64
<code>uint8_t vaddv_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>ADDV Bd,Vn.8B</code>	<code>Bd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint8_t vaddvq_u8(uint8x16_t a)	a -> Vn.16B	ADDV Bd,Vn.16B	Bd -> result	A64
uint16_t vaddv_u16(uint16x4_t a)	a -> Vn.4H	ADDV Hd,Vn.4H	Hd -> result	A64
uint16_t vaddvq_u16(uint16x8_t a)	a -> Vn.8H	ADDV Hd,Vn.8H	Hd -> result	A64
uint32_t vaddv_u32(uint32x2_t a)	a -> Vn.2S a -> Vm.2S	ADDP Vd.2S,Vn.2S,Vm.2S	Vd.S[0] -> result	A64
uint32_t vaddvq_u32(uint32x4_t a)	a -> Vn.4S	ADDV Sd,Vn.4S	Sd -> result	A64
uint64_t vaddvq_u64(uint64x2_t a)	a -> Vn.2D	ADDP Dd,Vn.2D	Dd -> result	A64
float32_t vaddv_f32(float32x2_t a)	a -> Vn.2S	FADDP Sd,Vn.2S	Sd -> result	A64
float32_t vaddvq_f32(float32x4_t a)	a -> Vn.4S a -> Vm.4S	FADDP Vt.4S,Vn.4S,Vm.4S FADDP Sd,Vt.2S	Sd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64_t vaddvq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FADDP Dd,Vn.2D</code>	<code>Dd -> result</code>	A64

2.1.1.13.1 Addition across vector

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16_t vaddlv_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>SADDLV Hd,Vn.8B</code>	<code>Hd -> result</code>	A64
<code>int16_t vaddlvq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>SADDLV Hd,Vn.16B</code>	<code>Hd -> result</code>	A64
<code>int32_t vaddlv_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>SADDLV Sd,Vn.4H</code>	<code>Sd -> result</code>	A64
<code>int32_t vaddlvq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SADDLV Sd,Vn.8H</code>	<code>Sd -> result</code>	A64
<code>int64_t vaddlv_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>SADDLP Vd.1D,Vn.2S</code>	<code>Dd -> result</code>	A64
<code>int64_t vaddlvq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>SADDLV Dd,Vn.4S</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16_t vaddlv_u8(uint8x8_t a)	a -> Vn.8B	UADDLV Hd,Vn.8B	Hd -> result	A64
uint16_t vaddlvq_u8(uint8x16_t a)	a -> Vn.16B	UADDLV Hd,Vn.16B	Hd -> result	A64
uint32_t vaddlv_u16(uint16x4_t a)	a -> Vn.4H	UADDLV Sd,Vn.4H	Sd -> result	A64
uint32_t vaddlvq_u16(uint16x8_t a)	a -> Vn.8H	UADDLV Sd,Vn.8H	Sd -> result	A64
uint64_t vaddlv_u32(uint32x2_t a)	a -> Vn.2S	UADDLP Vd.1D,Vn.2S	Dd -> result	A64
uint64_t vaddlvq_u32(uint32x4_t a)	a -> Vn.4S	UADDLV Dd,Vn.4S	Dd -> result	A64

2.1.1.13.2 Addition across vector widening

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8_t vmaxv_s8(int8x8_t a)	a -> Vn.8B	SMAVX Bd,Vn.8B	Bd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8_t vmaxvq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>SMAXV Bd,Vn.16B</code>	<code>Bd -> result</code>	A64
<code>int16_t vmaxv_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>SMAXV Hd,Vn.4H</code>	<code>Hd -> result</code>	A64
<code>int16_t vmaxvq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SMAXV Hd,Vn.8H</code>	<code>Hd -> result</code>	A64
<code>int32_t vmaxv_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code> <code>a -> Vm.2S</code>	<code>SMAXP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.S[0] -> result</code>	A64
<code>int32_t vmaxvq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>SMAXV Sd,Vn.4S</code>	<code>Sd -> result</code>	A64
<code>uint8_t vmaxv_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>UMAXV Bd,Vn.8B</code>	<code>Bd -> result</code>	A64
<code>uint8_t vmaxvq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>UMAXV Bd,Vn.16B</code>	<code>Bd -> result</code>	A64
<code>uint16_t vmaxv_u16(uint16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>UMAXV Hd,Vn.4H</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vmaxvq_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>UMAXV Hd,Vn.8H</code>	<code>Hd -> result</code>	A64
<code>uint32_t vmaxv_u32(uint32x2_t a)</code>	<code>a -> Vn.2S</code> <code>a -> Vm.2S</code>	<code>UMAXP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.S[0] -> result</code>	A64
<code>uint32_t vmaxvq_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>UMAXV Sd,Vn.4S</code>	<code>Sd -> result</code>	A64
<code>float32_t vmaxv_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FMAXP Sd,Vn.2S</code>	<code>Sd -> result</code>	A64
<code>float32_t vmaxvq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FMAXV Sd,Vn.4S</code>	<code>Sd -> result</code>	A64
<code>float64_t vmaxvq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FMAXP Dd,Vn.2D</code>	<code>Dd -> result</code>	A64

2.1.1.13.3 Maximum across vector

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8_t vminv_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>SMINV Bd,Vn.8B</code>	<code>Bd -> result</code>	A64
<code>int8_t vminvq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>SMINV Bd,Vn.16B</code>	<code>Bd -> result</code>	A64
<code>int16_t vminv_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>SMINV Hd,Vn.4H</code>	<code>Hd -> result</code>	A64
<code>int16_t vminvq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SMINV Hd,Vn.8H</code>	<code>Hd -> result</code>	A64
<code>int32_t vminv_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code> <code>a -> Vm.2S</code>	<code>SMINP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.S[0] -> result</code>	A64
<code>int32_t vminvq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>SMINV Sd,Vn.4S</code>	<code>Sd -> result</code>	A64
<code>uint8_t vminv_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>UMINV Bd,Vn.8B</code>	<code>Bd -> result</code>	A64
<code>uint8_t vminvq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>UMINV Bd,Vn.16B</code>	<code>Bd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vminv_u16(uint16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>UMINV Hd,Vn.4H</code>	<code>Hd -> result</code>	A64
<code>uint16_t vminvq_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>UMINV Hd,Vn.8H</code>	<code>Hd -> result</code>	A64
<code>uint32_t vminv_u32(uint32x2_t a)</code>	<code>a -> Vn.2S</code> <code>a -> Vm.2S</code>	<code>UMINP Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.S[0] -> result</code>	A64
<code>uint32_t vminvq_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>UMINV Sd,Vn.4S</code>	<code>Sd -> result</code>	A64
<code>float32_t vminv_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FMINP Sd,Vn.2S</code>	<code>Sd -> result</code>	A64
<code>float32_t vminvq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FMINV Sd,Vn.4S</code>	<code>Sd -> result</code>	A64
<code>float64_t vminvq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FMINP Dd,Vn.2D</code>	<code>Dd -> result</code>	A64

2.1.1.13.4 Minimum across vector

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32_t vmaxnmv_f32(float32x2_t a)	a -> Vn.2S	FMAXNMP Sd,Vn.2S	Sd -> result	A64
float32_t vmaxnmvq_f32(float32x4_t a)	a -> Vn.4S	FMAXNMV Sd,Vn.4S	Sd -> result	A64
float64_t vmaxnmvq_f64(float64x2_t a)	a -> Vn.2D	FMAXNMP Dd,Vn.2D	Dd -> result	A64

2.1.1.13.5 Maximum across vector (IEEE754)

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32_t vminnmv_f32(float32x2_t a)	a -> Vn.2S	FMINNMP Sd,Vn.2S	Sd -> result	A64
float32_t vminnmvq_f32(float32x4_t a)	a -> Vn.4S	FMINNMV Sd,Vn.4S	Sd -> result	A64
float64_t vminnmvq_f64(float64x2_t a)	a -> Vn.2D	FMINNMP Dd,Vn.2D	Dd -> result	A64

2.1.1.13.6 Minimum across vector (IEEE754)

2.1.2 Compare

2.1.2.1 Bitwise equal

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vceq_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMEQ Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vceqq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMEQ Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vceq_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMEQ Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vceqq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>CMEQ Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vceq_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>CMEQ Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vceqq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>CMEQ Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vceq_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMEQ Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vceqq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMEQ Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vceq_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMEQ Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vceqq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>CMEQ Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vceq_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>CMEQ Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vceqq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>CMEQ Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint32x2_t vceq_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FCMEQ Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vceqq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	<code>FCMEQ Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vceq_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B</code> <code>b -> Vm.8B</code>	<code>CMEQ Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vceqq_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B</code> <code>b -> Vm.16B</code>	<code>CMEQ Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint64x1_t vceq_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>CMEQ Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vceqq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D</code> <code>b -> Vm.2D</code>	<code>CMEQ Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vceq_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>CMEQ Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vceqq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D</code> <code>b -> Vm.2D</code>	<code>CMEQ Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vceq_p64(poly64x1_t a, poly64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMEQ Dd,Dn,Dm</code>	<code>Dd -> result</code>	A32/A64
<code>uint64x2_t vceqq_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>CMEQ Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A32/A64
<code>uint64x1_t vceq_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FCMEQ Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vceqq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FCMEQ Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64_t vceqd_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMEQ Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64_t vceqd_u64(uint64_t a, uint64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMEQ Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint32_t vceqs_f32(float32_t a, float32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>FCMEQ Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vceqd_f64(float64_t a, float64_t b)</code>	a -> Dn b -> Dm	FCMEQ Dd,Dn,Dm	Dd -> result	A64

2.1.2.2 Bitwise equal to zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vceqz_s8(int8x8_t a)</code>	a -> Vn.8B	CMEQ Vd.8B,Vn.8B,#0	Vd.8B -> result	A64
<code>uint8x16_t vceqzq_s8(int8x16_t a)</code>	a -> Vn.16B	CMEQ Vd.16B,Vn.16B,#0	Vd.16B -> result	A64
<code>uint16x4_t vceqz_s16(int16x4_t a)</code>	a -> Vn.4H	CMEQ Vd.4H,Vn.4H,#0	Vd.4H -> result	A64
<code>uint16x8_t vceqzq_s16(int16x8_t a)</code>	a -> Vn.8H	CMEQ Vd.8H,Vn.8H,#0	Vd.8H -> result	A64
<code>uint32x2_t vceqz_s32(int32x2_t a)</code>	a -> Vn.2S	CMEQ Vd.2S,Vn.2S,#0	Vd.2S -> result	A64
<code>uint32x4_t vceqzq_s32(int32x4_t a)</code>	a -> Vn.4S	CMEQ Vd.4S,Vn.4S,#0	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vceqz_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CMEQ Vd.8B,Vn.8B,#0</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vceqzq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CMEQ Vd.16B,Vn.16B,#0</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x4_t vceqz_u16(uint16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>CMEQ Vd.4H,Vn.4H,#0</code>	<code>Vd.4H -> result</code>	A64
<code>uint16x8_t vceqzq_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>CMEQ Vd.8H,Vn.8H,#0</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x2_t vceqz_u32(uint32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>CMEQ Vd.2S,Vn.2S,#0</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vceqzq_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>CMEQ Vd.4S,Vn.4S,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint32x2_t vceqz_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCMEQ Vd.2S,Vn.2S,#0</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vceqzq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCMEQ Vd.4S,Vn.4S,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x8_t vceqz_p8(poly8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CMEQ Vd.8B,Vn.8B,#0</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vceqzq_p8(poly8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CMEQ Vd.16B,Vn.16B,#0</code>	<code>Vd.16B -> result</code>	A64
<code>uint64x1_t vceqz_s64(int64x1_t a)</code>	<code>a -> Dn</code>	<code>CMEQ Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vceqzq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>CMEQ Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vceqz_u64(uint64x1_t a)</code>	<code>a -> Dn</code>	<code>CMEQ Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vceqzq_u64(uint64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>CMEQ Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vceqz_p64(poly64x1_t a)</code>	<code>a -> Dn</code>	<code>CMEQ Dd,Dn,#0</code>	<code>Dd -> result</code>	A32/A64
<code>uint64x2_t vceqzq_p64(poly64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>CMEQ Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A32/A64
<code>uint64x1_t vceqz_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCMEQ Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vceqzq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCMEQ Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vceqzd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>CMEQ Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64_t vceqzd_u64(uint64_t a)</code>	<code>a -> Dn</code>	<code>CMEQ Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint32_t vceqzs_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCMEQ Sd,Sn,#0</code>	<code>Sd -> result</code>	A64
<code>uint64_t vceqzd_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCMEQ Dd,Dn,#0</code>	<code>Dd -> result</code>	A64

2.1.2.3 Greater than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vcge_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B</code> <code>b -> Vm.8B</code>	<code>CMGE Vd.8B,Vm.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vcgeq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B</code> <code>b -> Vm.16B</code>	<code>CMGE Vd.16B,Vm.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcge_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMGE Vd.4H, Vm.4H, Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vcgeq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>CMGE Vd.8H, Vm.8H, Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vcge_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>CMGE Vd.2S, Vm.2S, Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcgeq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>CMGE Vd.4S, Vm.4S, Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vcge_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMHS Vd.8B, Vm.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vcgeq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMHS Vd.16B, Vm.16B, Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vcge_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMHS Vd.4H, Vm.4H, Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vcgeq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	CMHS <code>Vd.8H, Vm.8H, Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vcge_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	CMHS <code>Vd.2S, Vm.2S, Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcgeq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	CMHS <code>Vd.4S, Vm.4S, Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint32x2_t vcge_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	FCMGE <code>Vd.2S, Vm.2S, Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcgeq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	FCMGE <code>Vd.4S, Vm.4S, Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vcge_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	CMGE <code>Dd, Dn, Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcgeq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	CMGE <code>Vd.2D, Vm.2D, Vn.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vcge_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	CMHS Dd,Dn,Dm	Dd -> result	A64
<code>uint64x2_t vcgeq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	CMHS Vd.2D,Vm.2D,Vn.2D	Vd.2D -> result	A64
<code>uint64x1_t vcge_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	FCMGE Dd,Dn,Dm	Dd -> result	A64
<code>uint64x2_t vcgeq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	FCMGE Vd.2D,Vm.2D,Vn.2D	Vd.2D -> result	A64
<code>uint64_t vcged_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	CMGE Dd,Dn,Dm	Dd -> result	A64
<code>uint64_t vcged_u64(uint64_t a, uint64_t b)</code>	<code>a -> Dn b -> Dm</code>	CMHS Dd,Dn,Dm	Dd -> result	A64
<code>uint32_t vcges_f32(float32_t a, float32_t b)</code>	<code>a -> Sn b -> Sm</code>	FCMGE Sd,Sn,Sm	Sd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vcged_f64(float64_t a, float64_t b)</code>	a -> Dn b -> Dm	FCMGE Dd,Dn,Dm	Dd -> result	A64
<code>uint8x8_t vcgez_s8(int8x8_t a)</code>	a -> Vn.8B	CMGE Vd.8B,Vn.8B,#0	Vd.8B -> result	A64
<code>uint8x16_t vcgezq_s8(int8x16_t a)</code>	a -> Vn.16B	CMGE Vd.16B,Vn.16B,#0	Vd.16B -> result	A64
<code>uint16x4_t vcgez_s16(int16x4_t a)</code>	a -> Vn.4H	CMGE Vd.4H,Vn.4H,#0	Vd.4H -> result	A64
<code>uint16x8_t vcgezq_s16(int16x8_t a)</code>	a -> Vn.8H	CMGE Vd.8H,Vn.8H,#0	Vd.8H -> result	A64
<code>uint32x2_t vcgez_s32(int32x2_t a)</code>	a -> Vn.2S	CMGE Vd.2S,Vn.2S,#0	Vd.2S -> result	A64
<code>uint32x4_t vcgezq_s32(int32x4_t a)</code>	a -> Vn.4S	CMGE Vd.4S,Vn.4S,#0	Vd.4S -> result	A64
<code>uint64x1_t vcgez_s64(int64x1_t a)</code>	a -> Dn	CMGE Dd,Dn,#0	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vcgezq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>CMGE Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint32x2_t vcgez_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCMGE Vd.2S,Vn.2S,#0</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vcgezq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCMGE Vd.4S,Vn.4S,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x1_t vcgez_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCMGE Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcgezq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCMGE Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint64_t vcgezd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>CMGE Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint32_t vcgezs_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCMGE Sd,Sn,#0</code>	<code>Sd -> result</code>	A64
<code>uint64_t vcgezd_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCMGE Dd,Dn,#0</code>	<code>Dd -> result</code>	A64

2.1.2.4 Less than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vcle_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	CMGE Vd.8B, Vm.8B, Vn.8B	Vd.8B -> result	v7/A32/A64
<code>uint8x16_t vcleq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	CMGE Vd.16B, Vm.16B, Vn.16B	Vd.16B -> result	v7/A32/A64
<code>uint16x4_t vcle_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	CMGE Vd.4H, Vm.4H, Vn.4H	Vd.4H -> result	v7/A32/A64
<code>uint16x8_t vcleq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	CMGE Vd.8H, Vm.8H, Vn.8H	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vcle_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	CMGE Vd.2S, Vm.2S, Vn.2S	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vcleq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	CMGE Vd.4S, Vm.4S, Vn.4S	Vd.4S -> result	v7/A32/A64
<code>uint8x8_t vcle_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	CMHS Vd.8B, Vm.8B, Vn.8B	Vd.8B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vcleq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	CMHS Vd.16B, Vm.16B, Vn.16B	Vd.16B -> result	v7/A32/A64
<code>uint16x4_t vcle_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	CMHS Vd.4H, Vm.4H, Vn.4H	Vd.4H -> result	v7/A32/A64
<code>uint16x8_t vcleq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	CMHS Vd.8H, Vm.8H, Vn.8H	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vcle_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	CMHS Vd.2S, Vm.2S, Vn.2S	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vcleq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	CMHS Vd.4S, Vm.4S, Vn.4S	Vd.4S -> result	v7/A32/A64
<code>uint32x2_t vcle_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	FCMGE Vd.2S, Vm.2S, Vn.2S	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vcleq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	FCMGE Vd.4S, Vm.4S, Vn.4S	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vcle_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMGE Dd,Dm,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcleq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>CMGE Vd.2D,Vm.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcle_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMHS Dd,Dm,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcleq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>CMHS Vd.2D,Vm.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcle_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FCMGE Dd,Dm,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcleq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FCMGE Vd.2D,Vm.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64_t vcled_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMGE Dd,Dm,Dn</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vcled_u64(uint64_t a, uint64_t b)</code>	a -> Dn b -> Dm	CMHS Dd,Dm,Dn	Dd -> result	A64
<code>uint32_t vcles_f32(float32_t a, float32_t b)</code>	a -> Sn b -> Sm	FCMGE Sd,Sm,Sn	Sd -> result	A64
<code>uint64_t vcled_f64(float64_t a, float64_t b)</code>	a -> Dn b -> Dm	FCMGE Dd,Dm,Dn	Dd -> result	A64
<code>uint8x8_t vclez_s8(int8x8_t a)</code>	a -> Vn.8B	CMLE Vd.8B,Vn.8B,#0	Vd.8B -> result	A64
<code>uint8x16_t vclezq_s8(int8x16_t a)</code>	a -> Vn.16B	CMLE Vd.16B,Vn.16B,#0	Vd.16B -> result	A64
<code>uint16x4_t vclez_s16(int16x4_t a)</code>	a -> Vn.4H	CMLE Vd.4H,Vn.4H,#0	Vd.4H -> result	A64
<code>uint16x8_t vclezq_s16(int16x8_t a)</code>	a -> Vn.8H	CMLE Vd.8H,Vn.8H,#0	Vd.8H -> result	A64
<code>uint32x2_t vclez_s32(int32x2_t a)</code>	a -> Vn.2S	CMLE Vd.2S,Vn.2S,#0	Vd.2S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vclezq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>CMLE Vd.4S,Vn.4S,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x1_t vclez_s64(int64x1_t a)</code>	<code>a -> Dn</code>	<code>CMLE Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vclezq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>CMLE Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint32x2_t vclez_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>CMLE Vd.2S,Vn.2S,#0</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vclezq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCMLE Vd.4S,Vn.4S,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x1_t vclez_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCMLE Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vclezq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCMLE Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint64_t vclezd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>CMLE Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint32_t vclezs_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCMLE Sd,Sn,#0</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vclezd_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCMLE Dd,Dn,#0</code>	<code>Dd -> result</code>	A64

2.1.2.5 Greater than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vcgt_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMGT Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vcgtq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMGT Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vcgt_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMGT Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vcgtq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>CMGT Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vcgt_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>CMGT Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vcgtq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>CMGT Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vcgt_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMHI Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vcgtq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMHI Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vcgt_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMHI Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vcgtq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>CMHI Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vcgt_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>CMHI Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcgtq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>CMHI Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vcgt_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FCMGT Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcgtq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FCMGT Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vcgt_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMGT Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcgtq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>CMGT Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcgt_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMHI Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcgtq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>CMHI Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcgt_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FCMGT Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vcgtq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FCMGT Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64_t vcgtd_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMGT Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64_t vcgtd_u64(uint64_t a, uint64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMHI Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint32_t vcgts_f32(float32_t a, float32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>FCMGT Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>uint64_t vcgtd_f64(float64_t a, float64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>FCMGT Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint8x8_t vcgtz_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CMGT Vd.8B,Vn.8B,#0</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vcgtzq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CMGT Vd.16B,Vn.16B,#0</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcgtz_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>CMGT Vd.4H,Vn.4H,#0</code>	<code>Vd.4H -> result</code>	A64
<code>uint16x8_t vcgtzq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>CMGT Vd.8H,Vn.8H,#0</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x2_t vcgtz_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>CMGT Vd.2S,Vn.2S,#0</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vcgtzq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>CMGT Vd.4S,Vn.4S,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x1_t vcgtz_s64(int64x1_t a)</code>	<code>a -> Dn</code>	<code>CMGT Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcgtzq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>CMGT Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint32x2_t vcgtz_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCMGT Vd.2S,Vn.2S,#0</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vcgtzq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCMGT Vd.4S,Vn.4S,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x1_t vcgtz_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCMGT Dd,Dn,#0</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vcgtzq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCMGT Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint64_t vcgtzd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>CMGT Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint32_t vcgtzs_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCMGT Sd,Sn,#0</code>	<code>Sd -> result</code>	A64
<code>uint64_t vcgtzd_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCMGT Dd,Dn,#0</code>	<code>Dd -> result</code>	A64

2.1.2.6 Less than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vclt_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B</code> <code>b -> Vm.8B</code>	<code>CMGT Vd.8B,Vm.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vcltq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B</code> <code>b -> Vm.16B</code>	<code>CMGT Vd.16B,Vm.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vclt_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMGT Vd.4H, Vm.4H, Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vcltq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>CMGT Vd.8H, Vm.8H, Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vclt_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>CMGT Vd.2S, Vm.2S, Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcltq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>CMGT Vd.4S, Vm.4S, Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vclt_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMHI Vd.8B, Vm.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vcltq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMHI Vd.16B, Vm.16B, Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vclt_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMHI Vd.4H, Vm.4H, Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vcltq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	CMHI Vd.8H, Vm.8H, Vn.8H	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vclt_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	CMHI Vd.2S, Vm.2S, Vn.2S	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vcltq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	CMHI Vd.4S, Vm.4S, Vn.4S	Vd.4S -> result	v7/A32/A64
<code>uint32x2_t vclt_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	FCMGT Vd.2S, Vm.2S, Vn.2S	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vcltq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	FCMGT Vd.4S, Vm.4S, Vn.4S	Vd.4S -> result	v7/A32/A64
<code>uint64x1_t vclt_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	CMGT Dd, Dm, Dn	Dd -> result	A64
<code>uint64x2_t vcltq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	CMGT Vd.2D, Vm.2D, Vn.2D	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vclt_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	CMHI Dd,Dm,Dn	Dd -> result	A64
<code>uint64x2_t vcltq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	CMHI Vd.2D,Vm.2D,Vn.2D	Vd.2D -> result	A64
<code>uint64x1_t vclt_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	FCMGT Dd,Dm,Dn	Dd -> result	A64
<code>uint64x2_t vcltq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	FCMGT Vd.2D,Vm.2D,Vn.2D	Vd.2D -> result	A64
<code>uint64_t vcltd_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	CMGT Dd,Dm,Dn	Dd -> result	A64
<code>uint64_t vcltd_u64(uint64_t a, uint64_t b)</code>	<code>a -> Dn b -> Dm</code>	CMHI Dd,Dm,Dn	Dd -> result	A64
<code>uint32_t vclts_f32(float32_t a, float32_t b)</code>	<code>a -> Sn b -> Sm</code>	FCMGT Sd,Sm,Sn	Sd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vcltd_f64(float64_t a, float64_t b)</code>	a -> Dn b -> Dm	FCMGT Dd,Dm,Dn	Dd -> result	A64
<code>uint8x8_t vcltz_s8(int8x8_t a)</code>	a -> Vn.8B	CMLT Vd.8B,Vn.8B,#0	Vd.8B -> result	A64
<code>uint8x16_t vcltzq_s8(int8x16_t a)</code>	a -> Vn.16B	CMLT Vd.16B,Vn.16B,#0	Vd.16B -> result	A64
<code>uint16x4_t vcltz_s16(int16x4_t a)</code>	a -> Vn.4H	CMLT Vd.4H,Vn.4H,#0	Vd.4H -> result	A64
<code>uint16x8_t vcltzq_s16(int16x8_t a)</code>	a -> Vn.8H	CMLT Vd.8H,Vn.8H,#0	Vd.8H -> result	A64
<code>uint32x2_t vcltz_s32(int32x2_t a)</code>	a -> Vn.2S	CMLT Vd.2S,Vn.2S,#0	Vd.2S -> result	A64
<code>uint32x4_t vcltzq_s32(int32x4_t a)</code>	a -> Vn.4S	CMLT Vd.4S,Vn.4S,#0	Vd.4S -> result	A64
<code>uint64x1_t vcltz_s64(int64x1_t a)</code>	a -> Dn	CMLT Dd,Dn,#0	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vcltzq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>CMLT Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint32x2_t vcltz_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCMLT Vd.2S,Vn.2S,#0</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vcltzq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCMLT Vd.4S,Vn.4S,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x1_t vcltz_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCMLT Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcltzq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCMLT Vd.2D,Vn.2D,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint64_t vcltzd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>CMLT Dd,Dn,#0</code>	<code>Dd -> result</code>	A64
<code>uint32_t vcltzs_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCMLT Sd,Sn,#0</code>	<code>Sd -> result</code>	A64
<code>uint64_t vcltzd_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCMLT Dd,Dn,#0</code>	<code>Dd -> result</code>	A64

2.1.2.7 Absolute greater than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vcage_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S</code> <code>b -> Vm.2S</code>	<code>FACGE Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcageq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	<code>FACGE Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vcage_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>FACGE Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcageq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D</code> <code>b -> Vm.2D</code>	<code>FACGE Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint32_t vcages_f32(float32_t a, float32_t b)</code>	<code>a -> Sn</code> <code>b -> Sm</code>	<code>FACGE Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>uint64_t vcaged_f64(float64_t a, float64_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>FACGE Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64

2.1.2.8 Absolute less than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vcale_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S</code> <code>b -> Vm.2S</code>	<code>FACGE Vd.2S,Vm.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcaleq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	<code>FACGE Vd.4S,Vm.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vcale_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>FACGE Dd,Dm,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcaleq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D</code> <code>b -> Vm.2D</code>	<code>FACGE Vd.2D,Vm.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint32_t vcales_f32(float32_t a, float32_t b)</code>	<code>a -> Sn</code> <code>b -> Sm</code>	<code>FACGE Sd,Sm,Sn</code>	<code>Sd -> result</code>	A64
<code>uint64_t vcaled_f64(float64_t a, float64_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>FACGE Dd,Dm,Dn</code>	<code>Dd -> result</code>	A64

2.1.2.9 Absolute greater than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vcagt_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S</code> <code>b -> Vm.2S</code>	<code>FACGT Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcagtq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	<code>FACGT Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vcagt_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>FACGT Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcagtq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D</code> <code>b -> Vm.2D</code>	<code>FACGT Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint32_t vcagts_f32(float32_t a, float32_t b)</code>	<code>a -> Sn</code> <code>b -> Sm</code>	<code>FACGT Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>uint64_t vcagtd_f64(float64_t a, float64_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>FACGT Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64

2.1.2.10 Absolute less than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vcalt_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S</code> <code>b -> Vm.2S</code>	<code>FACGT Vd.2S, Vm.2S, Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcaltq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	<code>FACGT Vd.4S, Vm.4S, Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vcalt_f64(float64x1_t a, float64x1_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>FACGT Dd, Dm, Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcaltq_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D</code> <code>b -> Vm.2D</code>	<code>FACGT Vd.2D, Vn.2D, Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint32_t vcalts_f32(float32_t a, float32_t b)</code>	<code>a -> Sn</code> <code>b -> Sm</code>	<code>FACGT Sd, Sm, Sn</code>	<code>Sd -> result</code>	A64
<code>uint64_t vcaltd_f64(float64_t a, float64_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>FACGT Dd, Dm, Dn</code>	<code>Dd -> result</code>	A64

2.1.2.11 Bitwise not equal to zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vtst_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMTST Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vtstq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMTST Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vtst_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMTST Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vtstq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>CMTST Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vtst_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>CMTST Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vtstq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>CMTST Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vtst_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMTST Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vtstq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMTST Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vtst_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>CMTST Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vtstq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>CMTST Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vtst_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>CMTST Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vtstq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>CMTST Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vtst_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>CMTST Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vtstq_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>CMTST Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vtst_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMTST Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vtstq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>CMTST Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vtst_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMTST Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vtstq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>CMTST Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vtst_p64(poly64x1_t a, poly64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMTST Dd,Dn,Dm</code>	<code>Dd -> result</code>	A32/A64
<code>uint64x2_t vtstq_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>CMTST Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A32/A64
<code>uint64_t vtstd_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>CMTST Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vtstd_u64(uint64_t a, uint64_t b)</code>	a -> Dn b -> Dm	CMTST Dd,Dn,Dm	Dd -> result	A64

2.1.3 Shift

2.1.3.1 Left

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vshl_s8(int8x8_t a, int8x8_t b)</code>	a -> Vn.8B b -> Vm.8B	SSHL Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>int8x16_t vshlq_s8(int8x16_t a, int8x16_t b)</code>	a -> Vn.16B b -> Vm.16B	SSHL Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>int16x4_t vshl_s16(int16x4_t a, int16x4_t b)</code>	a -> Vn.4H b -> Vm.4H	SSHL Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	v7/A32/A64
<code>int16x8_t vshlq_s16(int16x8_t a, int16x8_t b)</code>	a -> Vn.8H b -> Vm.8H	SSHL Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vshl_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SSHL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vshlq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SSHL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vshl_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SSHL Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vshlq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SSHL Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vshl_u8(uint8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>USHL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vshlq_u8(uint8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>USHL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vshl_u16(uint16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>USHL Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vshlq_u16(uint16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	USHL Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vshl_u32(uint32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	USHL Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vshlq_u32(uint32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	USHL Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	v7/A32/A64
<code>uint64x1_t vshl_u64(uint64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	USHL Dd,Dn,Dm	Dd -> result	v7/A32/A64
<code>uint64x2_t vshlq_u64(uint64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	USHL Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	v7/A32/A64
<code>int64_t vshld_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	SSHL Dd,Dn,Dm	Dd -> result	A64
<code>uint64_t vshld_u64(uint64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	USHL Dd,Dn,Dm	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vshl_n_s8(int8x8_t a, const int n)</code>	<code>a -> Vn.8B 0 <= n <= 7</code>	<code>SHL Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vshlq_n_s8(int8x16_t a, const int n)</code>	<code>a -> Vn.16B 0 <= n <= 7</code>	<code>SHL Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vshl_n_s16(int16x4_t a, const int n)</code>	<code>a -> Vn.4H 0 <= n <= 15</code>	<code>SHL Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vshlq_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 0 <= n <= 15</code>	<code>SHL Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vshl_n_s32(int32x2_t a, const int n)</code>	<code>a -> Vn.2S 0 <= n <= 31</code>	<code>SHL Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vshlq_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 0 <= n <= 31</code>	<code>SHL Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vshl_n_s64(int64x1_t a, const int n)</code>	<code>a -> Dn 0 <= n <= 63</code>	<code>SHL Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vshlq_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 0 <= n <= 63</code>	<code>SHL Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vshl_n_u8(uint8x8_t a, const int n)</code>	<code>a -> Vn.8B 0 <= n <= 7</code>	<code>SHL Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vshlq_n_u8(uint8x16_t a, const int n)</code>	<code>a -> Vn.16B 0 <= n <= 7</code>	<code>SHL Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vshl_n_u16(uint16x4_t a, const int n)</code>	<code>a -> Vn.4H 0 <= n <= 15</code>	<code>SHL Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vshlq_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 0 <= n <= 15</code>	<code>SHL Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vshl_n_u32(uint32x2_t a, const int n)</code>	<code>a -> Vn.2S 0 <= n <= 31</code>	<code>SHL Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vshlq_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 0 <= n <= 31</code>	<code>SHL Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vshl_n_u64(uint64x1_t a, const int n)</code>	<code>a -> Dn</code> <code>0 <= n <= 63</code>	<code>SHL Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vshlq_n_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D</code> <code>0 <= n <= 63</code>	<code>SHL Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64_t vshld_n_s64(int64_t a, const int n)</code>	<code>a -> Dn</code> <code>0 <= n <= 63</code>	<code>SHL Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>uint64_t vshld_n_u64(uint64_t a, const int n)</code>	<code>a -> Dn</code> <code>0 <= n <= 63</code>	<code>SHL Dd,Dn,#n</code>	<code>Dd -> result</code>	A64

2.1.3.1.1 Vector shift left

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vqshl_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B</code> <code>b -> Vm.8B</code>	<code>SQSHL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vqshlq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>SQSHL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vqshl_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SQSHL Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vqshlq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SQSHL Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vqshl_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SQSHL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqshlq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SQSHL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vqshl_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SQSHL Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vqshlq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SQSHL Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vqshl_u8(uint8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UQSHL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vqshlq_u8(uint8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UQSHL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vqshl_u16(uint16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UQSHL Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vqshlq_u16(uint16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UQSHL Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vqshl_u32(uint32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UQSHL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vqshlq_u32(uint32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UQSHL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vqshl_u64(uint64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>UQSHL Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vqshlq_u64(uint64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UQSHL Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int8_t vqshlb_s8(int8_t a, int8_t b)</code>	<code>a -> Bn b -> Bm</code>	<code>SQSHL Bd,Bn,Bm</code>	<code>Bd -> result</code>	A64
<code>int16_t vqshlh_s16(int16_t a, int16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>SQSHL Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64
<code>int32_t vqshls_s32(int32_t a, int32_t b)</code>	<code>a -> Sn b -> Sm</code>	<code>SQSHL Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>int64_t vqshld_s64(int64_t a, int64_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SQSHL Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>uint8_t vqshlb_u8(uint8_t a, int8_t b)</code>	<code>a -> Bn b -> Bm</code>	<code>UQSHL Bd,Bn,Bm</code>	<code>Bd -> result</code>	A64
<code>uint16_t vqshlh_u16(uint16_t a, int16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>UQSHL Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32_t</code> <code>vqshls_u32(</code> <code>uint32_t a,</code> <code>int32_t b)</code>	<code>a -> Sn</code> <code>b -> Sm</code>	<code>UQSHL Sd,Sn,Sm</code>	<code>Sd -> result</code>	A64
<code>uint64_t</code> <code>vqshld_u64(</code> <code>uint64_t a,</code> <code>int64_t b)</code>	<code>a -> Dn</code> <code>b -> Dm</code>	<code>UQSHL Dd,Dn,Dm</code>	<code>Dd -> result</code>	A64
<code>int8x8_t</code> <code>vqshl_n_s8(</code> <code>int8x8_t a,</code> <code>const int n)</code>	<code>a -> Vn.8B</code> <code>0 <= n <= 7</code>	<code>SQSHL Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t</code> <code>vqshlq_n_s8(</code> <code>int8x16_t a,</code> <code>const int n)</code>	<code>a -> Vn.16B</code> <code>0 <= n <= 7</code>	<code>SQSHL Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t</code> <code>vqshl_n_s16(</code> <code>int16x4_t a,</code> <code>const int n)</code>	<code>a -> Vn.4H</code> <code>0 <= n <= 15</code>	<code>SQSHL Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t</code> <code>vqshlq_n_s16(</code> <code>int16x8_t a,</code> <code>const int n)</code>	<code>a -> Vn.8H</code> <code>0 <= n <= 15</code>	<code>SQSHL Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t</code> <code>vqshl_n_s32(</code> <code>int32x2_t a,</code> <code>const int n)</code>	<code>a -> Vn.2S</code> <code>0 <= n <= 31</code>	<code>SQSHL Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqshlq_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 0 <= n <= 31</code>	<code>SQSHL Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vqshl_n_s64(int64x1_t a, const int n)</code>	<code>a -> Dn 0 <= n <= 63</code>	<code>SQSHL Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vqshlq_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 0 <= n <= 63</code>	<code>SQSHL Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vqshl_n_u8(uint8x8_t a, const int n)</code>	<code>a -> Vn.8B 0 <= n <= 7</code>	<code>UQSHL Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vqshlq_n_u8(uint8x16_t a, const int n)</code>	<code>a -> Vn.16B 0 <= n <= 7</code>	<code>UQSHL Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vqshl_n_u16(uint16x4_t a, const int n)</code>	<code>a -> Vn.4H 0 <= n <= 15</code>	<code>UQSHL Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vqshlq_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 0 <= n <= 15</code>	<code>UQSHL Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vqshl_n_u32(uint32x2_t a, const int n)</code>	<code>a -> Vn.2S 0 <= n <= 31</code>	<code>UQSHL Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vqshlq_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 0 <= n <= 31</code>	<code>UQSHL Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vqshl_n_u64(uint64x1_t a, const int n)</code>	<code>a -> Dn 0 <= n <= 63</code>	<code>UQSHL Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vqshlq_n_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D 0 <= n <= 63</code>	<code>UQSHL Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int8_t vqshlb_n_s8(int8_t a, const int n)</code>	<code>a -> Bn 0 <= n <= 7</code>	<code>SQSHL Bd,Bn,#n</code>	<code>Bd -> result</code>	A64
<code>int16_t vqshlh_n_s16(int16_t a, const int n)</code>	<code>a -> Hn 0 <= n <= 15</code>	<code>SQSHL Hd,Hn,#n</code>	<code>Hd -> result</code>	A64
<code>int32_t vqshls_n_s32(int32_t a, const int n)</code>	<code>a -> Sn 0 <= n <= 31</code>	<code>SQSHL Sd,Sn,#n</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vqshld_n_s64(int64_t a, const int n)</code>	<code>a -> Dn</code> <code>0 <= n <= 63</code>	<code>SQSHL Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>uint8_t vqshlb_n_u8(uint8_t a, const int n)</code>	<code>a -> Bn</code> <code>0 <= n <= 7</code>	<code>UQSHL Bd,Bn,#n</code>	<code>Bd -> result</code>	A64
<code>uint16_t vqshlh_n_u16(uint16_t a, const int n)</code>	<code>a -> Hn</code> <code>0 <= n <= 15</code>	<code>UQSHL Hd,Hn,#n</code>	<code>Hd -> result</code>	A64
<code>uint32_t vqshls_n_u32(uint32_t a, const int n)</code>	<code>a -> Sn</code> <code>0 <= n <= 31</code>	<code>UQSHL Sd,Sn,#n</code>	<code>Sd -> result</code>	A64
<code>uint64_t vqshld_n_u64(uint64_t a, const int n)</code>	<code>a -> Dn</code> <code>0 <= n <= 63</code>	<code>UQSHL Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>uint8x8_t vqshlu_n_s8(int8x8_t a, const int n)</code>	<code>a -> Vn.8B</code> <code>0 <= n <= 7</code>	<code>SQSHLU Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vqshluq_n_s8(int8x16_t a, const int n)</code>	<code>a -> Vn.16B</code> <code>0 <= n <= 7</code>	<code>SQSHLU Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vqshlu_n_s16(int16x4_t a, const int n)</code>	<code>a -> Vn.4H 0 <= n <= 15</code>	<code>SQSHLU Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vqshluq_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 0 <= n <= 15</code>	<code>SQSHLU Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vqshlu_n_s32(int32x2_t a, const int n)</code>	<code>a -> Vn.2S 0 <= n <= 31</code>	<code>SQSHLU Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vqshluq_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 0 <= n <= 31</code>	<code>SQSHLU Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vqshlu_n_s64(int64x1_t a, const int n)</code>	<code>a -> Dn 0 <= n <= 63</code>	<code>SQSHLU Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vqshluq_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 0 <= n <= 63</code>	<code>SQSHLU Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8_t vqshlub_n_s8(int8_t a, const int n)</code>	<code>a -> Bn 0 <= n <= 7</code>	<code>SQSHLU Bd,Bn,#n</code>	<code>Bd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t</code> <code>vqshluh_n_s16</code> (<code>int16_t</code> <code>a</code> , <code>const int</code> <code>n</code>)	<code>a</code> -> <code>Hn</code> <code>0</code> <= <code>n</code> <= 15	SQSHLU <code>Hd</code> , <code>Hn</code> , <code>#n</code>	<code>Hd</code> -> result	A64
<code>uint32_t</code> <code>vqshlus_n_s32</code> (<code>int32_t</code> <code>a</code> , <code>const int</code> <code>n</code>)	<code>a</code> -> <code>Sn</code> <code>0</code> <= <code>n</code> <= 31	SQSHLU <code>Sd</code> , <code>Sn</code> , <code>#n</code>	<code>Sd</code> -> result	A64
<code>uint64_t</code> <code>vqshlud_n_s64</code> (<code>int64_t</code> <code>a</code> , <code>const int</code> <code>n</code>)	<code>a</code> -> <code>Dn</code> <code>0</code> <= <code>n</code> <= 63	SQSHLU <code>Dd</code> , <code>Dn</code> , <code>#n</code>	<code>Dd</code> -> result	A64

2.1.3.1.2 Vector saturating shift left

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t</code> <code>vrshl_s8</code> (<code>int8x8_t</code> <code>a</code> , <code>int8x8_t</code> <code>b</code>)	<code>a</code> -> <code>Vn.8B</code> <code>b</code> -> <code>Vm.8B</code>	SRSHL <code>Vd.8B</code> , <code>Vn.8B</code> , <code>Vm.8B</code>	<code>Vd.8B</code> -> result	v7/A32/A64
<code>int8x16_t</code> <code>vrshlq_s8</code> (<code>int8x16_t</code> <code>a</code> , <code>int8x16_t</code> <code>b</code>)	<code>a</code> -> <code>Vn.16B</code> <code>b</code> -> <code>Vm.16B</code>	SRSHL <code>Vd.16B</code> , <code>Vn.16B</code> , <code>Vm.16B</code>	<code>Vd.16B</code> -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vrshl_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>SRSHL Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vrshlq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SRSHL Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vrshl_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SRSHL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vrshlq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SRSHL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vrshl_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SRSHL Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vrshlq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SRSHL Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vrshl_u8(uint8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>URSHL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vrshlq_u8(uint8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	URSHL Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>uint16x4_t vrshl_u16(uint16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	URSHL Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	v7/A32/A64
<code>uint16x8_t vrshlq_u16(uint16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	URSHL Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vrshl_u32(uint32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	URSHL Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vrshlq_u32(uint32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	URSHL Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	v7/A32/A64
<code>uint64x1_t vrshl_u64(uint64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	URSHL Dd,Dn,Dm	Dd -> result	v7/A32/A64
<code>uint64x2_t vrshlq_u64(uint64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	URSHL Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vrshld_s64(int64_t a, int64_t b)</code>	a -> Dn b -> Dm	SRSHL Dd,Dn,Dm	Dd -> result	A64
<code>uint64_t vrshld_u64(uint64_t a, int64_t b)</code>	a -> Dn b -> Dm	URSHL Dd,Dn,Dm	Dd -> result	A64

2.1.3.1.3 Vector rounding shift left

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vqrshl_s8(int8x8_t a, int8x8_t b)</code>	a -> Vn.8B b -> Vm.8B	SQRSHL Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>int8x16_t vqrshlq_s8(int8x16_t a, int8x16_t b)</code>	a -> Vn.16B b -> Vm.16B	SQRSHL Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>int16x4_t vqrshl_s16(int16x4_t a, int16x4_t b)</code>	a -> Vn.4H b -> Vm.4H	SQRSHL Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vqrshlq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>SQRSHL Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vqrshl_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>SQRSHL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqrshlq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SQRSHL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vqrshl_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>SQRSHL Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vqrshlq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>SQRSHL Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vqrshl_u8(uint8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UQRSHL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vqrshlq_u8(uint8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UQRSHL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vqrshl_u16(uint16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UQRSHL Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vqrshlq_u16(uint16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UQRSHL Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vqrshl_u32(uint32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UQRSHL Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vqrshlq_u32(uint32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UQRSHL Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vqrshl_u64(uint64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>UQRSHL Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vqrshlq_u64(uint64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UQRSHL Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int8_t vqrshlb_s8(int8_t a, int8_t b)</code>	<code>a -> Bn b -> Bm</code>	<code>SQRSHL Bd,Bn,Bm</code>	<code>Bd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16_t vqrshlh_s16(int16_t a, int16_t b)</code>	a -> Hn b -> Hm	SQRSHL Hd,Hn,Hm	Hd -> result	A64
<code>int32_t vqrshls_s32(int32_t a, int32_t b)</code>	a -> Sn b -> Sm	SQRSHL Sd,Sn,Sm	Sd -> result	A64
<code>int64_t vqrshld_s64(int64_t a, int64_t b)</code>	a -> Dn b -> Dm	SQRSHL Dd,Dn,Dm	Dd -> result	A64
<code>uint8_t vqrshlb_u8(uint8_t a, uint8_t b)</code>	a -> Bn b -> Bm	UQRSHL Bd,Bn,Bm	Bd -> result	A64
<code>uint16_t vqrshlh_u16(uint16_t a, uint16_t b)</code>	a -> Hn b -> Hm	UQRSHL Hd,Hn,Hm	Hd -> result	A64
<code>uint32_t vqrshls_u32(uint32_t a, uint32_t b)</code>	a -> Sn b -> Sm	UQRSHL Sd,Sn,Sm	Sd -> result	A64
<code>uint64_t vqrshld_u64(uint64_t a, uint64_t b)</code>	a -> Dn b -> Dm	UQRSHL Dd,Dn,Dm	Dd -> result	A64

2.1.3.1.4 Vector saturating rounding shift left

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vshll_n_s8(int8x8_t a, const int n)</code>	<code>a -> Vn.8B 0 <= n <= 7</code>	<code>SSHLL Vd.8H,Vn.8B,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vshll_n_s16(int16x4_t a, const int n)</code>	<code>a -> Vn.4H 0 <= n <= 15</code>	<code>SSHLL Vd.4S,Vn.4H,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vshll_n_s32(int32x2_t a, const int n)</code>	<code>a -> Vn.2S 0 <= n <= 31</code>	<code>SSHLL Vd.2D,Vn.2S,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint16x8_t vshll_n_u8(uint8x8_t a, const int n)</code>	<code>a -> Vn.8B 0 <= n <= 7</code>	<code>USHLL Vd.8H,Vn.8B,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vshll_n_u16(uint16x4_t a, const int n)</code>	<code>a -> Vn.4H 0 <= n <= 15</code>	<code>USHLL Vd.4S,Vn.4H,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vshll_n_u32(uint32x2_t a, const int n)</code>	<code>a -> Vn.2S 0 <= n <= 31</code>	<code>USHLL Vd.2D,Vn.2S,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vshll_high_n_s8(int8x16_t a, const int n)</code>	<code>a -> Vn.16B 0 <= n <= 7</code>	<code>SSHLL2 Vd.8H,Vn.16B,#n</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vshll_high_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 0 <= n <= 15</code>	<code>SSHLL2 Vd.4S,Vn.8H,#n</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vshll_high_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 0 <= n <= 31</code>	<code>SSHLL2 Vd.2D,Vn.4S,#n</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vshll_high_n_u8(uint8x16_t a, const int n)</code>	<code>a -> Vn.16B 0 <= n <= 7</code>	<code>USHLL2 Vd.8H,Vn.16B,#n</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vshll_high_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 0 <= n <= 15</code>	<code>USHLL2 Vd.4S,Vn.8H,#n</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vshll_high_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 0 <= n <= 31</code>	<code>USHLL2 Vd.2D,Vn.4S,#n</code>	<code>Vd.2D -> result</code>	A64
<code>int16x8_t vshll_n_s8(int8x8_t a, const int n)</code>	<code>a -> Vn.8B n == 8</code>	<code>SHLL Vd.8H,Vn.8B,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vshll_n_s16(int16x4_t a, const int n)</code>	<code>a -> Vn.4H n == 16</code>	SHLL Vd.4S,Vn.4H,#n	Vd.4S -> result	v7/A32/A64
<code>int64x2_t vshll_n_s32(int32x2_t a, const int n)</code>	<code>a -> Vn.2S n == 32</code>	SHLL Vd.2D,Vn.2S,#n	Vd.2D -> result	v7/A32/A64
<code>uint16x8_t vshll_n_u8(uint8x8_t a, const int n)</code>	<code>a -> Vn.8B n == 8</code>	SHLL Vd.8H,Vn.8B,#n	Vd.8H -> result	v7/A32/A64
<code>uint32x4_t vshll_n_u16(uint16x4_t a, const int n)</code>	<code>a -> Vn.4H n == 16</code>	SHLL Vd.4S,Vn.4H,#n	Vd.4S -> result	v7/A32/A64
<code>uint64x2_t vshll_n_u32(uint32x2_t a, const int n)</code>	<code>a -> Vn.2S n == 32</code>	SHLL Vd.2D,Vn.2S,#n	Vd.2D -> result	v7/A32/A64
<code>int16x8_t vshll_high_n_s8(int8x16_t a, const int n)</code>	<code>a -> Vn.16B n == 8</code>	SHLL2 Vd.8H,Vn.16B,#n	Vd.8H -> result	A64
<code>int32x4_t vshll_high_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H n == 16</code>	SHLL2 Vd.4S,Vn.8H,#n	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vshll_high_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S n == 32</code>	<code>SHLL2 Vd.2D,Vn.4S,#n</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vshll_high_n_u8(uint8x16_t a, const int n)</code>	<code>a -> Vn.16B n == 8</code>	<code>SHLL2 Vd.8H,Vn.16B,#n</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vshll_high_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H n == 16</code>	<code>SHLL2 Vd.4S,Vn.8H,#n</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vshll_high_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S n == 32</code>	<code>SHLL2 Vd.2D,Vn.4S,#n</code>	<code>Vd.2D -> result</code>	A64

2.1.3.1.5 Vector shift left and widen

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vsli_n_s8(int8x8_t a, int8x8_t b, const int n)</code>	<code>a -> Vd.8B b -> Vn.8B 0 <= n <= 7</code>	<code>SLI Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vsliq_n_s8(int8x16_t a, int8x16_t b, const int n)</code>	<code>a -> Vd.16B b -> Vn.16B 0 <= n <= 7</code>	<code>SLI Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vsli_n_s16(int16x4_t a, int16x4_t b, const int n)</code>	<code>a -> Vd.4H b -> Vn.4H 0 <= n <= 15</code>	<code>SLI Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vsliq_n_s16(int16x8_t a, int16x8_t b, const int n)</code>	<code>a -> Vd.8H b -> Vn.8H 0 <= n <= 15</code>	<code>SLI Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vsli_n_s32(int32x2_t a, int32x2_t b, const int n)</code>	<code>a -> Vd.2S b -> Vn.2S 0 <= n <= 31</code>	<code>SLI Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vsliq_n_s32(int32x4_t a, int32x4_t b, const int n)</code>	<code>a -> Vd.4S b -> Vn.4S 0 <= n <= 31</code>	<code>SLI Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int64x1_t vsli_n_s64(int64x1_t a, int64x1_t b, const int n)	a -> Dd b -> Dn 0 <= n <= 63	SLI Dd,Dn,#n	Dd -> result	v7/A32/A64
int64x2_t vsliq_n_s64(int64x2_t a, int64x2_t b, const int n)	a -> Vd.2D b -> Vn.2D 0 <= n <= 63	SLI Vd.2D,Vn.2D,#n	Vd.2D -> result	v7/A32/A64
uint8x8_t vsli_n_u8(uint8x8_t a, uint8x8_t b, const int n)	a -> Vd.8B b -> Vn.8B 0 <= n <= 7	SLI Vd.8B,Vn.8B,#n	Vd.8B -> result	v7/A32/A64
uint8x16_t vsliq_n_u8(uint8x16_t a, uint8x16_t b, const int n)	a -> Vd.16B b -> Vn.16B 0 <= n <= 7	SLI Vd.16B,Vn.16B,#n	Vd.16B -> result	v7/A32/A64
uint16x4_t vsli_n_u16(uint16x4_t a, uint16x4_t b, const int n)	a -> Vd.4H b -> Vn.4H 0 <= n <= 15	SLI Vd.4H,Vn.4H,#n	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vsliq_n_u16(uint16x8_t a, uint16x8_t b, const int n)</code>	<code>a -> Vd.8H b -> Vn.8H 0 <= n <= 15</code>	<code>SLI Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vsli_n_u32(uint32x2_t a, uint32x2_t b, const int n)</code>	<code>a -> Vd.2S b -> Vn.2S 0 <= n <= 31</code>	<code>SLI Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vsliq_n_u32(uint32x4_t a, uint32x4_t b, const int n)</code>	<code>a -> Vd.4S b -> Vn.4S 0 <= n <= 31</code>	<code>SLI Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vsli_n_u64(uint64x1_t a, uint64x1_t b, const int n)</code>	<code>a -> Dd b -> Dn 0 <= n <= 63</code>	<code>SLI Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vsliq_n_u64(uint64x2_t a, uint64x2_t b, const int n)</code>	<code>a -> Vd.2D b -> Vn.2D 0 <= n <= 63</code>	<code>SLI Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x1_t vsli_n_p64(poly64x1_t a, poly64x1_t b, const int n)	a -> Dd b -> Dn 0 <= n <= 63	SLI Dd,Dn,#n	Dd -> result	A32/A64
poly64x2_t vsliq_n_p64(poly64x2_t a, poly64x2_t b, const int n)	a -> Vd.2D b -> Vn.2D 0 <= n <= 63	SLI Vd.2D,Vn.2D,#n	Vd.2D -> result	A32/A64
poly8x8_t vsli_n_p8(poly8x8_t a, poly8x8_t b, const int n)	a -> Vd.8B b -> Vn.8B 0 <= n <= 7	SLI Vd.8B,Vn.8B,#n	Vd.8B -> result	v7/A32/A64
poly8x16_t vsliq_n_p8(poly8x16_t a, poly8x16_t b, const int n)	a -> Vd.16B b -> Vn.16B 0 <= n <= 7	SLI Vd.16B,Vn.16B,#n	Vd.16B -> result	v7/A32/A64
poly16x4_t vsli_n_p16(poly16x4_t a, poly16x4_t b, const int n)	a -> Vd.4H b -> Vn.4H 0 <= n <= 15	SLI Vd.4H,Vn.4H,#n	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x8_t vsliq_n_p16(poly16x8_t a, poly16x8_t b, const int n)</code>	a -> Vd.8H b -> Vn.8H 0 <= n <= 15	SLI Vd.8H,Vn.8H,#n	Vd.8H -> result	v7/A32/A64
<code>int64_t vslid_n_s64(int64_t a, int64_t b, const int n)</code>	a -> Dd b -> Dn 0 <= n <= 63	SLI Dd,Dn,#n	Dd -> result	A64
<code>uint64_t vslid_n_u64(uint64_t a, uint64_t b, const int n)</code>	a -> Dd b -> Dn 0 <= n <= 63	SLI Dd,Dn,#n	Dd -> result	A64

2.1.3.1.6 Vector shift left and insert**2.1.3.2 Right**

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vshr_n_s8(int8x8_t a, const int n)</code>	a -> Vn.8B 1 <= n <= 8	SSHR Vd.8B,Vn.8B,#n	Vd.8B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vshrq_n_s8(int8x16_t a, const int n)</code>	<code>a -> Vn.16B 1 <= n <= 8</code>	<code>SSHR Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vshr_n_s16(int16x4_t a, const int n)</code>	<code>a -> Vn.4H 1 <= n <= 16</code>	<code>SSHR Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vshrq_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 16</code>	<code>SSHR Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vshr_n_s32(int32x2_t a, const int n)</code>	<code>a -> Vn.2S 1 <= n <= 32</code>	<code>SSHR Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vshrq_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 32</code>	<code>SSHR Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vshr_n_s64(int64x1_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 64</code>	<code>SSHR Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vshrq_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 64</code>	<code>SSHR Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vshr_n_u8(uint8x8_t a, const int n)</code>	<code>a -> Vn.8B 1 <= n <= 8</code>	<code>USHR Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vshrq_n_u8(uint8x16_t a, const int n)</code>	<code>a -> Vn.16B 1 <= n <= 8</code>	<code>USHR Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vshr_n_u16(uint16x4_t a, const int n)</code>	<code>a -> Vn.4H 1 <= n <= 16</code>	<code>USHR Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vshrq_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 16</code>	<code>USHR Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vshr_n_u32(uint32x2_t a, const int n)</code>	<code>a -> Vn.2S 1 <= n <= 32</code>	<code>USHR Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vshrq_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 32</code>	<code>USHR Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vshr_n_u64(uint64x1_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 64</code>	<code>USHR Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vshrq_n_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D</code> <code>1 <= n <= 64</code>	<code>USHR Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64_t vshrd_n_s64(int64_t a, const int n)</code>	<code>a -> Dn</code> <code>1 <= n <= 64</code>	<code>SSHR Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>uint64_t vshrd_n_u64(uint64_t a, const int n)</code>	<code>a -> Dn</code> <code>1 <= n <= 64</code>	<code>USHR Dd,Dn,#n</code>	<code>Dd -> result</code>	A64

2.1.3.2.1 Vector shift right

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vrshr_n_s8(int8x8_t a, const int n)</code>	<code>a -> Vn.8B</code> <code>1 <= n <= 8</code>	<code>SRSR Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vrshrq_n_s8(int8x16_t a, const int n)</code>	<code>a -> Vn.16B</code> <code>1 <= n <= 8</code>	<code>SRSR Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vrshr_n_s16(int16x4_t a, const int n)</code>	<code>a -> Vn.4H 1 <= n <= 16</code>	<code>SRSHR Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vrshrq_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 16</code>	<code>SRSHR Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vrshr_n_s32(int32x2_t a, const int n)</code>	<code>a -> Vn.2S 1 <= n <= 32</code>	<code>SRSHR Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vrshrq_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 32</code>	<code>SRSHR Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vrshr_n_s64(int64x1_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 64</code>	<code>SRSHR Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vrshrq_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 64</code>	<code>SRSHR Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vrshr_n_u8(uint8x8_t a, const int n)</code>	<code>a -> Vn.8B 1 <= n <= 8</code>	<code>URSHR Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vrshrq_n_u8(uint8x16_t a, const int n)</code>	<code>a -> Vn.16B 1 <= n <= 8</code>	URSHR Vd.16B,Vn.16B,#n	Vd.16B -> result	v7/A32/A64
<code>uint16x4_t vrshrq_n_u16(uint16x4_t a, const int n)</code>	<code>a -> Vn.4H 1 <= n <= 16</code>	URSHR Vd.4H,Vn.4H,#n	Vd.4H -> result	v7/A32/A64
<code>uint16x8_t vrshrq_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 16</code>	URSHR Vd.8H,Vn.8H,#n	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vrshrq_n_u32(uint32x2_t a, const int n)</code>	<code>a -> Vn.2S 1 <= n <= 32</code>	URSHR Vd.2S,Vn.2S,#n	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vrshrq_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 32</code>	URSHR Vd.4S,Vn.4S,#n	Vd.4S -> result	v7/A32/A64
<code>uint64x1_t vrshrq_n_u64(uint64x1_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 64</code>	URSHR Dd,Dn,#n	Dd -> result	v7/A32/A64
<code>uint64x2_t vrshrq_n_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 64</code>	URSHR Vd.2D,Vn.2D,#n	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vrshrd_n_s64(int64_t a, const int n)</code>	a -> Dn 1 <= n <= 64	SRRH Dd,Dn,#n	Dd -> result	A64
<code>uint64_t vrshrd_n_u64(uint64_t a, const int n)</code>	a -> Dn 1 <= n <= 64	URSHR Dd,Dn,#n	Dd -> result	A64

2.1.3.2.2 Vector rounding shift right

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vsra_n_s8(int8x8_t a, int8x8_t b, const int n)</code>	a -> Vd.8B b -> Vn.8B 1 <= n <= 8	SSRA Vd.8B,Vn.8B,#n	Vd.8B -> result	v7/A32/A64
<code>int8x16_t vsraq_n_s8(int8x16_t a, int8x16_t b, const int n)</code>	a -> Vd.16B b -> Vn.16B 1 <= n <= 8	SSRA Vd.16B,Vn.16B,#n	Vd.16B -> result	v7/A32/A64
<code>int16x4_t vsra_n_s16(int16x4_t a, int16x4_t b, const int n)</code>	a -> Vd.4H b -> Vn.4H 1 <= n <= 16	SSRA Vd.4H,Vn.4H,#n	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vsraq_n_s16(int16x8_t a, int16x8_t b, const int n)</code>	<code>a -> Vd.8H b -> Vn.8H 1 <= n <= 16</code>	<code>SSRA Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vsra_n_s32(int32x2_t a, int32x2_t b, const int n)</code>	<code>a -> Vd.2S b -> Vn.2S 1 <= n <= 32</code>	<code>SSRA Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vsraq_n_s32(int32x4_t a, int32x4_t b, const int n)</code>	<code>a -> Vd.4S b -> Vn.4S 1 <= n <= 32</code>	<code>SSRA Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vsra_n_s64(int64x1_t a, int64x1_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	<code>SSRA Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vsraq_n_s64(int64x2_t a, int64x2_t b, const int n)</code>	<code>a -> Vd.2D b -> Vn.2D 1 <= n <= 64</code>	<code>SSRA Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vsra_n_u8(uint8x8_t a, uint8x8_t b, const int n)</code>	<code>a -> Vd.8B b -> Vn.8B 1 <= n <= 8</code>	USRA Vd.8B,Vn.8B,#n	Vd.8B -> result	v7/A32/A64
<code>uint8x16_t vsraq_n_u8(uint8x16_t a, uint8x16_t b, const int n)</code>	<code>a -> Vd.16B b -> Vn.16B 1 <= n <= 8</code>	USRA Vd.16B,Vn.16B,#n	Vd.16B -> result	v7/A32/A64
<code>uint16x4_t vsra_n_u16(uint16x4_t a, uint16x4_t b, const int n)</code>	<code>a -> Vd.4H b -> Vn.4H 1 <= n <= 16</code>	USRA Vd.4H,Vn.4H,#n	Vd.4H -> result	v7/A32/A64
<code>uint16x8_t vsraq_n_u16(uint16x8_t a, uint16x8_t b, const int n)</code>	<code>a -> Vd.8H b -> Vn.8H 1 <= n <= 16</code>	USRA Vd.8H,Vn.8H,#n	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vsra_n_u32(uint32x2_t a, uint32x2_t b, const int n)</code>	<code>a -> Vd.2S b -> Vn.2S 1 <= n <= 32</code>	USRA Vd.2S,Vn.2S,#n	Vd.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsraq_n_u32(uint32x4_t a, uint32x4_t b, const int n)</code>	<code>a -> Vd.4S b -> Vn.4S 1 <= n <= 32</code>	<code>USRA Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vsra_n_u64(uint64x1_t a, uint64x1_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	<code>USRA Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vsraq_n_u64(uint64x2_t a, uint64x2_t b, const int n)</code>	<code>a -> Vd.2D b -> Vn.2D 1 <= n <= 64</code>	<code>USRA Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64_t vsrad_n_s64(int64_t a, int64_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	<code>SSRA Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>uint64_t vsrad_n_u64(uint64_t a, uint64_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	<code>USRA Dd,Dn,#n</code>	<code>Dd -> result</code>	A64

2.1.3.2.3 Vector shift right and accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vrsra_n_s8(int8x8_t a, int8x8_t b, const int n)</code>	<code>a -> Vd.8B b -> Vn.8B 1 <= n <= 8</code>	<code>SRSRA Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vrsraq_n_s8(int8x16_t a, int8x16_t b, const int n)</code>	<code>a -> Vd.16B b -> Vn.16B 1 <= n <= 8</code>	<code>SRSRA Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vrsra_n_s16(int16x4_t a, int16x4_t b, const int n)</code>	<code>a -> Vd.4H b -> Vn.4H 1 <= n <= 16</code>	<code>SRSRA Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vrsraq_n_s16(int16x8_t a, int16x8_t b, const int n)</code>	<code>a -> Vd.8H b -> Vn.8H 1 <= n <= 16</code>	<code>SRSRA Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vrsra_n_s32(int32x2_t a, int32x2_t b, const int n)</code>	<code>a -> Vd.2S b -> Vn.2S 1 <= n <= 32</code>	<code>SRSRA Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vrsraq_n_s32(int32x4_t a, int32x4_t b, const int n)</code>	<code>a -> Vd.4S b -> Vn.4S 1 <= n <= 32</code>	<code>SRSRA Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vrsra_n_s64(int64x1_t a, int64x1_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	<code>SRSRA Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vrsraq_n_s64(int64x2_t a, int64x2_t b, const int n)</code>	<code>a -> Vd.2D b -> Vn.2D 1 <= n <= 64</code>	<code>SRSRA Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vrsra_n_u8(uint8x8_t a, uint8x8_t b, const int n)</code>	<code>a -> Vd.8B b -> Vn.8B 1 <= n <= 8</code>	<code>URSRA Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vrsraq_n_u8(uint8x16_t a, uint8x16_t b, const int n)</code>	<code>a -> Vd.16B b -> Vn.16B 1 <= n <= 8</code>	<code>URSRA Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vrsra_n_u16(uint16x4_t a, uint16x4_t b, const int n)</code>	<code>a -> Vd.4H b -> Vn.4H 1 <= n <= 16</code>	URSRA Vd.4H,Vn.4H,#n	Vd.4H -> result	v7/A32/A64
<code>uint16x8_t vrsraq_n_u16(uint16x8_t a, uint16x8_t b, const int n)</code>	<code>a -> Vd.8H b -> Vn.8H 1 <= n <= 16</code>	URSRA Vd.8H,Vn.8H,#n	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vrsra_n_u32(uint32x2_t a, uint32x2_t b, const int n)</code>	<code>a -> Vd.2S b -> Vn.2S 1 <= n <= 32</code>	URSRA Vd.2S,Vn.2S,#n	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vrsraq_n_u32(uint32x4_t a, uint32x4_t b, const int n)</code>	<code>a -> Vd.4S b -> Vn.4S 1 <= n <= 32</code>	URSRA Vd.4S,Vn.4S,#n	Vd.4S -> result	v7/A32/A64
<code>uint64x1_t vrsra_n_u64(uint64x1_t a, uint64x1_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	URSRA Dd,Dn,#n	Dd -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vrsraq_n_u64(uint64x2_t a, uint64x2_t b, const int n)</code>	<code>a -> Vd.2D b -> Vn.2D 1 <= n <= 64</code>	URSRA Vd.2D,Vn.2D,#n	Vd.2D -> result	v7/A32/A64
<code>int64_t vrsrad_n_s64(int64_t a, int64_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	SRSRA Dd,Dn,#n	Dd -> result	A64
<code>uint64_t vrsrad_n_u64(uint64_t a, uint64_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	URSRA Dd,Dn,#n	Dd -> result	A64

2.1.3.2.4 Vector rounding shift right and accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vshrn_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	SHRN Vd.8B,Vn.8H,#n	Vd.8B -> result	v7/A32/A64
<code>int16x4_t vshrn_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	SHRN Vd.4H,Vn.4S,#n	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vshrn_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>SHRN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vshrn_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>SHRN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vshrn_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>SHRN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vshrn_n_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>SHRN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8x16_t vshrn_high_n_s16(int8x8_t r, int16x8_t a, const int n)</code>	<code>r -> Vd.8B a -> Vn.8H 1 <= n <= 8</code>	<code>SHRN2 Vd.16B,Vn.8H,#n</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vshrn_high_n_s32(int16x4_t r, int32x4_t a, const int n)</code>	<code>r -> Vd.4H a -> Vn.4S 1 <= n <= 16</code>	<code>SHRN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vshrn_high_n_s64(int32x2_t r, int64x2_t a, const int n)</code>	<code>r -> Vd.2S a -> Vn.2D 1 <= n <= 32</code>	<code>SHRN2 Vd.4S,Vn.2D,#n</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vshrn_high_n_u16(uint8x8_t r, uint16x8_t a, const int n)</code>	<code>r -> Vd.8B a -> Vn.8H 1 <= n <= 8</code>	<code>SHRN2 Vd.16B,Vn.8H,#n</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vshrn_high_n_u32(uint16x4_t r, uint32x4_t a, const int n)</code>	<code>r -> Vd.4H a -> Vn.4S 1 <= n <= 16</code>	<code>SHRN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vshrn_high_n_u64(uint32x2_t r, uint64x2_t a, const int n)</code>	<code>r -> Vd.2S a -> Vn.2D 1 <= n <= 32</code>	<code>SHRN2 Vd.4S,Vn.2D,#n</code>	<code>Vd.4S -> result</code>	A64

2.1.3.2.5 Vector shift right and narrow

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vqshrun_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>SQSHRUN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vqshrun_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>SQSHRUN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vqshrun_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>SQSHRUN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8_t vqshrunh_n_s16(int16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 8</code>	<code>SQSHRUN Bd,Hn,#n</code>	<code>Bd -> result</code>	A64
<code>uint16_t vqshruns_n_s32(int32_t a, const int n)</code>	<code>a -> Sn 1 <= n <= 16</code>	<code>SQSHRUN Hd,Sn,#n</code>	<code>Hd -> result</code>	A64
<code>uint32_t vqshrund_n_s64(int64_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 32</code>	<code>SQSHRUN Sd,Dn,#n</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vqshrun_high_n_s16(uint8x8_t r, int16x8_t a, const int n)</code>	<code>r -> Vd.8B a -> Vn.8H 1 <= n <= 8</code>	<code>SQSHRUN2 Vd.16B,Vn.8H,#n</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vqshrun_high_n_s32(uint16x4_t r, int32x4_t a, const int n)</code>	<code>r -> Vd.4H a -> Vn.4S 1 <= n <= 16</code>	<code>SQSHRUN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vqshrun_high_n_s64(uint32x2_t r, int64x2_t a, const int n)</code>	<code>r -> Vd.2S a -> Vn.2D 1 <= n <= 32</code>	<code>SQSHRUN2 Vd.4S,Vn.2D,#n</code>	<code>Vd.4S -> result</code>	A64
<code>int8x8_t vqshrn_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>SQSHRN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vqshrn_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>SQSHRN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vqshrn_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>SQSHRN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vqshrn_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>UQSHRN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vqshrn_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>UQSHRN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vqshrn_n_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>UQSHRN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8_t vqshrn_h_n_s16(int16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 8</code>	<code>SQSHRN Bd,Hn,#n</code>	<code>Bd -> result</code>	A64
<code>int16_t vqshrns_n_s32(int32_t a, const int n)</code>	<code>a -> Sn 1 <= n <= 16</code>	<code>SQSHRN Hd,Sn,#n</code>	<code>Hd -> result</code>	A64
<code>int32_t vqshrnd_n_s64(int64_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 32</code>	<code>SQSHRN Sd,Dn,#n</code>	<code>Sd -> result</code>	A64
<code>uint8_t vqshrn_h_n_u16(uint16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 8</code>	<code>UQSHRN Bd,Hn,#n</code>	<code>Bd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t</code> <code>vqshrns_n_u32(</code> <code>uint32_t</code> <code>a,</code> <code>const int</code> <code>n)</code>	<code>a -> Sn</code> <code>1 <= n <= 16</code>	<code>UQSHRN Hd,Sn,#n</code>	<code>Hd -> result</code>	A64
<code>uint32_t</code> <code>vqshrnd_n_u64(</code> <code>uint64_t</code> <code>a,</code> <code>const int</code> <code>n)</code>	<code>a -> Dn</code> <code>1 <= n <= 32</code>	<code>UQSHRN Sd,Dn,#n</code>	<code>Sd -> result</code>	A64
<code>int8x16_t</code> <code>vqshrn_high_n_s16(</code> <code>int8x8_t</code> <code>r,</code> <code>int16x8_t</code> <code>a,</code> <code>const int</code> <code>n)</code>	<code>r -> Vd.8B</code> <code>a -> Vn.8H</code> <code>1 <= n <= 8</code>	<code>SQSHRN2 Vd.16B,Vn.8H,#n</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t</code> <code>vqshrn_high_n_s32(</code> <code>int16x4_t</code> <code>r,</code> <code>int32x4_t</code> <code>a,</code> <code>const int</code> <code>n)</code>	<code>r -> Vd.4H</code> <code>a -> Vn.4S</code> <code>1 <= n <= 16</code>	<code>SQSHRN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t</code> <code>vqshrn_high_n_s64(</code> <code>int32x2_t</code> <code>r,</code> <code>int64x2_t</code> <code>a,</code> <code>const int</code> <code>n)</code>	<code>r -> Vd.2S</code> <code>a -> Vn.2D</code> <code>1 <= n <= 32</code>	<code>SQSHRN2 Vd.4S,Vn.2D,#n</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t</code> <code>vqshrn_high_n_u16(</code> <code>uint8x8_t</code> <code>r,</code> <code>uint16x8_t</code> <code>a,</code> <code>const int</code> <code>n)</code>	<code>r -> Vd.8B</code> <code>a -> Vn.8H</code> <code>1 <= n <= 8</code>	<code>UQSHRN2 Vd.16B,Vn.8H,#n</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vqshrn_high_n_u32(uint16x4_t r, uint32x4_t a, const int n)</code>	<code>r -> Vd.4H a -> Vn.4S 1 <= n <= 16</code>	<code>UQSHRN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vqshrn_high_n_u64(uint32x2_t r, uint64x2_t a, const int n)</code>	<code>r -> Vd.2S a -> Vn.2D 1 <= n <= 32</code>	<code>UQSHRN2 Vd.4S,Vn.2D,#n</code>	<code>Vd.4S -> result</code>	A64

2.1.3.2.6 Vector saturating shift right and narrow

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vqrshrun_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>SQRSHRUN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vqrshrun_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>SQRSHRUN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vqrshrun_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>SQRSHRUN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8_t vqrshrunh_n_s16(int16_t a, const int n)</code>	a -> Hn 1 <= n <= 8	SQRSHRUN Bd,Hn,#n	Bd -> result	A64
<code>uint16_t vqrshruns_n_s32(int32_t a, const int n)</code>	a -> Sn 1 <= n <= 16	SQRSHRUN Hd,Sn,#n	Hd -> result	A64
<code>uint32_t vqrshrund_n_s64(int64_t a, const int n)</code>	a -> Dn 1 <= n <= 32	SQRSHRUN Sd,Dn,#n	Sd -> result	A64
<code>uint8x16_t vqrshrun_high_n_s16(uint8x8_t r, int16x8_t a, const int n)</code>	r -> Vd.8B a -> Vn.8H 1 <= n <= 8	SQRSHRUN2 Vd.16B,Vn.8H,#n	Vd.16B -> result	A64
<code>uint16x8_t vqrshrun_high_n_s32(uint16x4_t r, int32x4_t a, const int n)</code>	r -> Vd.4H a -> Vn.4S 1 <= n <= 16	SQRSHRUN2 Vd.8H,Vn.4S,#n	Vd.8H -> result	A64
<code>uint32x4_t vqrshrun_high_n_s64(uint32x2_t r, int64x2_t a, const int n)</code>	r -> Vd.2S a -> Vn.2D 1 <= n <= 32	SQRSHRUN2 Vd.4S,Vn.2D,#n	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vqrshrn_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>SQRSHRN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vqrshrn_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>SQRSHRN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vqrshrn_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>SQRSHRN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vqrshrn_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>UQRSHRN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vqrshrn_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>UQRSHRN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vqrshrn_n_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>UQRSHRN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8_t vqrshrn_h_n_s16(int16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 8</code>	<code>SQRSHRN Bd,Hn,#n</code>	<code>Bd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16_t vqrshrns_n_s32(int32_t a, const int n)</code>	a -> Sn 1 <= n <= 16	SQRSHRN Hd,Sn,#n	Hd -> result	A64
<code>int32_t vqrshrnd_n_s64(int64_t a, const int n)</code>	a -> Dn 1 <= n <= 32	SQRSHRN Sd,Dn,#n	Sd -> result	A64
<code>uint8_t vqrshrnh_n_u16(uint16_t a, const int n)</code>	a -> Hn 1 <= n <= 8	UQRSHRN Bd,Hn,#n	Bd -> result	A64
<code>uint16_t vqrshrns_n_u32(uint32_t a, const int n)</code>	a -> Sn 1 <= n <= 16	UQRSHRN Hd,Sn,#n	Hd -> result	A64
<code>uint32_t vqrshrnd_n_u64(uint64_t a, const int n)</code>	a -> Dn 1 <= n <= 32	UQRSHRN Sd,Dn,#n	Sd -> result	A64
<code>int8x16_t vqrshrnh_high_n_s16(int8x8_t r, int16x8_t a, const int n)</code>	r -> Vd.8B a -> Vn.8H 1 <= n <= 8	SQRSHRN2 Vd.16B,Vn.8H,#n	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vqrshrn_high_n_s32(int16x4_t r, int32x4_t a, const int n)</code>	<code>r -> Vd.4H a -> Vn.4S 1 <= n <= 16</code>	<code>SQRSHRN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vqrshrn_high_n_s64(int32x2_t r, int64x2_t a, const int n)</code>	<code>r -> Vd.2S a -> Vn.2D 1 <= n <= 32</code>	<code>SQRSHRN2 Vd.4S,Vn.2D,#n</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vqrshrn_high_n_u16(uint8x8_t r, uint16x8_t a, const int n)</code>	<code>r -> Vd.8B a -> Vn.8H 1 <= n <= 8</code>	<code>UQRSHRN2 Vd.16B,Vn.8H,#n</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vqrshrn_high_n_u32(uint16x4_t r, uint32x4_t a, const int n)</code>	<code>r -> Vd.4H a -> Vn.4S 1 <= n <= 16</code>	<code>UQRSHRN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vqrshrn_high_n_u64(uint32x2_t r, uint64x2_t a, const int n)</code>	<code>r -> Vd.2S a -> Vn.2D 1 <= n <= 32</code>	<code>UQRSHRN2 Vd.4S,Vn.2D,#n</code>	<code>Vd.4S -> result</code>	A64

2.1.3.2.7 Vector saturating rounding shift right and narrow

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vrshrn_n_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>RSHRN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vrshrn_n_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>RSHRN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vrshrn_n_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>RSHRN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vrshrn_n_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H 1 <= n <= 8</code>	<code>RSHRN Vd.8B,Vn.8H,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vrshrn_n_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S 1 <= n <= 16</code>	<code>RSHRN Vd.4H,Vn.4S,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vrshrn_n_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 32</code>	<code>RSHRN Vd.2S,Vn.2D,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vrshrn_high_n_s16(int8x8_t r, int16x8_t a, const int n)</code>	<code>r -> Vd.8B a -> Vn.8H 1 <= n <= 8</code>	<code>RSHRN2 Vd.16B,Vn.8H,#n</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vrshrn_high_n_s32(int16x4_t r, int32x4_t a, const int n)</code>	<code>r -> Vd.4H a -> Vn.4S 1 <= n <= 16</code>	<code>RSHRN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vrshrn_high_n_s64(int32x2_t r, int64x2_t a, const int n)</code>	<code>r -> Vd.2S a -> Vn.2D 1 <= n <= 32</code>	<code>RSHRN2 Vd.4S,Vn.2D,#n</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vrshrn_high_n_u16(uint8x8_t r, uint16x8_t a, const int n)</code>	<code>r -> Vd.8B a -> Vn.8H 1 <= n <= 8</code>	<code>RSHRN2 Vd.16B,Vn.8H,#n</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vrshrn_high_n_u32(uint16x4_t r, uint32x4_t a, const int n)</code>	<code>r -> Vd.4H a -> Vn.4S 1 <= n <= 16</code>	<code>RSHRN2 Vd.8H,Vn.4S,#n</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x4_t vrshrn_high_n_u64(uint32x2_t r, uint64x2_t a, const int n)	r -> 32(Vd) a -> Vn.2D 1 <= n <= 32	RSHRN2 Vd.4S,Vn.2D,#n	Vd.4S -> result	A64

2.1.3.2.8 Vector rounding shift right and narrow

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8x8_t vsri_n_s8(int8x8_t a, int8x8_t b, const int n)	a -> Vd.8B b -> Vn.8B 1 <= n <= 8	SRI Vd.8B,Vn.8B,#n	Vd.8B -> result	v7/A32/A64
int8x16_t vsriq_n_s8(int8x16_t a, int8x16_t b, const int n)	a -> Vd.16B b -> Vn.16B 1 <= n <= 8	SRI Vd.16B,Vn.16B,#n	Vd.16B -> result	v7/A32/A64
int16x4_t vsri_n_s16(int16x4_t a, int16x4_t b, const int n)	a -> Vd.4H b -> Vn.4H 1 <= n <= 16	SRI Vd.4H,Vn.4H,#n	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vsriq_n_s16(int16x8_t a, int16x8_t b, const int n)</code>	<code>a -> Vd.8H b -> Vn.8H 1 <= n <= 16</code>	<code>SRI Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vsri_n_s32(int32x2_t a, int32x2_t b, const int n)</code>	<code>a -> Vd.2S b -> Vn.2S 1 <= n <= 32</code>	<code>SRI Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vsriq_n_s32(int32x4_t a, int32x4_t b, const int n)</code>	<code>a -> Vd.4S b -> Vn.4S 1 <= n <= 32</code>	<code>SRI Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vsri_n_s64(int64x1_t a, int64x1_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	<code>SRI Dd,Dn,#n</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vsriq_n_s64(int64x2_t a, int64x2_t b, const int n)</code>	<code>a -> Vd.2D b -> Vn.2D 1 <= n <= 64</code>	<code>SRI Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vsri_n_u8(uint8x8_t a, uint8x8_t b, const int n)</code>	<code>a -> Vd.8B b -> Vn.8B 1 <= n <= 8</code>	<code>SRI Vd.8B,Vn.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vsriq_n_u8(uint8x16_t a, uint8x16_t b, const int n)</code>	<code>a -> Vd.16B b -> Vn.16B 1 <= n <= 8</code>	<code>SRI Vd.16B,Vn.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vsri_n_u16(uint16x4_t a, uint16x4_t b, const int n)</code>	<code>a -> Vd.4H b -> Vn.4H 1 <= n <= 16</code>	<code>SRI Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vsriq_n_u16(uint16x8_t a, uint16x8_t b, const int n)</code>	<code>a -> Vd.8H b -> Vn.8H 1 <= n <= 16</code>	<code>SRI Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vsri_n_u32(uint32x2_t a, uint32x2_t b, const int n)</code>	<code>a -> Vd.2S b -> Vn.2S 1 <= n <= 32</code>	<code>SRI Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x4_t vsriq_n_u32(uint32x4_t a, uint32x4_t b, const int n)	a -> Vd.4S b -> Vn.4S 1 <= n <= 32	SRI Vd.4S,Vn.4S,#n	Vd.4S -> result	v7/A32/A64
uint64x1_t vsri_n_u64(uint64x1_t a, uint64x1_t b, const int n)	a -> Dd b -> Dn 1 <= n <= 64	SRI Dd,Dn,#n	Dd -> result	v7/A32/A64
uint64x2_t vsriq_n_u64(uint64x2_t a, uint64x2_t b, const int n)	a -> Vd.2D b -> Vn.2D 1 <= n <= 64	SRI Vd.2D,Vn.2D,#n	Vd.2D -> result	v7/A32/A64
poly64x1_t vsri_n_p64(poly64x1_t a, poly64x1_t b, const int n)	a -> Dd b -> Dn 1 <= n <= 64	SRI Dd,Dn,#n	Dd -> result	A32/A64
poly64x2_t vsriq_n_p64(poly64x2_t a, poly64x2_t b, const int n)	a -> Vd.2D b -> Vn.2D 1 <= n <= 64	SRI Vd.2D,Vn.2D,#n	Vd.2D -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x8_t vsri_n_p8(poly8x8_t a, poly8x8_t b, const int n)	a -> Vd.8B b -> Vn.8B 1 <= n <= 8	SRI Vd.8B,Vn.8B,#n	Vd.8B -> result	v7/A32/A64
poly8x16_t vsriq_n_p8(poly8x16_t a, poly8x16_t b, const int n)	a -> Vd.16B b -> Vn.16B 1 <= n <= 8	SRI Vd.16B,Vn.16B,#n	Vd.16B -> result	v7/A32/A64
poly16x4_t vsri_n_p16(poly16x4_t a, poly16x4_t b, const int n)	a -> Vd.4H b -> Vn.4H 1 <= n <= 16	SRI Vd.4H,Vn.4H,#n	Vd.4H -> result	v7/A32/A64
poly16x8_t vsriq_n_p16(poly16x8_t a, poly16x8_t b, const int n)	a -> Vd.8H b -> Vn.8H 1 <= n <= 16	SRI Vd.8H,Vn.8H,#n	Vd.8H -> result	v7/A32/A64
int64_t vsrid_n_s64(int64_t a, int64_t b, const int n)	a -> Dd b -> Dn 1 <= n <= 64	SRI Dd,Dn,#n	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64_t vsrid_n_u64(uint64_t a, uint64_t b, const int n)</code>	<code>a -> Dd b -> Dn 1 <= n <= 64</code>	<code>SRI Dd,Dn,#n</code>	<code>Dd -> result</code>	A64

2.1.3.2.9 Vector shift right and insert

2.1.4 Data type conversion

2.1.4.1 Conversions

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vcvts32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTZS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vcvttq_s32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTZS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint32x2_t vcvtu32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTZU Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcvttq_u32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTZU Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vcvtn_s32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTNS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>int32x4_t vcvtnq_s32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTNS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>uint32x2_t vcvtn_u32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTNU Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>uint32x4_t vcvtnq_u32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTNU Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x2_t vcvtm_s32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTMS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>int32x4_t vcvtmq_s32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTMS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>uint32x2_t vcvtm_u32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTMU Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>uint32x4_t vcvtmq_u32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTMU Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x2_t vcvtp_s32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTPS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vcvtpq_s32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTPS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>uint32x2_t vcvtp_u32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTPU Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>uint32x4_t vcvtpq_u32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTPU Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x2_t vcvta_s32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTAS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>int32x4_t vcvtaq_s32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTAS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>uint32x2_t vcvta_u32_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FCVTAU Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	A32/A64
<code>uint32x4_t vcvtaq_u32_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FCVTAU Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32_t vcvts_s32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTZS Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>uint32_t vcvts_u32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTZU Sd,Sn</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t vcvtns_s32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTNS Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>uint32_t vcvtns_u32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTNU Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>int32_t vcvtms_s32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTMS Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>uint32_t vcvtms_u32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTMU Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>int32_t vcvtps_s32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTPS Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>uint32_t vcvtps_u32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTPU Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>int32_t vcvtas_s32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTAS Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>uint32_t vcvtas_u32_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>FCVTAU Sd,Sn</code>	<code>Sd -> result</code>	A64
<code>int64x1_t vcvt_s64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTZS Dd,Dn</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vcvttq_s64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTZS Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcvtt_u64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTZU Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcvttq_u64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTZU Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int64x1_t vcvtn_s64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTNS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vcvtnq_s64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTNS Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcvtn_u64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTNU Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcvtnq_u64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTNU Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int64x1_t vcvtm_s64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTMS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vcvtmq_s64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTMS Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vcvtm_u64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTMU Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcvtmq_u64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTMU Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int64x1_t vcvtp_s64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTPS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vcvtpq_s64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTPS Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcvtp_u64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTPU Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcvtpq_u64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTPU Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int64x1_t vcvta_s64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTAS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vcvtaq_s64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTAS Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcvta_u64_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FCVTAU Dd,Dn</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vcvtaq_u64_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>FCVTAU Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int64_t vcvt_d_s64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTZS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64_t vcvt_d_u64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTZU Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64_t vcvtnd_s64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTNS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64_t vcvtnd_u64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTNU Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64_t vcvtmd_s64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTMS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64_t vcvtmd_u64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTMU Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64_t vcvt_p_s64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTPS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64_t vcvt_p_u64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTPU Dd,Dn</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vcvtd_s64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTAS Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>uint64_t vcvtd_u64_f64(float64_t a)</code>	<code>a -> Dn</code>	<code>FCVTAU Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int32x2_t vcvtn_s32_f32(float32x2_t a, const int n)</code>	<code>a -> Vn.2S</code> <code>1 <= n <= 32</code>	<code>FCVTZS Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vcvtnq_s32_f32(float32x4_t a, const int n)</code>	<code>a -> Vn.4S</code> <code>1 <= n <= 32</code>	<code>FCVTZS Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint32x2_t vcvtn_u32_f32(float32x2_t a, const int n)</code>	<code>a -> Vn.2S</code> <code>1 <= n <= 32</code>	<code>FCVTZU Vd.2S,Vn.2S,#n</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vcvtnq_u32_f32(float32x4_t a, const int n)</code>	<code>a -> Vn.4S</code> <code>1 <= n <= 32</code>	<code>FCVTZU Vd.4S,Vn.4S,#n</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int32_t vcvts_n_s32_f32(float32_t a, const int n)</code>	<code>a -> Sn</code> <code>1 <= n <= 32</code>	<code>FCVTZS Sd,Sn,#n</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32_t vcvts_n_u32_f32(float32_t a, const int n)</code>	<code>a -> Sn 1 <= n <= 32</code>	<code>FCVTZU Sd,Sn,#n</code>	<code>Sd -> result</code>	A64
<code>int64x1_t vcvt_n_s64_f64(float64x1_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 64</code>	<code>FCVTZS Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vcvtq_n_s64_f64(float64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 64</code>	<code>FCVTZS Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	A64
<code>uint64x1_t vcvt_n_u64_f64(float64x1_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 64</code>	<code>FCVTZU Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vcvtq_n_u64_f64(float64x2_t a, const int n)</code>	<code>a -> Vn.2D 1 <= n <= 64</code>	<code>FCVTZU Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	A64
<code>int64_t vcvtd_n_s64_f64(float64_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 64</code>	<code>FCVTZS Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>uint64_t vcvtd_n_u64_f64(float64_t a, const int n)</code>	<code>a -> Dn 1 <= n <= 64</code>	<code>FCVTZU Dd,Dn,#n</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vcvtf_f32_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	SCVTF Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
<code>float32x4_t vcvttq_f32_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	SCVTF Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
<code>float32x2_t vcvtf_f32_u32(uint32x2_t a)</code>	<code>a -> Vn.2S</code>	UCVTF Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
<code>float32x4_t vcvttq_f32_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	UCVTF Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
<code>float32_t vcvts_f32_s32(int32_t a)</code>	<code>a -> Sn</code>	SCVTF Sd,Sn	Sd -> result	A64
<code>float32_t vcvts_f32_u32(uint32_t a)</code>	<code>a -> Sn</code>	UCVTF Sd,Sn	Sd -> result	A64
<code>float64x1_t vcvtf_f64_s64(int64x1_t a)</code>	<code>a -> Dn</code>	SCVTF Dd,Dn	Dd -> result	A64
<code>float64x2_t vcvttq_f64_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	SCVTF Vd.2D,Vn.2D	Vd.2D -> result	A64
<code>float64x1_t vcvtf_f64_u64(uint64x1_t a)</code>	<code>a -> Dn</code>	UCVTF Dd,Dn	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2_t vcvttq_f64_u64(uint64x2_t a)</code>	<code>a -> Vn.2D</code>	UCVTF Vd.2D,Vn.2D	Vd.2D -> result	A64
<code>float64_t vcvtd_f64_s64(int64_t a)</code>	<code>a -> Dn</code>	SCVTF Dd,Dn	Dd -> result	A64
<code>float64_t vcvtd_f64_u64(uint64_t a)</code>	<code>a -> Dn</code>	UCVTF Dd,Dn	Dd -> result	A64
<code>float32x2_t vcvtn_f32_s32(int32x2_t a, const int n)</code>	<code>a -> Vn.2S</code> <code>1 <= n <= 32</code>	SCVTF Vd.2S,Vn.2S,#n	Vd.2S -> result	v7/A32/A64
<code>float32x4_t vcvtn_f32_s32(int32x4_t a, const int n)</code>	<code>a -> Vn.4S</code> <code>1 <= n <= 32</code>	SCVTF Vd.4S,Vn.4S,#n	Vd.4S -> result	v7/A32/A64
<code>float32x2_t vcvtn_f32_u32(uint32x2_t a, const int n)</code>	<code>a -> Vn.2S</code> <code>1 <= n <= 32</code>	UCVTF Vd.2S,Vn.2S,#n	Vd.2S -> result	v7/A32/A64
<code>float32x4_t vcvtn_f32_u32(uint32x4_t a, const int n)</code>	<code>a -> Vn.4S</code> <code>1 <= n <= 32</code>	UCVTF Vd.4S,Vn.4S,#n	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32_t vcvts_n_f32_s32(int32_t a, const int n)</code>	<code>a -> Sn</code> <code>1 <= n <= 32</code>	<code>SCVTF Sd,Sn,#n</code>	<code>Sd -> result</code>	A64
<code>float32_t vcvts_n_f32_u32(uint32_t a, const int n)</code>	<code>a -> Sn</code> <code>1 <= n <= 32</code>	<code>UCVTF Sd,Sn,#n</code>	<code>Sd -> result</code>	A64
<code>float64x1_t vcvtn_f64_s64(int64x1_t a, const int n)</code>	<code>a -> Dn</code> <code>1 <= n <= 64</code>	<code>SCVTF Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vcvtn_f64_s64(int64x2_t a, const int n)</code>	<code>a -> Vn.2D</code> <code>1 <= n <= 64</code>	<code>SCVTF Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	A64
<code>float64x1_t vcvtn_f64_u64(uint64x1_t a, const int n)</code>	<code>a -> Dn</code> <code>1 <= n <= 64</code>	<code>UCVTF Dd,Dn,#n</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vcvtn_f64_u64(uint64x2_t a, const int n)</code>	<code>a -> Vn.2D</code> <code>1 <= n <= 64</code>	<code>UCVTF Vd.2D,Vn.2D,#n</code>	<code>Vd.2D -> result</code>	A64
<code>float64_t vcvtd_n_f64_s64(int64_t a, const int n)</code>	<code>a -> Dn</code> <code>1 <= n <= 64</code>	<code>SCVTF Dd,Dn,#n</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64_t vcvtd_n_f64_u64(uint64_t a, const int n)</code>	<code>a -> Dn</code> <code>1 <= n <= 64</code>	UCVTF Dd,Dn,#n	Dd -> result	A64
<code>float16x4_t vcvf_f16_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	FCVTN Vd.4H,Vn.4S	Vd.4H -> result	v7/A32/A64
<code>float16x8_t vcvf_high_f16_f32(float16x4_t r, float32x4_t a)</code>	<code>r -> Vd.4H</code> <code>a -> Vn.4S</code>	FCVTN2 Vd.8H,Vn.4S	Vd.8H -> result	A64
<code>float32x2_t vcvf_f32_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	FCVTN Vd.2S,Vn.2D	Vd.2S -> result	A64
<code>float32x4_t vcvf_high_f32_f64(float32x2_t r, float64x2_t a)</code>	<code>r -> Vd.2S</code> <code>a -> Vn.2D</code>	FCVTN2 Vd.4S,Vn.2D	Vd.4S -> result	A64
<code>float32x4_t vcvf_f32_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	FCVTL Vd.4S,Vn.4H	Vd.4S -> result	v7/A32/A64
<code>float32x4_t vcvf_high_f32_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	FCVTL2 Vd.4S,Vn.8H	Vd.4S -> result	A64
<code>float64x2_t vcvf_f64_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	FCVTL Vd.2D,Vn.2S	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x2_t vcvth_high_f64_f32(float32x4_t a)	a -> Vn.4S	FCVTL2 Vd.2D,Vn.4S	Vd.2D -> result	A64
float32x2_t vcvtx_f32_f64(float64x2_t a)	a -> Vn.2D	FCVTXN Vd.2S,Vn.2D	Vd.2S -> result	A64
float32_t vcvtxd_f32_f64(float64_t a)	a -> Dn	FCVTXN Sd,Dn	Sd -> result	A64
float32x4_t vcvtx_high_f32_f64(float32x2_t r, float64x2_t a)	r -> Vd.2S a -> Vn.2D	FCVTXN2 Vd.4S,Vn.2D	Vd.4S -> result	A64

2.1.4.2 Reinterpret casts

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int16x4_t vreinterpret_s16_s8(int8x8_t a)	a -> Vd.8B	NOP	Vd.4H -> result	v7/A32/A64
int32x2_t vreinterpret_s32_s8(int8x8_t a)	a -> Vd.8B	NOP	Vd.2S -> result	v7/A32/A64
float32x2_t vreinterpret_f32_s8(int8x8_t a)	a -> Vd.8B	NOP	Vd.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vreinterpret_u8_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>mfloat8x8_t vreinterpret_mf8_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x1_t vreinterpret_s64_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>float64x1_t vreinterpret_f64_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.1D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly64x1_t vreinterpret_p64_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>float16x4_t vreinterpret_f16_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int32x2_t vreinterpret_s32_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x2_t vreinterpret_f32_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vreinterpret_u8_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x4_t vreinterpret_p16_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>mfloating8x8_t vreinterpret_mf8_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x1_t vreinterpret_s64_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>float64x1_t vreinterpret_f64_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>poly64x1_t vreinterpret_p64_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>float16x4_t vreinterpret_f16_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vreinterpret_s16_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vreinterpret_f32_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vreinterpret_u8_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>mfloat8x8_t vreinterpret_mf8_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x1_t vreinterpret_s64_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x1_t vreinterpret_f64_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>poly64x1_t vreinterpret_p64_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>float16x4_t vreinterpret_f16_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vreinterpret_s16_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vreinterpret_s32_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vreinterpret_u8_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x8_t vreinterpret_p8_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	Vd.8B -> result	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	Vd.4H -> result	v7/A32/A64
<code>mf8x8_t vreinterpret_mf8_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	Vd.8B -> result	A64
<code>uint64x1_t vreinterpret_u64_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	Vd.1D -> result	v7/A32/A64
<code>int64x1_t vreinterpret_s64_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	Vd.1D -> result	v7/A32/A64
<code>float64x1_t vreinterpret_f64_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	Vd.1D -> result	A64
<code>poly64x1_t vreinterpret_p64_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	Vd.1D -> result	A32/A64
<code>poly64x1_t vreinterpret_p64_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.1D -> result	A64
<code>float16x4_t vreinterpret_f16_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vreinterpret_s8_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vreinterpret_s16_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vreinterpret_s32_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x2_t vreinterpret_f32_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>mfloat8x8_t vreinterpret_mf8_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vreinterpret_u64_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.1D -> result	v7/A32/A64
<code>int64x1_t vreinterpret_s64_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.1D -> result	v7/A32/A64
<code>float64x1_t vreinterpret_f64_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.1D -> result	A64
<code>poly64x1_t vreinterpret_p64_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.1D -> result	A32/A64
<code>float16x4_t vreinterpret_f16_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.4H -> result	v7/A32/A64
<code>int8x8_t vreinterpret_s8_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.8B -> result	v7/A32/A64
<code>int16x4_t vreinterpret_s16_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.4H -> result	v7/A32/A64
<code>int32x2_t vreinterpret_s32_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.2S -> result	v7/A32/A64
<code>float32x2_t vreinterpret_f32_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vreinterpret_u8_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>mfloat8x8_t vreinterpret_mf8_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x1_t vreinterpret_s64_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>float64x1_t vreinterpret_f64_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>poly64x1_t vreinterpret_p64_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vreinterpret_f16_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vreinterpret_s16_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vreinterpret_s32_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x2_t vreinterpret_f32_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vreinterpret_u8_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mffloat8x8_t vreinterpret_mf8_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x1_t vreinterpret_s64_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>float64x1_t vreinterpret_f64_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>poly64x1_t vreinterpret_p64_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>float16x4_t vreinterpret_f16_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vreinterpret_s16_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vreinterpret_s32_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vreinterpret_f32_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vreinterpret_u8_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint64x1_t vreinterpret_u64_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x1_t vreinterpret_s64_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>float64x1_t vreinterpret_f64_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>poly64x1_t vreinterpret_p64_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.1D -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vreinterpret_f16_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>int16x4_t vreinterpret_s16_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	A64
<code>int32x2_t vreinterpret_s32_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	A64
<code>float32x2_t vreinterpret_f32_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	A64
<code>uint8x8_t vreinterpret_u8_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint16x4_t vreinterpret_u16_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	A64
<code>uint32x2_t vreinterpret_u32_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.2S -> result</code>	A64
<code>poly16x4_t vreinterpret_p16_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vreinterpret_u64_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.1D -> result	A64
<code>int64x1_t vreinterpret_s64_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.1D -> result	A64
<code>float64x1_t vreinterpret_f64_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.1D -> result	A64
<code>poly64x1_t vreinterpret_p64_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.1D -> result	A64
<code>float16x4_t vreinterpret_f16_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.4H -> result	A64
<code>int8x8_t vreinterpret_s8_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.8B -> result	v7/A32/A64
<code>int16x4_t vreinterpret_s16_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.4H -> result	v7/A32/A64
<code>int32x2_t vreinterpret_s32_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.2S -> result	v7/A32/A64
<code>float32x2_t vreinterpret_f32_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vreinterpret_u8_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>mfloat8x8_t vreinterpret_mf8_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x1_t vreinterpret_s64_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>float64x1_t vreinterpret_f64_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>poly64x1_t vreinterpret_p64_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vreinterpret_f16_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vreinterpret_s16_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vreinterpret_s32_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x2_t vreinterpret_f32_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vreinterpret_u8_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x4_t vreinterpret_p16_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>mfloat8x8_t vreinterpret_mf8_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>int64x1_t vreinterpret_s64_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>float64x1_t vreinterpret_f64_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>poly64x1_t vreinterpret_p64_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>float16x4_t vreinterpret_f16_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vreinterpret_s16_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vreinterpret_s32_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vreinterpret_f32_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vreinterpret_u8_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>poly8x8_t vreinterpret_p8_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>mfloat8x8_t vreinterpret_mf8_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	v7/A32/A64
<code>float64x1_t vreinterpret_f64_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vreinterpret_u64_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>float16x4_t vreinterpret_f16_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int8x8_t vreinterpret_s8_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vreinterpret_s16_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vreinterpret_s32_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x2_t vreinterpret_f32_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8x8_t vreinterpret_u8_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vreinterpret_u16_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vreinterpret_u32_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x8_t vreinterpret_p8_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.8B -> result	v7/A32/A64
<code>poly16x4_t vreinterpret_p16_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.4H -> result	v7/A32/A64
<code>mfloating8x8_t vreinterpret_mf8_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.8B -> result	A64
<code>uint64x1_t vreinterpret_u64_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.1D -> result	v7/A32/A64
<code>int64x1_t vreinterpret_s64_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.1D -> result	v7/A32/A64
<code>float64x1_t vreinterpret_f64_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.1D -> result	A64
<code>poly64x1_t vreinterpret_p64_f16(float16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	Vd.1D -> result	A32/A64
<code>int16x8_t vreinterpretq_s16_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int32x4_t vreinterpretq_s32_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vreinterpretq_f32_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>uint64x2_t vreinterpretq_u64_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2_t vreinterpretq_f64_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vreinterpretq_p64_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A32/A64
<code>poly128_t vreinterpretq_p128_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.1Q -> result</code>	A32/A64
<code>float16x8_t vreinterpretq_f16_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int32x4_t vreinterpretq_s32_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x4_t vreinterpretq_f32_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vreinterpretq_u32_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>uint64x2_t vreinterpretq_u64_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vreinterpretq_p64_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	A32/A64
<code>poly128_t vreinterpretq_p128_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.1Q -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vreinterpretq_f16_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x8_t vreinterpretq_s16_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>float32x4_t vreinterpretq_f32_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x16_t vreinterpretq_mf8_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.16B -> result	A64
<code>uint64x2_t vreinterpretq_u64_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	A64
<code>poly64x2_t vreinterpretq_p64_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	A32/A64
<code>poly128_t vreinterpretq_p128_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.1Q -> result	A32/A64
<code>float16x8_t vreinterpretq_f16_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>int16x8_t vreinterpretq_s16_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vreinterpretq_s32_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>uint64x2_t vreinterpretq_u64_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2_t vreinterpretq_f64_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	A64
<code>poly64x2_t vreinterpretq_p64_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	A32/A64
<code>poly128_t vreinterpretq_p128_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.1Q -> result	A32/A64
<code>poly64x2_t vreinterpretq_p64_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.2D -> result	A64
<code>poly128_t vreinterpretq_p128_f64(float64x2_t a)</code>	<code>a -> Vd.1Q</code>	NOP	Vd.2D -> result	A64
<code>float16x8_t vreinterpretq_f16_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>int16x8_t vreinterpretq_s16_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int32x4_t vreinterpretq_s32_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vreinterpretq_f32_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>uint64x2_t vreinterpretq_u64_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly64x2_t vreinterpretq_p64_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A32/A64
<code>poly128_t vreinterpretq_p128_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.1Q -> result</code>	A32/A64
<code>float16x8_t vreinterpretq_f16_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x8_t vreinterpretq_s16_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vreinterpretq_s32_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x4_t vreinterpretq_f32_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x16_t vreinterpretq_p8_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>mf8x16_t vreinterpretq_mf8_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.16B -> result	A64
<code>uint64x2_t vreinterpretq_u64_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.2D -> result	A64
<code>poly64x2_t vreinterpretq_p64_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.2D -> result	A32/A64
<code>poly128_t vreinterpretq_p128_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.1Q -> result	A32/A64
<code>float16x8_t vreinterpretq_f16_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vreinterpretq_s8_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>int16x8_t vreinterpretq_s16_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int32x4_t vreinterpretq_s32_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.4S -> result	v7/A32/A64
<code>float32x4_t vreinterpretq_f32_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.4S -> result	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>mfloating8x16_t vreinterpretq_mf8_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vreinterpretq_u64_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	A64
<code>poly64x2_t vreinterpretq_p64_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.2D -> result	A32/A64
<code>poly128_t vreinterpretq_p128_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.1Q -> result	A32/A64
<code>float16x8_t vreinterpretq_f16_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>int16x8_t vreinterpretq_s16_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int32x4_t vreinterpretq_s32_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vreinterpretq_f32_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint64x2_t vreinterpretq_u64_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vreinterpretq_p64_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly128_t vreinterpretq_p128_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.1Q -> result</code>	A32/A64
<code>float16x8_t vreinterpretq_f16_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vreinterpretq_s16_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vreinterpretq_s32_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	A64
<code>float32x4_t vreinterpretq_f32_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vreinterpretq_u8_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vreinterpretq_u16_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vreinterpretq_u32_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x8_t vreinterpretq_p16_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	A64
<code>uint64x2_t vreinterpretq_u64_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>int64x2_t vreinterpretq_s64_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>float64x2_t vreinterpretq_f64_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vreinterpretq_p64_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>poly128_t vreinterpretq_p128_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.1Q -> result</code>	A64
<code>float16x8_t vreinterpretq_f16_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	A64
<code>int8x16_t vreinterpretq_s8_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x8_t vreinterpretq_s16_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vreinterpretq_s32_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x4_t vreinterpretq_f32_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>uint64x2_t vreinterpretq_u64_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x2_t vreinterpretq_f64_p16(poly16x8_t a)	a -> Vd.8H	NOP	Vd.2D -> result	A64
poly64x2_t vreinterpretq_p64_p16(poly16x8_t a)	a -> Vd.8H	NOP	Vd.2D -> result	A32/A64
poly128_t vreinterpretq_p128_p16(poly16x8_t a)	a -> Vd.8H	NOP	Vd.1Q -> result	A32/A64
float16x8_t vreinterpretq_f16_p16(poly16x8_t a)	a -> Vd.8H	NOP	Vd.8H -> result	v7/A32/A64
int8x16_t vreinterpretq_s8_u64(uint64x2_t a)	a -> Vd.2D	NOP	Vd.16B -> result	v7/A32/A64
int16x8_t vreinterpretq_s16_u64(uint64x2_t a)	a -> Vd.2D	NOP	Vd.8H -> result	v7/A32/A64
int32x4_t vreinterpretq_s32_u64(uint64x2_t a)	a -> Vd.2D	NOP	Vd.4S -> result	v7/A32/A64
float32x4_t vreinterpretq_f32_u64(uint64x2_t a)	a -> Vd.2D	NOP	Vd.4S -> result	v7/A32/A64
uint8x16_t vreinterpretq_u8_u64(uint64x2_t a)	a -> Vd.2D	NOP	Vd.16B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vreinterpretq_u16_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>int64x2_t vreinterpretq_s64_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.2D -> result</code>	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vreinterpretq_p64_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.2D -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly128_t vreinterpretq_p128_s64(int64x2_t a)</code>	<code>a -> Vd.1Q</code>	NOP	Vd.2D -> result	A32/A64
<code>poly64x2_t vreinterpretq_p64_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.2D -> result	A32/A64
<code>poly128_t vreinterpretq_p128_u64(uint64x2_t a)</code>	<code>a -> Vd.1Q</code>	NOP	Vd.2D -> result	A32/A64
<code>float16x8_t vreinterpretq_f16_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>int16x8_t vreinterpretq_s16_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int32x4_t vreinterpretq_s32_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.4S -> result	v7/A32/A64
<code>float32x4_t vreinterpretq_f32_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.4S -> result	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.16B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vreinterpretq_u16_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.4S -> result	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.16B -> result	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.16B -> result	A64
<code>uint64x2_t vreinterpretq_u64_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>uint64x2_t vreinterpretq_u64_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.2D -> result	A32/A64
<code>float16x8_t vreinterpretq_f16_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.8H -> result	v7/A32/A64
<code>int8x16_t vreinterpretq_s8_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.16B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vreinterpretq_s16_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vreinterpretq_s32_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x4_t vreinterpretq_f32_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vreinterpretq_u8_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x8_t vreinterpretq_u16_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vreinterpretq_u32_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x16_t vreinterpretq_p8_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x8_t vreinterpretq_p16_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	v7/A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vreinterpretq_u64_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>int64x2_t vreinterpretq_s64_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.2D -> result	v7/A32/A64
<code>float64x2_t vreinterpretq_f64_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.2D -> result	A64
<code>poly64x2_t vreinterpretq_p64_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.2D -> result	A32/A64
<code>poly128_t vreinterpretq_p128_f16(float16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	Vd.1Q -> result	A32/A64
<code>int8x8_t vreinterpret_s8_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.8B -> result	A64
<code>int16x4_t vreinterpret_s16_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.4H -> result	A64
<code>int32x2_t vreinterpret_s32_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.2S -> result	A64
<code>uint8x8_t vreinterpret_u8_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vreinterpret_u16_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A64
<code>uint32x2_t vreinterpret_u32_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	A64
<code>poly8x8_t vreinterpret_p8_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>poly16x4_t vreinterpret_p16_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A64
<code>mfloat8x8_t vreinterpret_mf8_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>int64x1_t vreinterpret_s64_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>float16x4_t vreinterpret_f16_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A64
<code>float32x2_t vreinterpret_f32_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vreinterpretq_s8_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vreinterpretq_s16_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vreinterpretq_s32_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vreinterpretq_u8_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vreinterpretq_u16_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vreinterpretq_u32_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.4S -> result</code>	A64
<code>poly8x16_t vreinterpretq_p8_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>poly16x8_t vreinterpretq_p16_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A64
<code>mfloat8x16_t vreinterpretq_mf8_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vreinterpretq_u64_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.2D -> result	A64
<code>int64x2_t vreinterpretq_s64_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.2D -> result	A64
<code>float16x8_t vreinterpretq_f16_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.8H -> result	A64
<code>float32x4_t vreinterpretq_f32_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	Vd.4S -> result	A64
<code>int8x8_t vreinterpret_s8_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.8B -> result	A32/A64
<code>int16x4_t vreinterpret_s16_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.4H -> result	A32/A64
<code>int32x2_t vreinterpret_s32_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.2S -> result	A32/A64
<code>uint8x8_t vreinterpret_u8_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.8B -> result	A32/A64
<code>uint16x4_t vreinterpret_u16_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	Vd.4H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vreinterpret_u32_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.2S -> result</code>	A32/A64
<code>poly8x8_t vreinterpret_p8_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	A32/A64
<code>poly16x4_t vreinterpret_p16_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>mfloat8x8_t vreinterpret_mf8_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.8B -> result</code>	A32/A64
<code>int64x1_t vreinterpret_s64_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>float64x1_t vreinterpret_f64_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>float16x4_t vreinterpret_f16_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>int8x16_t vreinterpretq_s8_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>int16x8_t vreinterpretq_s16_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vreinterpretq_s32_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.4S -> result</code>	A32/A64
<code>uint8x16_t vreinterpretq_u8_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>uint16x8_t vreinterpretq_u16_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>uint32x4_t vreinterpretq_u32_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.4S -> result</code>	A32/A64
<code>poly8x16_t vreinterpretq_p8_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>poly16x8_t vreinterpretq_p16_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>int64x2_t vreinterpretq_s64_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.2D -> result</code>	A32/A64
<code>float64x2_t vreinterpretq_f64_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vreinterpretq_f16_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>int8x16_t vreinterpretq_s8_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>int16x8_t vreinterpretq_s16_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>int32x4_t vreinterpretq_s32_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.4S -> result</code>	A32/A64
<code>uint8x16_t vreinterpretq_u8_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>uint16x8_t vreinterpretq_u16_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>uint32x4_t vreinterpretq_u32_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.4S -> result</code>	A32/A64
<code>poly8x16_t vreinterpretq_p8_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>poly16x8_t vreinterpretq_p16_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.8H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x16_t vreinterpretq_mf8_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	Vd.16B -> result	A64
<code>uint64x2_t vreinterpretq_u64_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	Vd.2D -> result	A32/A64
<code>int64x2_t vreinterpretq_s64_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	Vd.2D -> result	A32/A64
<code>float64x2_t vreinterpretq_f64_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	Vd.2D -> result	A64
<code>float16x8_t vreinterpretq_f16_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	Vd.8H -> result	A32/A64
<code>mfloat8x8_t vreinterpret_mf8_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.8B -> result	A64
<code>mfloat8x16_t vreinterpretq_mf8_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	Vd.16B -> result	A64
<code>uint8x8_t vreinterpret_u8_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint8x16_t vreinterpretq_u8_mf8(mfloat8x16_t a)	a -> Vd.16B	NOP	Vd.16B -> result	A64

2.1.5 Move

2.1.5.1 Narrow

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8x8_t vmovn_s16(int16x8_t a)	a -> Vn.8H	XTN Vd.8B,Vn.8H	Vd.8B -> result	v7/A32/A64
int16x4_t vmovn_s32(int32x4_t a)	a -> Vn.4S	XTN Vd.4H,Vn.4S	Vd.4H -> result	v7/A32/A64
int32x2_t vmovn_s64(int64x2_t a)	a -> Vn.2D	XTN Vd.2S,Vn.2D	Vd.2S -> result	v7/A32/A64
uint8x8_t vmovn_u16(uint16x8_t a)	a -> Vn.8H	XTN Vd.8B,Vn.8H	Vd.8B -> result	v7/A32/A64
uint16x4_t vmovn_u32(uint32x4_t a)	a -> Vn.4S	XTN Vd.4H,Vn.4S	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vmovn_u64(uint64x2_t a)</code>	<code>a -> Vn.2D</code>	XTN <code>Vd.2S, Vn.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8x16_t vmovn_high_s16(int8x8_t r, int16x8_t a)</code>	<code>r -> Vd.8B</code> <code>a -> Vn.8H</code>	XTN2 <code>Vd.16B, Vn.8H</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x8_t vmovn_high_s32(int16x4_t r, int32x4_t a)</code>	<code>r -> Vd.4H</code> <code>a -> Vn.4S</code>	XTN2 <code>Vd.8H, Vn.4S</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vmovn_high_s64(int32x2_t r, int64x2_t a)</code>	<code>r -> Vd.2S</code> <code>a -> Vn.2D</code>	XTN2 <code>Vd.4S, Vn.2D</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x16_t vmovn_high_u16(uint8x8_t r, uint16x8_t a)</code>	<code>r -> Vd.8B</code> <code>a -> Vn.8H</code>	XTN2 <code>Vd.16B, Vn.8H</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x8_t vmovn_high_u32(uint16x4_t r, uint32x4_t a)</code>	<code>r -> Vd.4H</code> <code>a -> Vn.4S</code>	XTN2 <code>Vd.8H, Vn.4S</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vmovn_high_u64(uint32x2_t r, uint64x2_t a)</code>	<code>r -> Vd.2S</code> <code>a -> Vn.2D</code>	XTN2 <code>Vd.4S, Vn.2D</code>	<code>Vd.4S -> result</code>	v7/A32/A64

2.1.5.2 Widen

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vmovl_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>SSHLL Vd.8H,Vn.8B,#0</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x4_t vmovl_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>SSHLL Vd.4S,Vn.4H,#0</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x2_t vmovl_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>SSHLL Vd.2D,Vn.2S,#0</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint16x8_t vmovl_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>USHLL Vd.8H,Vn.8B,#0</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x4_t vmovl_u16(uint16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>USHLL Vd.4S,Vn.4H,#0</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vmovl_u32(uint32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>USHLL Vd.2D,Vn.2S,#0</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int16x8_t vmovl_high_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>SSHLL2 Vd.8H,Vn.16B,#0</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vmovl_high_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SSHLL2 Vd.4S,Vn.8H,#0</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vmovl_high_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>SSHLL2 Vd.2D,Vn.4S,#0</code>	<code>Vd.2D -> result</code>	A64
<code>uint16x8_t vmovl_high_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>USHLL2 Vd.8H,Vn.16B,#0</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vmovl_high_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>USHLL2 Vd.4S,Vn.8H,#0</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vmovl_high_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>USHLL2 Vd.2D,Vn.4S,#0</code>	<code>Vd.2D -> result</code>	A64

2.1.5.3 Saturating narrow

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vqmovn_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SQXTN Vd.8B,Vn.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vqmovn_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>SQXTN Vd.4H,Vn.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vqmovn_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>SQXTN Vd.2S,Vn.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vqmovn_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>UQXTN Vd.8B,Vn.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vqmovn_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>UQXTN Vd.4H,Vn.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vqmovn_u64(uint64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>UQXTN Vd.2S,Vn.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int8_t vqmovnh_s16(int16_t a)</code>	<code>a -> Hn</code>	<code>SQXTN Bd,Hn</code>	<code>Bd -> result</code>	A64
<code>int16_t vqmovns_s32(int32_t a)</code>	<code>a -> Sn</code>	<code>SQXTN Hd,Sn</code>	<code>Hd -> result</code>	A64
<code>int32_t vqmovnd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>SQXTN Sd,Dn</code>	<code>Sd -> result</code>	A64
<code>uint8_t vqmovnh_u16(uint16_t a)</code>	<code>a -> Hn</code>	<code>UQXTN Bd,Hn</code>	<code>Bd -> result</code>	A64
<code>uint16_t vqmovns_u32(uint32_t a)</code>	<code>a -> Sn</code>	<code>UQXTN Hd,Sn</code>	<code>Hd -> result</code>	A64
<code>uint32_t vqmovnd_u64(uint64_t a)</code>	<code>a -> Dn</code>	<code>UQXTN Sd,Dn</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vqmovn_high_s16(int8x8_t r, int16x8_t a)</code>	<code>r -> Vd.8B a -> Vn.8H</code>	<code>SQXTN2 Vd.16B,Vn.8H</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vqmovn_high_s32(int16x4_t r, int32x4_t a)</code>	<code>r -> Vd.4H a -> Vn.4S</code>	<code>SQXTN2 Vd.8H,Vn.4S</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vqmovn_high_s64(int32x2_t r, int64x2_t a)</code>	<code>r -> Vd.2S a -> Vn.2D</code>	<code>SQXTN2 Vd.4S,Vn.2D</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x16_t vqmovn_high_u16(uint8x8_t r, uint16x8_t a)</code>	<code>r -> Vd.8B a -> Vn.8H</code>	<code>UQXTN2 Vd.16B,Vn.8H</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vqmovn_high_u32(uint16x4_t r, uint32x4_t a)</code>	<code>r -> Vd.4H a -> Vn.4S</code>	<code>UQXTN2 Vd.8H,Vn.4S</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vqmovn_high_u64(uint32x2_t r, uint64x2_t a)</code>	<code>r -> Vd.2S a -> Vn.2D</code>	<code>UQXTN2 Vd.4S,Vn.2D</code>	<code>Vd.4S -> result</code>	A64
<code>uint8x8_t vqmovun_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SQXTUN Vd.8B,Vn.8H</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vqmovun_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>SQXTUN Vd.4H,Vn.4S</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint32x2_t vqmovun_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>SQXTUN Vd.2S,Vn.2D</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint8_t vqmovunh_s16(int16_t a)</code>	<code>a -> Hn</code>	<code>SQXTUN Bd,Hn</code>	<code>Bd -> result</code>	A64
<code>uint16_t vqmovuns_s32(int32_t a)</code>	<code>a -> Sn</code>	<code>SQXTUN Hd,Sn</code>	<code>Hd -> result</code>	A64
<code>uint32_t vqmovund_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>SQXTUN Sd,Dn</code>	<code>Sd -> result</code>	A64
<code>uint8x16_t vqmovun_high_s16(uint8x8_t r, int16x8_t a)</code>	<code>r -> Vd.8B</code> <code>a -> Vn.8H</code>	<code>SQXTUN2 Vd.16B,Vn.8H</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vqmovun_high_s32(uint16x4_t r, int32x4_t a)</code>	<code>r -> Vd.4H</code> <code>a -> Vn.4S</code>	<code>SQXTUN2 Vd.8H,Vn.4S</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x4_t vqmovun_high_s64(uint32x2_t r, int64x2_t a)</code>	<code>r -> Vd.2S</code> <code>a -> Vn.2D</code>	<code>SQXTUN2 Vd.4S,Vn.2D</code>	<code>Vd.4S -> result</code>	A64

2.1.6 Scalar arithmetic

2.1.6.1 Vector multiply-accumulate by scalar

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>int16x4_t vmla_lane_s16(int16x4_t a, int16x4_t b, int16x4_t v, const int lane)</pre>	<pre>a -> Vd.4H b -> Vn.4H v -> Vm.4H 0 <= lane <= 3</pre>	MLA Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	v7/A32/A64
<pre>int16x8_t vmlaq_lane_s16(int16x8_t a, int16x8_t b, int16x4_t v, const int lane)</pre>	<pre>a -> Vd.8H b -> Vn.8H v -> Vm.4H 0 <= lane <= 3</pre>	MLA Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	v7/A32/A64
<pre>int32x2_t vmla_lane_s32(int32x2_t a, int32x2_t b, int32x2_t v, const int lane)</pre>	<pre>a -> Vd.2S b -> Vn.2S v -> Vm.2S 0 <= lane <= 1</pre>	MLA Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	v7/A32/A64
<pre>int32x4_t vmlaq_lane_s32(int32x4_t a, int32x4_t b, int32x2_t v, const int lane)</pre>	<pre>a -> Vd.4S b -> Vn.4S v -> Vm.2S 0 <= lane <= 1</pre>	MLA Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x4_t vmla_lane_u16(uint16x4_t a, uint16x4_t b, uint16x4_t v, const int lane)	a -> Vd.4H b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	MLA Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	v7/A32/A64
uint16x8_t vmlaq_lane_u16(uint16x8_t a, uint16x8_t b, uint16x4_t v, const int lane)	a -> Vd.8H b -> Vn.8H v -> Vm.4H 0 <= lane <= 3	MLA Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	v7/A32/A64
uint32x2_t vmla_lane_u32(uint32x2_t a, uint32x2_t b, uint32x2_t v, const int lane)	a -> Vd.2S b -> Vn.2S v -> Vm.2S 0 <= lane <= 1	MLA Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	v7/A32/A64
uint32x4_t vmlaq_lane_u32(uint32x4_t a, uint32x4_t b, uint32x2_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.2S 0 <= lane <= 1	MLA Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	v7/A32/A64
float32x2_t vmla_lane_f32(float32x2_t a, float32x2_t b, float32x2_t v, const int lane)	0 <= lane <= 1	RESULT[I] = a[i] + (b[i] * v[lane]) for i = 0 to 1N/A		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vmlaq_lane_f32(float32x4_t a, float32x4_t b, float32x2_t v, const int lane)	0 <= lane <= 1	RESULT[I] = a[i] + (b[i] * v[lane]) for i = 0 to 3N/A		v7/A32/A64
int16x4_t vmla_laneq_s16(int16x4_t a, int16x4_t b, int16x8_t v, const int lane)	a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	MLA Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
int16x8_t vmlaq_laneq_s16(int16x8_t a, int16x8_t b, int16x8_t v, const int lane)	a -> Vd.8H b -> Vn.8H v -> Vm.8H 0 <= lane <= 7	MLA Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64
int32x2_t vmla_laneq_s32(int32x2_t a, int32x2_t b, int32x4_t v, const int lane)	a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	MLA Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64
int32x4_t vmlaq_laneq_s32(int32x4_t a, int32x4_t b, int32x4_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.4S 0 <= lane <= 3	MLA Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x4_t vmla_laneq_u16(uint16x4_t a, uint16x4_t b, uint16x8_t v, const int lane)	a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	MLA Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
uint16x8_t vmlaq_laneq_u16(uint16x8_t a, uint16x8_t b, uint16x8_t v, const int lane)	a -> Vd.8H b -> Vn.8H v -> Vm.8H 0 <= lane <= 7	MLA Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64
uint32x2_t vmla_laneq_u32(uint32x2_t a, uint32x2_t b, uint32x4_t v, const int lane)	a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	MLA Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64
uint32x4_t vmlaq_laneq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.4S 0 <= lane <= 3	MLA Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
float32x2_t vmla_laneq_f32(float32x2_t a, float32x2_t b, float32x4_t v, const int lane)	0 <= lane <= 3	RESULT[I] = a[i] + (b[i] * v[lane]) for i = 0 to 1N/A		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vmlaq_laneq_f32(float32x4_t a, float32x4_t b, float32x4_t v, const int lane)	0 <= lane <= 3	RESULT[I] = a[i] + (b[i] * v[lane]) for i = 0 to 3N/A		A64
int32x4_t vmlal_lane_s16(int32x4_t a, int16x4_t b, int16x4_t v, const int lane)	a -> Vd.4S b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	SMLAL Vd.4S,Vn.4H,Vm.H[lane]	Vd.4S -> result	v7/A32/A64
int64x2_t vmlal_lane_s32(int64x2_t a, int32x2_t b, int32x2_t v, const int lane)	a -> Vd.2D b -> Vn.2S v -> Vm.2S 0 <= lane <= 1	SMLAL Vd.2D,Vn.2S,Vm.S[lane]	Vd.2D -> result	v7/A32/A64
uint32x4_t vmlal_lane_u16(uint32x4_t a, uint16x4_t b, uint16x4_t v, const int lane)	a -> Vd.4S b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	UMLAL Vd.4S,Vn.4H,Vm.H[lane]	Vd.4S -> result	v7/A32/A64
uint64x2_t vmlal_lane_u32(uint64x2_t a, uint32x2_t b, uint32x2_t v, const int lane)	a -> Vd.2D b -> Vn.2S v -> Vm.2S 0 <= lane <= 1	UMLAL Vd.2D,Vn.2S,Vm.S[lane]	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vmlal_high_lane_s16(</code> <code>int32x4_t a,</code> <code>int16x8_t b,</code> <code>int16x4_t v,</code> <code>const int lane)</code>	<code>a -> Vd.4S</code> <code>b -> Vn.8H</code> <code>v -> Vm.4H</code> <code>0 <= lane <= 3</code>	<code>SMLAL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	<code>A64</code>
<code>int64x2_t vmlal_high_lane_s32(</code> <code>int64x2_t a,</code> <code>int32x4_t b,</code> <code>int32x2_t v,</code> <code>const int lane)</code>	<code>a -> Vd.2D</code> <code>b -> Vn.4S</code> <code>v -> Vm.2S</code> <code>0 <= lane <= 1</code>	<code>SMLAL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	<code>A64</code>
<code>uint32x4_t vmlal_high_lane_u16(</code> <code>uint32x4_t a,</code> <code>uint16x8_t b,</code> <code>uint16x4_t v,</code> <code>const int lane)</code>	<code>a -> Vd.4S</code> <code>b -> Vn.8H</code> <code>v -> Vm.4H</code> <code>0 <= lane <= 3</code>	<code>UMLAL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	<code>A64</code>
<code>uint64x2_t vmlal_high_lane_u32(</code> <code>uint64x2_t a,</code> <code>uint32x4_t b,</code> <code>uint32x2_t v,</code> <code>const int lane)</code>	<code>a -> Vd.2D</code> <code>b -> Vn.4S</code> <code>v -> Vm.2S</code> <code>0 <= lane <= 1</code>	<code>UMLAL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	<code>A64</code>
<code>int32x4_t vmlal_laneq_s16(</code> <code>int32x4_t a,</code> <code>int16x4_t b,</code> <code>int16x8_t v,</code> <code>const int lane)</code>	<code>a -> Vd.4S</code> <code>b -> Vn.4H</code> <code>v -> Vm.8H</code> <code>0 <= lane <= 7</code>	<code>SMLAL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	<code>A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int64x2_t vmlal_laneq_s32(int64x2_t a, int32x2_t b, int32x4_t v, const int lane)	a -> Vd.2D b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	SMLAL Vd.2D,Vn.2S,Vm.S[lane]	Vd.2D -> result	A64
uint32x4_t vmlal_laneq_u16(uint32x4_t a, uint16x4_t b, uint16x8_t v, const int lane)	a -> Vd.4S b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	UMLAL Vd.4S,Vn.4H,Vm.H[lane]	Vd.4S -> result	A64
uint64x2_t vmlal_laneq_u32(uint64x2_t a, uint32x2_t b, uint32x4_t v, const int lane)	a -> Vd.2D b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	UMLAL Vd.2D,Vn.2S,Vm.S[lane]	Vd.2D -> result	A64
int32x4_t vmlal_high_laneq_s16(int32x4_t a, int16x8_t b, int16x8_t v, const int lane)	a -> Vd.4S b -> Vn.8H v -> Vm.8H 0 <= lane <= 7	SMLAL2 Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A64
int64x2_t vmlal_high_laneq_s32(int64x2_t a, int32x4_t b, int32x4_t v, const int lane)	a -> Vd.2D b -> Vn.4S v -> Vm.4S 0 <= lane <= 3	SMLAL2 Vd.2D,Vn.4S,Vm.S[lane]	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vmlal_high_laneq_u16(uint32x4_t a, uint16x8_t b, uint16x8_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>UMLAL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	<code>A64</code>
<code>uint64x2_t vmlal_high_laneq_u32(uint64x2_t a, uint32x4_t b, uint32x4_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>UMLAL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	<code>A64</code>
<code>int16x4_t vmla_n_s16(int16x4_t a, int16x4_t b, int16_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.H[0]</code>	<code>MLA Vd.4H,Vn.4H,Vm.H[0]</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>int16x8_t vmlaq_n_s16(int16x8_t a, int16x8_t b, int16_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.H[0]</code>	<code>MLA Vd.8H,Vn.8H,Vm.H[0]</code>	<code>Vd.8H -> result</code>	<code>v7/A32/A64</code>
<code>int32x2_t vmla_n_s32(int32x2_t a, int32x2_t b, int32_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.S[0]</code>	<code>MLA Vd.2S,Vn.2S,Vm.S[0]</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vmlaq_n_s32(int32x4_t a, int32x4_t b, int32_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.S[0]</code>	<code>MLA Vd.4S,Vn.4S,Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint16x4_t vmla_n_u16(uint16x4_t a, uint16x4_t b, uint16_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.H[0]</code>	<code>MLA Vd.4H,Vn.4H,Vm.H[0]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vmlaq_n_u16(uint16x8_t a, uint16x8_t b, uint16_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.H[0]</code>	<code>MLA Vd.8H,Vn.8H,Vm.H[0]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vmla_n_u32(uint32x2_t a, uint32x2_t b, uint32_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.S[0]</code>	<code>MLA Vd.2S,Vn.2S,Vm.S[0]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vmlaq_n_u32(uint32x4_t a, uint32x4_t b, uint32_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.S[0]</code>	<code>MLA Vd.4S,Vn.4S,Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vmla_n_f32(float32x2_t a, float32x2_t b, float32_t c)</code>	N/A	RESULT[I] = a[i] + (b[i] * c) for i = 0 to 1	N/A	v7/A32/A64
<code>float32x4_t vmlaq_n_f32(float32x4_t a, float32x4_t b, float32_t c)</code>	N/A	RESULT[I] = a[i] + (b[i] * c) for i = 0 to 3	N/A	v7/A32/A64

2.1.6.2 Vector multiply-subtract by scalar

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vmls_lane_s16(int16x4_t a, int16x4_t b, int16x4_t v, const int lane)</code>	a -> Vd.4H b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	MLS Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	v7/A32/A64
<code>int16x8_t vmlsq_lane_s16(int16x8_t a, int16x8_t b, int16x4_t v, const int lane)</code>	a -> Vd.8H b -> Vn.8H v -> Vm.4H 0 <= lane <= 3	MLS Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vmls_lane_s32(int32x2_t a, int32x2_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>MLS Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>int32x4_t vmlsq_lane_s32(int32x4_t a, int32x4_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>MLS Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	<code>v7/A32/A64</code>
<code>uint16x4_t vmls_lane_u16(uint16x4_t a, uint16x4_t b, uint16x4_t v, const int lane)</code>	<code>a -> Vd.4H b -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>MLS Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>uint16x8_t vmlsq_lane_u16(uint16x8_t a, uint16x8_t b, uint16x4_t v, const int lane)</code>	<code>a -> Vd.8H b -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>MLS Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	<code>v7/A32/A64</code>
<code>uint32x2_t vmls_lane_u32(uint32x2_t a, uint32x2_t b, uint32x2_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>MLS Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x4_t vmlsq_lane_u32(uint32x4_t a, uint32x4_t b, uint32x2_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.2S 0 <= lane <= 1	MLS Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	v7/A32/A64
float32x2_t vmls_lane_f32(float32x2_t a, float32x2_t b, float32x2_t v, const int lane)	0 <= lane <= 1	RESULT[I] = a[i] - (b[i] * v[lane]) for i = 0 to 1N/A		v7/A32/A64
float32x4_t vmlsq_lane_f32(float32x4_t a, float32x4_t b, float32x2_t v, const int lane)	0 <= lane <= 1	RESULT[I] = a[i] - (b[i] * v[lane]) for i = 0 to 3N/A		v7/A32/A64
int16x4_t vmls_laneq_s16(int16x4_t a, int16x4_t b, int16x8_t v, const int lane)	a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	MLS Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
int16x8_t vmlsq_laneq_s16(int16x8_t a, int16x8_t b, int16x8_t v, const int lane)	a -> Vd.8H b -> Vn.8H v -> Vm.8H 0 <= lane <= 7	MLS Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int32x2_t vmls_laneq_s32(int32x2_t a, int32x2_t b, int32x4_t v, const int lane)	a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	MLS Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64
int32x4_t vmlsq_laneq_s32(int32x4_t a, int32x4_t b, int32x4_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.4S 0 <= lane <= 3	MLS Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
uint16x4_t vmls_laneq_u16(uint16x4_t a, uint16x4_t b, uint16x8_t v, const int lane)	a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	MLS Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
uint16x8_t vmlsq_laneq_u16(uint16x8_t a, uint16x8_t b, uint16x8_t v, const int lane)	a -> Vd.8H b -> Vn.8H v -> Vm.8H 0 <= lane <= 7	MLS Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64
uint32x2_t vmls_laneq_u32(uint32x2_t a, uint32x2_t b, uint32x4_t v, const int lane)	a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	MLS Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x4_t vmlsq_laneq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t v, const int lane)	a -> Vd.4S b -> Vn.4S v -> Vm.4S 0 <= lane <= 3	MLS Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	A64
float32x2_t vmls_laneq_f32(float32x2_t a, float32x2_t b, float32x4_t v, const int lane)	0 <= lane <= 3	RESULT[I] = a[i] - (b[i] * v[lane]) for i = 0 to 1N/A		A64
float32x4_t vmlsq_laneq_f32(float32x4_t a, float32x4_t b, float32x4_t v, const int lane)	0 <= lane <= 3	RESULT[I] = a[i] - (b[i] * v[lane]) for i = 0 to 3N/A		A64
int32x4_t vmlsl_lane_s16(int32x4_t a, int16x4_t b, int16x4_t v, const int lane)	a -> Vd.4S b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	SMLS Vd.4S,Vn.4H,Vm.H[lane]	Vd.4S -> result	v7/A32/A64
int64x2_t vmlsl_lane_s32(int64x2_t a, int32x2_t b, int32x2_t v, const int lane)	a -> Vd.2D b -> Vn.2S v -> Vm.2S 0 <= lane <= 1	SMLS Vd.2D,Vn.2S,Vm.S[lane]	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x4_t vmlsl_lane_u16(uint32x4_t a, uint16x4_t b, uint16x4_t v, const int lane)	a -> Vd.4S b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	UMLSL Vd.4S,Vn.4H,Vm.H[lane]	Vd.4S -> result	v7/A32/A64
uint64x2_t vmlsl_lane_u32(uint64x2_t a, uint32x2_t b, uint32x2_t v, const int lane)	a -> Vd.2D b -> Vn.2S v -> Vm.2S 0 <= lane <= 1	UMLSL Vd.2D,Vn.2S,Vm.S[lane]	Vd.2D -> result	v7/A32/A64
int32x4_t vmlsl_high_lane_s16(int32x4_t a, int16x8_t b, int16x4_t v, const int lane)	a -> Vd.4S b -> Vn.8H v -> Vm.4H 0 <= lane <= 3	SMLSL2 Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A64
int64x2_t vmlsl_high_lane_s32(int64x2_t a, int32x4_t b, int32x2_t v, const int lane)	a -> Vd.2D b -> Vn.4S v -> Vm.2S 0 <= lane <= 1	SMLSL2 Vd.2D,Vn.4S,Vm.S[lane]	Vd.2D -> result	A64
uint32x4_t vmlsl_high_lane_u16(uint32x4_t a, uint16x8_t b, uint16x4_t v, const int lane)	a -> Vd.4S b -> Vn.8H v -> Vm.4H 0 <= lane <= 3	UMLSL2 Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint64x2_t vmlsl_high_lane_u32(uint64x2_t a, uint32x4_t b, uint32x2_t v, const int lane)	a -> Vd.2D b -> Vn.4S v -> Vm.2S 0 <= lane <= 1	UMLSL2 Vd.2D,Vn.4S,Vm.S[lane]	Vd.2D -> result	A64
int32x4_t vmlsl_laneq_s16(int32x4_t a, int16x4_t b, int16x8_t v, const int lane)	a -> Vd.4S b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	SMLSL Vd.4S,Vn.4H,Vm.H[lane]	Vd.4S -> result	A64
int64x2_t vmlsl_laneq_s32(int64x2_t a, int32x2_t b, int32x4_t v, const int lane)	a -> Vd.2D b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	SMLSL Vd.2D,Vn.2S,Vm.S[lane]	Vd.2D -> result	A64
uint32x4_t vmlsl_laneq_u16(uint32x4_t a, uint16x4_t b, uint16x8_t v, const int lane)	a -> Vd.4S b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	UMLSL Vd.4S,Vn.4H,Vm.H[lane]	Vd.4S -> result	A64
uint64x2_t vmlsl_laneq_u32(uint64x2_t a, uint32x2_t b, uint32x4_t v, const int lane)	a -> Vd.2D b -> Vn.2S v -> Vm.4S 0 <= lane <= 3	UMLSL Vd.2D,Vn.2S,Vm.S[lane]	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vmlsl_high_laneq_s16(int32x4_t a, int16x8_t b, int16x8_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>SMLSL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	<code>A64</code>
<code>int64x2_t vmlsl_high_laneq_s32(int64x2_t a, int32x4_t b, int32x4_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>SMLSL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	<code>A64</code>
<code>uint32x4_t vmlsl_high_laneq_u16(uint32x4_t a, uint16x8_t b, uint16x8_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>UMLSL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	<code>A64</code>
<code>uint64x2_t vmlsl_high_laneq_u32(uint64x2_t a, uint32x4_t b, uint32x4_t v, const int lane)</code>	<code>a -> Vd.2D b -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>UMLSL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	<code>A64</code>

2.1.6.3 Vector multiply by scalar

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vmul_n_s16(int16x4_t a, int16_t b)</code>	<code>a -> Vn.4H b -> Vm.H[0]</code>	<code>MUL Vd.4H,Vn.4H,Vm.H[0]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vmulq_n_s16(int16x8_t a, int16_t b)</code>	<code>a -> Vn.8H b -> Vm.H[0]</code>	<code>MUL Vd.8H,Vn.8H,Vm.H[0]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vmul_n_s32(int32x2_t a, int32_t b)</code>	<code>a -> Vn.2S b -> Vm.S[0]</code>	<code>MUL Vd.2S,Vn.2S,Vm.S[0]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vmulq_n_s32(int32x4_t a, int32_t b)</code>	<code>a -> Vn.4S b -> Vm.S[0]</code>	<code>MUL Vd.4S,Vn.4S,Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint16x4_t vmul_n_u16(uint16x4_t a, uint16_t b)</code>	<code>a -> Vn.4H b -> Vm.H[0]</code>	<code>MUL Vd.4H,Vn.4H,Vm.H[0]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vmulq_n_u16(uint16x8_t a, uint16_t b)</code>	<code>a -> Vn.8H b -> Vm.H[0]</code>	<code>MUL Vd.8H,Vn.8H,Vm.H[0]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vmul_n_u32(uint32x2_t a, uint32_t b)</code>	<code>a -> Vn.2S b -> Vm.S[0]</code>	<code>MUL Vd.2S,Vn.2S,Vm.S[0]</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vmulq_n_u32(uint32x4_t a, uint32_t b)</code>	<code>a -> Vn.4S b -> Vm.S[0]</code>	<code>MUL Vd.4S,Vn.4S,Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x2_t vmul_n_f32(float32x2_t a, float32_t b)</code>	<code>a -> Vn.2S b -> Vm.S[0]</code>	<code>FMUL Vd.2S,Vn.2S,Vm.S[0]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vmulq_n_f32(float32x4_t a, float32_t b)</code>	<code>a -> Vn.4S b -> Vm.S[0]</code>	<code>FMUL Vd.4S,Vn.4S,Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float64x1_t vmul_n_f64(float64x1_t a, float64_t b)</code>	<code>a -> Dn b -> Vm.D[0]</code>	<code>FMUL Dd,Dn,Vm.D[0]</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vmulq_n_f64(float64x2_t a, float64_t b)</code>	<code>a -> Vn.2D b -> Vm.D[0]</code>	<code>FMUL Vd.2D,Vn.2D,Vm.D[0]</code>	<code>Vd.2D -> result</code>	A64
<code>int16x4_t vmul_lane_s16(int16x4_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>MUL Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vmulq_lane_s16(int16x8_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>MUL Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vmul_lane_s32(int32x2_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>MUL Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vmulq_lane_s32(int32x4_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>MUL Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint16x4_t vmul_lane_u16(uint16x4_t a, uint16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>MUL Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vmulq_lane_u16(uint16x8_t a, uint16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>MUL Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x2_t vmul_lane_u32(uint32x2_t a, uint32x2_t v, const int lane)	a -> Vn.2S v -> Vm.2S 0 <= lane <= 1	MUL Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	v7/A32/A64
uint32x4_t vmulq_lane_u32(uint32x4_t a, uint32x2_t v, const int lane)	a -> Vn.4S v -> Vm.2S 0 <= lane <= 1	MUL Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	v7/A32/A64
float32x2_t vmul_lane_f32(float32x2_t a, float32x2_t v, const int lane)	a -> Vn.2S v -> Vm.2S 0 <= lane <= 1	FMUL Vd.2S,Vn.2S,Vm.S[lane]	Vd.2S -> result	v7/A32/A64
float32x4_t vmulq_lane_f32(float32x4_t a, float32x2_t v, const int lane)	a -> Vn.4S v -> Vm.2S 0 <= lane <= 1	FMUL Vd.4S,Vn.4S,Vm.S[lane]	Vd.4S -> result	v7/A32/A64
float64x1_t vmul_lane_f64(float64x1_t a, float64x1_t v, const int lane)	a -> Dn v -> Vm.1D 0 <= lane <= 0	FMUL Dd,Dn,Vm.D[lane]	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x2_t vmulq_lane_f64(float64x2_t a, float64x1_t v, const int lane)	a -> Vn.2D v -> Vm.1D 0 <= lane <= 0	FMUL Vd.2D,Vn.2D,Vm.D[lane]	Vd.2D -> result	A64
float32_t vmuls_lane_f32(float32_t a, float32x2_t v, const int lane)	a -> Sn v -> Vm.2S 0 <= lane <= 1	FMUL Sd,Sn,Vm.S[lane]	Sd -> result	A64
float64_t vmuld_lane_f64(float64_t a, float64x1_t v, const int lane)	a -> Dn v -> Vm.1D 0 <= lane <= 0	FMUL Dd,Dn,Vm.S[lane]	Dd -> result	A64
int16x4_t vmul_laneq_s16(int16x4_t a, int16x8_t v, const int lane)	a -> Vn.4H v -> Vm.8H 0 <= lane <= 7	MUL Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
int16x8_t vmulq_laneq_s16(int16x8_t a, int16x8_t v, const int lane)	a -> Vn.8H v -> Vm.8H 0 <= lane <= 7	MUL Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vmul_laneq_s32(int32x2_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>MUL Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vmulq_laneq_s32(int32x4_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>MUL Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>uint16x4_t vmul_laneq_u16(uint16x4_t a, uint16x8_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>MUL Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>uint16x8_t vmulq_laneq_u16(uint16x8_t a, uint16x8_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>MUL Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x2_t vmul_laneq_u32(uint32x2_t a, uint32x4_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>MUL Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vmulq_laneq_u32(uint32x4_t a, uint32x4_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>MUL Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>float32x2_t vmul_laneq_f32(float32x2_t a, float32x4_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>FMUL Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vmulq_laneq_f32(float32x4_t a, float32x4_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>FMUL Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>float64x1_t vmul_laneq_f64(float64x1_t a, float64x2_t v, const int lane)</code>	<code>a -> Dn v -> Vm.2D 0 <= lane <= 1</code>	<code>FMUL Dd,Dn,Vm.D[lane]</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vmulq_laneq_f64(float64x2_t a, float64x2_t v, const int lane)</code>	<code>a -> Vn.2D v -> Vm.2D 0 <= lane <= 1</code>	<code>FMUL Vd.2D,Vn.2D,Vm.D[lane]</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32_t vmuls_laneq_f32(float32_t a, float32x4_t v, const int lane)	a -> Sn v -> Vm.4S 0 <= lane <= 3	FMUL Sd,Sn,Vm.S[lane]	Sd -> result	A64
float64_t vmuld_laneq_f64(float64_t a, float64x2_t v, const int lane)	a -> Dn v -> Vm.2D 0 <= lane <= 1	FMUL Dd,Dn,Vm.D[lane]	Dd -> result	A64

2.1.6.4 Vector multiply by scalar and widen

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int32x4_t vmull_n_s16(int16x4_t a, int16_t b)	a -> Vn.4H b -> Vm.H[0]	SMULL Vd.4S,Vn.4H,Vm.H[0]	Vd.4S -> result	v7/A32/A64
int64x2_t vmull_n_s32(int32x2_t a, int32_t b)	a -> Vn.2S b -> Vm.S[0]	SMULL Vd.2D,Vn.2S,Vm.S[0]	Vd.2D -> result	v7/A32/A64
uint32x4_t vmull_n_u16(uint16x4_t a, uint16_t b)	a -> Vn.4H b -> Vm.H[0]	UMULL Vd.4S,Vn.4H,Vm.H[0]	Vd.4S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vmull_n_u32(uint32x2_t a, uint32_t b)</code>	<code>a -> Vn.2S b -> Vm.S[0]</code>	<code>UMULL Vd.2D,Vn.2S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int32x4_t vmull_high_n_s16(int16x8_t a, int16_t b)</code>	<code>a -> Vn.8H b -> Vm.H[0]</code>	<code>SMULL2 Vd.4S,Vn.8H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vmull_high_n_s32(int32x4_t a, int32_t b)</code>	<code>a -> Vn.4S b -> Vm.S[0]</code>	<code>SMULL2 Vd.2D,Vn.4S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	A64
<code>uint32x4_t vmull_high_n_u16(uint16x8_t a, uint16_t b)</code>	<code>a -> Vn.8H b -> Vm.H[0]</code>	<code>UMULL2 Vd.4S,Vn.8H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vmull_high_n_u32(uint32x4_t a, uint32_t b)</code>	<code>a -> Vn.4S b -> Vm.S[0]</code>	<code>UMULL2 Vd.2D,Vn.4S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	A64
<code>int32x4_t vmull_lane_s16(int16x4_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>SMULL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vmull_lane_s32(int32x2_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>SMULL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint32x4_t vmull_lane_u16(uint16x4_t a, uint16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>UMULL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x2_t vmull_lane_u32(uint32x2_t a, uint32x2_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>UMULL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>int32x4_t vmull_high_lane_s16(int16x8_t a, int16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>SMULL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vmull_high_lane_s32(int32x4_t a, int32x2_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>SMULL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vmull_high_lane_u16(uint16x8_t a, uint16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>UMULL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vmull_high_lane_u32(uint32x4_t a, uint32x2_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>UMULL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>int32x4_t vmull_laneq_s16(int16x4_t a, int16x8_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SMULL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vmull_laneq_s32(int32x2_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SMULL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>uint32x4_t vmull_laneq_u16(uint16x4_t a, uint16x8_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>UMULL Vd.4S,Vn.4H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vmull_laneq_u32(uint32x2_t a, uint32x4_t v, const int lane)</code>	<code>a -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>UMULL Vd.2D,Vn.2S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>int32x4_t vmull_high_laneq_s16(int16x8_t a, int16x8_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>SMULL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vmull_high_laneq_s32(int32x4_t a, int32x4_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>SMULL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>uint32x4_t vmull_high_laneq_u16(uint16x8_t a, uint16x8_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>UMULL2 Vd.4S,Vn.8H,Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vmull_high_laneq_u32(uint32x4_t a, uint32x4_t v, const int lane)</code>	<code>a -> Vn.4S v -> Vm.4S 0 <= lane <= 3</code>	<code>UMULL2 Vd.2D,Vn.4S,Vm.S[lane]</code>	<code>Vd.2D -> result</code>	A64

2.1.6.5 Vector multiply-accumulate by scalar and widen

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vmlal_n_s16(int32x4_t a, int16x4_t b, int16_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.H[0]</code>	<code>SMLAL Vd.4S,Vn.4H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	<code>v7/A32/A64</code>
<code>int64x2_t vmlal_n_s32(int64x2_t a, int32x2_t b, int32_t c)</code>	<code>a -> Vd.2D b -> Vn.2S c -> Vm.S[0]</code>	<code>SMLAL Vd.2D,Vn.2S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	<code>v7/A32/A64</code>
<code>uint32x4_t vmlal_n_u16(uint32x4_t a, uint16x4_t b, uint16_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.H[0]</code>	<code>UMLAL Vd.4S,Vn.4H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	<code>v7/A32/A64</code>
<code>uint64x2_t vmlal_n_u32(uint64x2_t a, uint32x2_t b, uint32_t c)</code>	<code>a -> Vd.2D b -> Vn.2S c -> Vm.S[0]</code>	<code>UMLAL Vd.2D,Vn.2S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	<code>v7/A32/A64</code>
<code>int32x4_t vmlal_high_n_s16(int32x4_t a, int16x8_t b, int16_t c)</code>	<code>a -> Vd.4S b -> Vn.8H c -> Vm.H[0]</code>	<code>SMLAL2 Vd.4S,Vn.8H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	<code>A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int64x2_t vmlal_high_n_s32(int64x2_t a, int32x4_t b, int32_t c)	a -> Vd.2D b -> Vn.4S c -> Vm.S[0]	SMLAL2 Vd.2D,Vn.4S,Vm.S[0]	Vd.2D -> result	A64
uint32x4_t vmlal_high_n_u16(uint32x4_t a, uint16x8_t b, uint16_t c)	a -> Vd.4S b -> Vn.8H c -> Vm.H[0]	UMLAL2 Vd.4S,Vn.8H,Vm.H[0]	Vd.4S -> result	A64
uint64x2_t vmlal_high_n_u32(uint64x2_t a, uint32x4_t b, uint32_t c)	a -> Vd.2D b -> Vn.4S c -> Vm.S[0]	UMLAL2 Vd.2D,Vn.4S,Vm.S[0]	Vd.2D -> result	A64
int16x4_t vmls_n_s16(int16x4_t a, int16x4_t b, int16_t c)	a -> Vd.4H b -> Vn.4H c -> Vm.H[0]	MLS Vd.4H,Vn.4H,Vm.H[0]	Vd.4H -> result	v7/A32/A64
int16x8_t vmlsq_n_s16(int16x8_t a, int16x8_t b, int16_t c)	a -> Vd.8H b -> Vn.8H c -> Vm.H[0]	MLS Vd.8H,Vn.8H,Vm.H[0]	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vmls_n_s32(int32x2_t a, int32x2_t b, int32_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.S[0]</code>	<code>MLS Vd.2S, Vn.2S, Vm.S[0]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vmlsq_n_s32(int32x4_t a, int32x4_t b, int32_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.S[0]</code>	<code>MLS Vd.4S, Vn.4S, Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint16x4_t vmls_n_u16(uint16x4_t a, uint16x4_t b, uint16_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.H[0]</code>	<code>MLS Vd.4H, Vn.4H, Vm.H[0]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vmlsq_n_u16(uint16x8_t a, uint16x8_t b, uint16_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.H[0]</code>	<code>MLS Vd.8H, Vn.8H, Vm.H[0]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vmls_n_u32(uint32x2_t a, uint32x2_t b, uint32_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.S[0]</code>	<code>MLS Vd.2S, Vn.2S, Vm.S[0]</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x4_t vmlsq_n_u32(uint32x4_t a, uint32x4_t b, uint32_t c)	a -> Vd.4S b -> Vn.4S c -> Vm.S[0]	MLS Vd.4S,Vn.4S,Vm.S[0]	Vd.4S -> result	v7/A32/A64
float32x2_t vmls_n_f32(float32x2_t a, float32x2_t b, float32_t c)	N/A	RESULT[I] = a[i] - (b[i] * c) for i = 0 to 1	N/A	v7/A32/A64
float32x4_t vmlsq_n_f32(float32x4_t a, float32x4_t b, float32_t c)	N/A	RESULT[I] = a[i] - (b[i] * c) for i = 0 to 3	N/A	v7/A32/A64
int32x4_t vmlsl_n_s16(int32x4_t a, int16x4_t b, int16_t c)	a -> Vd.4S b -> Vn.4H c -> Vm.H[0]	SMLS Vd.4S,Vn.4H,Vm.H[0]	Vd.4S -> result	v7/A32/A64
int64x2_t vmlsl_n_s32(int64x2_t a, int32x2_t b, int32_t c)	a -> Vd.2D b -> Vn.2S c -> Vm.S[0]	SMLS Vd.2D,Vn.2S,Vm.S[0]	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vmlsl_n_u16(uint32x4_t a, uint16x4_t b, uint16_t c)</code>	<code>a -> Vd.4S b -> Vn.4H c -> Vm.H[0]</code>	<code>UMLSL Vd.4S,Vn.4H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	<code>v7/A32/A64</code>
<code>uint64x2_t vmlsl_n_u32(uint64x2_t a, uint32x2_t b, uint32_t c)</code>	<code>a -> Vd.2D b -> Vn.2S c -> Vm.S[0]</code>	<code>UMLSL Vd.2D,Vn.2S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	<code>v7/A32/A64</code>
<code>int32x4_t vmlsl_high_n_s16(int32x4_t a, int16x8_t b, int16_t c)</code>	<code>a -> Vd.4S b -> Vn.8H c -> Vm.H[0]</code>	<code>SMLSL2 Vd.4S,Vn.8H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	<code>A64</code>
<code>int64x2_t vmlsl_high_n_s32(int64x2_t a, int32x4_t b, int32_t c)</code>	<code>a -> Vd.2D b -> Vn.4S c -> Vm.S[0]</code>	<code>SMLSL2 Vd.2D,Vn.4S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	<code>A64</code>
<code>uint32x4_t vmlsl_high_n_u16(uint32x4_t a, uint16x8_t b, uint16_t c)</code>	<code>a -> Vd.4S b -> Vn.8H c -> Vm.H[0]</code>	<code>UMLSL2 Vd.4S,Vn.8H,Vm.H[0]</code>	<code>Vd.4S -> result</code>	<code>A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vmlsl_high_n_u32(uint64x2_t a, uint32x4_t b, uint32_t c)</code>	<code>a -> Vd.2D b -> Vn.4S c -> Vm.S[0]</code>	<code>UMLSL2 Vd.2D,Vn.4S,Vm.S[0]</code>	<code>Vd.2D -> result</code>	A64

2.1.6.6 Fused multiply-accumulate by scalar

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vfma_n_f32(float32x2_t a, float32x2_t b, float32_t n)</code>	<code>n -> Vm.S[0] b -> Vn.2S a -> Vd.2S</code>	<code>FMLA Vd.2S,Vn.2S,Vm.S[0]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vfmaq_n_f32(float32x4_t a, float32x4_t b, float32_t n)</code>	<code>n -> Vm.S[0] b -> Vn.4S a -> Vd.4S</code>	<code>FMLA Vd.4S,Vn.4S,Vm.S[0]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x2_t vfms_n_f32(float32x2_t a, float32x2_t b, float32_t n)</code>	<code>n -> Vm.S[0] b -> Vn.2S a -> Vd.2S</code>	<code>FMLS Vd.2S,Vn.2S,Vm.S[0]</code>	<code>Vd.2S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vfmsq_n_f32(float32x4_t a, float32x4_t b, float32_t n)	n -> Vm.S[0] b -> Vn.4S a -> Vd.4S	FMLS Vd.4S,Vn.4S,Vm.S[0]	Vd.4S -> result	A64
float64x1_t vfma_n_f64(float64x1_t a, float64x1_t b, float64_t n)	b -> Dn n -> Dm a -> Da	FMADD Dd,Dn,Dm,Da	Dd -> result	A64
float64x2_t vfmaq_n_f64(float64x2_t a, float64x2_t b, float64_t n)	n -> Vm.D[0] b -> Vn.2D a -> Vd.2D	FMLA Vd.2D,Vn.2D,Vm.D[0]	Vd.2D -> result	A64
float64x1_t vfms_n_f64(float64x1_t a, float64x1_t b, float64_t n)	b -> Dn n -> Dm a -> Da	FMSUB Dd,Dn,Dm,Da	Dd -> result	A64
float64x2_t vfmsq_n_f64(float64x2_t a, float64x2_t b, float64_t n)	n -> Vm.D[0] b -> Vn.2D a -> Vd.2D	FMLS Vd.2D,Vn.2D,Vm.D[0]	Vd.2D -> result	A64

2.1.7 Logical

2.1.7.1 Negate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vneg_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>NEG Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vnegq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>NEG Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vneg_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>NEG Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vnegq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>NEG Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vneg_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>NEG Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vnegq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>NEG Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>float32x2_t vneg_f32(float32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>FNEG Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vnegq_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>FNEG Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vneg_s64(int64x1_t a)</code>	<code>a -> Dn</code>	<code>NEG Dd,Dn</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vnegd_s64(int64_t a)</code>	<code>a -> Dn</code>	NEG Dd,Dn	<code>Dd -> result</code>	A64
<code>int64x2_t vnegq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	NEG Vd.2D,Vn.2D	<code>Vd.2D -> result</code>	A64
<code>float64x1_t vneg_f64(float64x1_t a)</code>	<code>a -> Dn</code>	FNEG Dd,Dn	<code>Dd -> result</code>	A64
<code>float64x2_t vnegq_f64(float64x2_t a)</code>	<code>a -> Vn.2D</code>	FNEG Vd.2D,Vn.2D	<code>Vd.2D -> result</code>	A64

2.1.7.2 Saturating Negate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vqneg_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	SQNEG Vd.8B,Vn.8B	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vqnegq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	SQNEG Vd.16B,Vn.16B	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vqneg_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	SQNEG Vd.4H,Vn.4H	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vqnegq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SQNEG Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vqneg_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>SQNEG Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vqnegq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>SQNEG Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vqneg_s64(int64x1_t a)</code>	<code>a -> Dn</code>	<code>SQNEG Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>int64x2_t vqnegq_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>SQNEG Vd.2D,Vn.2D</code>	<code>Vd.2D -> result</code>	A64
<code>int8_t vqnegb_s8(int8_t a)</code>	<code>a -> Bn</code>	<code>SQNEG Bd,Bn</code>	<code>Bd -> result</code>	A64
<code>int16_t vqnegh_s16(int16_t a)</code>	<code>a -> Hn</code>	<code>SQNEG Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int32_t vqnegs_s32(int32_t a)</code>	<code>a -> Sn</code>	<code>SQNEG Sd,Sn</code>	<code>Sd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vqnegd_s64(int64_t a)</code>	<code>a -> Dn</code>	<code>SQNEG Dd,Dn</code>	<code>Dd -> result</code>	A64

2.1.7.3 Bitwise NOT

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vmvn_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>MVN Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vmvnq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>MVN Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vmvn_s16(int16x4_t a)</code>	<code>a -> Vn.8B</code>	<code>MVN Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x8_t vmvnq_s16(int16x8_t a)</code>	<code>a -> Vn.16B</code>	<code>MVN Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int32x2_t vmvn_s32(int32x2_t a)</code>	<code>a -> Vn.8B</code>	<code>MVN Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int32x4_t vmvnq_s32(int32x4_t a)</code>	<code>a -> Vn.16B</code>	<code>MVN Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vmvn_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>MVN Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vmvnq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>MVN Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vmvn_u16(uint16x4_t a)</code>	<code>a -> Vn.8B</code>	<code>MVN Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x8_t vmvnq_u16(uint16x8_t a)</code>	<code>a -> Vn.16B</code>	<code>MVN Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint32x2_t vmvn_u32(uint32x2_t a)</code>	<code>a -> Vn.8B</code>	<code>MVN Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint32x4_t vmvnq_u32(uint32x4_t a)</code>	<code>a -> Vn.16B</code>	<code>MVN Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly8x8_t vmvn_p8(poly8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>MVN Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x16_t vmvnq_p8(poly8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>MVN Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

2.1.7.4 AND

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vand_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>AND Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vandq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>AND Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vand_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>AND Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x8_t vandq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>AND Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int32x2_t vand_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>AND Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int32x4_t vandq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>AND Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int64x1_t vand_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Dn b -> Dm</code>	<code>AND Dd,Dn,Dm</code>	<code>Dd -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vandq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>AND Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint8x8_t vand_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>AND Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vandq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>AND Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vand_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>AND Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x8_t vandq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>AND Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint32x2_t vand_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>AND Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint32x4_t vandq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>AND Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vand_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>AND Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint64x2_t vandq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>AND Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

2.1.7.5 OR

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vorr_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vorrq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vorr_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vorrq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int32x2_t vorrq_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int32x4_t vorrq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int64x1_t vorrq_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int64x2_t vorrq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint8x8_t vorrq_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vorrq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vorr_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x8_t vorrq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint32x2_t vorr_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint32x4_t vorrq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint64x1_t vorr_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint64x2_t vorrq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

2.1.7.6 Exclusive OR

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t veor_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>EOR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t veorq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>EOR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t veor_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>EOR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x8_t veorq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>EOR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int32x2_t veor_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>EOR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int32x4_t veorq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>EOR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int64x1_t veor_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>EOR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t veorq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>EOR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint8x8_t veor_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>EOR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t veorq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>EOR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t veor_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>EOR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x8_t veorq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>EOR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint32x2_t veor_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>EOR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint32x4_t veorq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>EOR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t veor_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>EOR Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint64x2_t veorq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>EOR Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

2.1.7.7 OR-NOT

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vorn_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORN Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vornq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ORN Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vorn_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ORN Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vornq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ORN Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>int32x2_t vorn_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ORN Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>int32x4_t vornq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ORN Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>int64x1_t vorn_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ORN Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>int64x2_t vornq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ORN Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>uint8x8_t vorn_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ORN Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>uint8x16_t vornq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ORN Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vorn_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ORN Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>uint16x8_t vorrq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ORN Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>uint32x2_t vorn_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ORN Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>uint32x4_t vorrq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ORN Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
<code>uint64x1_t vorn_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ORN Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
<code>uint64x2_t vorrq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ORN Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64

2.1.8 Bit manipulation

2.1.8.1 Count leading sign bits

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vcls_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CLS Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vclsq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CLS Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vcls_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>CLS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vclsq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>CLS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vcls_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>CLS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vclsq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>CLS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int8x8_t vcls_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CLS Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vclsq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CLS Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vcls_u16(uint16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>CLS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vclsq_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>CLS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vcls_u32(uint32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>CLS Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vclsq_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>CLS Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

2.1.8.2 Count leading zeros

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vclz_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CLZ Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vclzq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CLZ Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vclz_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>CLZ Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vclzq_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>CLZ Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vclz_s32(int32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>CLZ Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vclzq_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>CLZ Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint8x8_t vclz_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CLZ Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vclzq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CLZ Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vclz_u16(uint16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>CLZ Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vclzq_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>CLZ Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vclz_u32(uint32x2_t a)</code>	<code>a -> Vn.2S</code>	<code>CLZ Vd.2S,Vn.2S</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vclzq_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>CLZ Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	v7/A32/A64

2.1.8.3 Population count

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vcnt_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CNT Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vcntq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CNT Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint8x8_t vcnt_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CNT Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vcntq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CNT Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly8x8_t vcnt_p8(poly8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>CNT Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x16_t vcntq_p8(poly8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>CNT Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

2.1.8.4 Bitwise clear

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vbic_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>BIC Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vbicq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>BIC Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vbic_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>BIC Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x8_t vbicq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>BIC Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int32x2_t vbic_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>BIC Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int32x4_t vbicq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>BIC Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int64x1_t vbic_s64(int64x1_t a, int64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>BIC Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vbicq_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>BIC Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint8x8_t vbic_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>BIC Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vbicq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>BIC Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vbic_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>BIC Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x8_t vbicq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>BIC Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint32x2_t vbic_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>BIC Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint32x4_t vbicq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>BIC Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vbic_u64(uint64x1_t a, uint64x1_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>BIC Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint64x2_t vbicq_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>BIC Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

2.1.8.5 Bitwise select

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vbsl_s8(uint8x8_t a, int8x8_t b, int8x8_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vbslq_s8(uint8x16_t a, int8x16_t b, int8x16_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vbsl_s16(uint16x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vbslq_s16(uint16x8_t a, int16x8_t b, int16x8_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int32x2_t vbsl_s32(uint32x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int32x4_t vbslq_s32(uint32x4_t a, int32x4_t b, int32x4_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int64x1_t vbsl_s64(uint64x1_t a, int64x1_t b, int64x1_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int64x2_t vbslq_s64(uint64x2_t a, int64x2_t b, int64x2_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vbsl_u8(uint8x8_t a, uint8x8_t b, uint8x8_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vbslq_u8(uint8x16_t a, uint8x16_t b, uint8x16_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vbsl_u16(uint16x4_t a, uint16x4_t b, uint16x4_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x8_t vbslq_u16(uint16x8_t a, uint16x8_t b, uint16x8_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint32x2_t vbsl_u32(uint32x2_t a, uint32x2_t b, uint32x2_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vbslq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint64x1_t vbsl_u64(uint64x1_t a, uint64x1_t b, uint64x1_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint64x2_t vbslq_u64(uint64x2_t a, uint64x2_t b, uint64x2_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly64x1_t vbsl_p64(poly64x1_t a, poly64x1_t b, poly64x1_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A32/A64
<code>poly64x2_t vbslq_p64(poly64x2_t a, poly64x2_t b, poly64x2_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vbsl_f32(uint32x2_t a, float32x2_t b, float32x2_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>float32x4_t vbslq_f32(uint32x4_t a, float32x4_t b, float32x4_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly8x8_t vbsl_p8(uint8x8_t a, poly8x8_t b, poly8x8_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x16_t vbslq_p8(uint8x16_t a, poly8x16_t b, poly8x16_t c)</code>	<code>a -> Vd.16B b -> Vn.16B c -> Vm.16B</code>	<code>BSL Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x4_t vbsl_p16(uint16x4_t a, poly16x4_t b, poly16x4_t c)</code>	<code>a -> Vd.8B b -> Vn.8B c -> Vm.8B</code>	<code>BSL Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly16x8_t vbslq_p16(uint16x8_t a, poly16x8_t b, poly16x8_t c)	a -> Vd.16B b -> Vn.16B c -> Vm.16B	BSL Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
float64x1_t vbsl_f64(uint64x1_t a, float64x1_t b, float64x1_t c)	a -> Vd.8B b -> Vn.8B c -> Vm.8B	BSL Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
float64x2_t vbslq_f64(uint64x2_t a, float64x2_t b, float64x2_t c)	a -> Vd.16B b -> Vn.16B c -> Vm.16B	BSL Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
mfloat8x8_t vbsl_mf8(uint8x8_t a, mfloat8x8_t b, mfloat8x8_t c)	a -> Vd.8B b -> Vn.8B c -> Vm.8B	BSL Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
mfloat8x16_t vbslq_mf8(uint8x16_t a, mfloat8x16_t b, mfloat8x16_t c)	a -> Vd.16B b -> Vn.16B c -> Vm.16B	BSL Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64

2.1.9 Vector manipulation

2.1.9.1 Copy vector lane

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vcopy_lane_s8(int8x8_t a, const int lane1, int8x8_t b, const int lane2)</code>	<code>a -> Vd.8B 0 <= lane1 <= 7 b -> Vn.8B 0 <= lane2 <= 7</code>	INS Vd.B[lane1],Vn.B[lane2]	Vd.8B -> result	A64
<code>int8x16_t vcopyq_lane_s8(int8x16_t a, const int lane1, int8x8_t b, const int lane2)</code>	<code>a -> Vd.16B 0 <= lane1 <= 15 b -> Vn.8B 0 <= lane2 <= 7</code>	INS Vd.B[lane1],Vn.B[lane2]	Vd.16B -> result	A64
<code>int16x4_t vcopy_lane_s16(int16x4_t a, const int lane1, int16x4_t b, const int lane2)</code>	<code>a -> Vd.4H 0 <= lane1 <= 3 b -> Vn.4H 0 <= lane2 <= 3</code>	INS Vd.H[lane1],Vn.H[lane2]	Vd.4H -> result	A64
<code>int16x8_t vcopyq_lane_s16(int16x8_t a, const int lane1, int16x4_t b, const int lane2)</code>	<code>a -> Vd.8H 0 <= lane1 <= 7 b -> Vn.4H 0 <= lane2 <= 3</code>	INS Vd.H[lane1],Vn.H[lane2]	Vd.8H -> result	A64
<code>int32x2_t vcopy_lane_s32(int32x2_t a, const int lane1, int32x2_t b, const int lane2)</code>	<code>a -> Vd.2S 0 <= lane1 <= 1 b -> Vn.2S 0 <= lane2 <= 1</code>	INS Vd.S[lane1],Vn.S[lane2]	Vd.2S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>int32x4_t vcopyq_lane_s32(int32x4_t a, const int lane1, int32x2_t b, const int lane2)</pre>	<pre>a -> Vd.4S 0 <= lane1 <= 3 b -> Vn.2S 0 <= lane2 <= 1</pre>	INS Vd.S[lane1],Vn.S[lane2]	Vd.4S -> result	A64
<pre>int64x1_t vcopy_lane_s64(int64x1_t a, const int lane1, int64x1_t b, const int lane2)</pre>	<pre>a -> UNUSED 0 <= lane1 <= 0 b -> Vn.1D 0 <= lane2 <= 0</pre>	DUP Dd,Vn.D[lane2]	Dd -> result	A64
<pre>int64x2_t vcopyq_lane_s64(int64x2_t a, const int lane1, int64x1_t b, const int lane2)</pre>	<pre>a -> Vd.2D 0 <= lane1 <= 1 b -> Vn.1D 0 <= lane2 <= 0</pre>	INS Vd.D[lane1],Vn.D[lane2]	Vd.2D -> result	A64
<pre>uint8x8_t vcopy_lane_u8(uint8x8_t a, const int lane1, uint8x8_t b, const int lane2)</pre>	<pre>a -> Vd.8B 0 <= lane1 <= 7 b -> Vn.8B 0 <= lane2 <= 7</pre>	INS Vd.B[lane1],Vn.B[lane2]	Vd.8B -> result	A64
<pre>uint8x16_t vcopyq_lane_u8(uint8x16_t a, const int lane1, uint8x8_t b, const int lane2)</pre>	<pre>a -> Vd.16B 0 <= lane1 <= 15 b -> Vn.8B 0 <= lane2 <= 7</pre>	INS Vd.B[lane1],Vn.B[lane2]	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x4_t vcopy_lane_u16(uint16x4_t a, const int lane1, uint16x4_t b, const int lane2)	a -> Vd.4H 0 <= lane1 <= 3 b -> Vn.4H 0 <= lane2 <= 3	INS Vd.H[lane1],Vn.H[lane2]	Vd.4H -> result	A64
uint16x8_t vcopyq_lane_u16(uint16x8_t a, const int lane1, uint16x4_t b, const int lane2)	a -> Vd.8H 0 <= lane1 <= 7 b -> Vn.4H 0 <= lane2 <= 3	INS Vd.H[lane1],Vn.H[lane2]	Vd.8H -> result	A64
uint32x2_t vcopy_lane_u32(uint32x2_t a, const int lane1, uint32x2_t b, const int lane2)	a -> Vd.2S 0 <= lane1 <= 1 b -> Vn.2S 0 <= lane2 <= 1	INS Vd.S[lane1],Vn.S[lane2]	Vd.2S -> result	A64
uint32x4_t vcopyq_lane_u32(uint32x4_t a, const int lane1, uint32x2_t b, const int lane2)	a -> Vd.4S 0 <= lane1 <= 3 b -> Vn.2S 0 <= lane2 <= 1	INS Vd.S[lane1],Vn.S[lane2]	Vd.4S -> result	A64
uint64x1_t vcopy_lane_u64(uint64x1_t a, const int lane1, uint64x1_t b, const int lane2)	a -> UNUSED 0 <= lane1 <= 0 b -> Vn.1D 0 <= lane2 <= 0	DUP Dd,Vn.D[lane2]	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint64x2_t vcopyq_lane_u64(uint64x2_t a, const int lane1, uint64x1_t b, const int lane2)	a -> Vd.2D 0 <= lane1 <= 1 b -> Vn.1D 0 <= lane2 <= 0	INS Vd.D[lane1],Vn.D[lane2]	Vd.2D -> result	A64
poly64x1_t vcopy_lane_p64(poly64x1_t a, const int lane1, poly64x1_t b, const int lane2)	a -> UNUSED 0 <= lane1 <= 0 b -> Vn.1D 0 <= lane2 <= 0	DUP Dd,Vn.D[lane2]	Dd -> result	A32/A64
poly64x2_t vcopyq_lane_p64(poly64x2_t a, const int lane1, poly64x1_t b, const int lane2)	a -> Vd.2D 0 <= lane1 <= 1 b -> Vn.1D 0 <= lane2 <= 0	INS Vd.D[lane1],Vn.D[lane2]	Vd.2D -> result	A32/A64
float32x2_t vcopy_lane_f32(float32x2_t a, const int lane1, float32x2_t b, const int lane2)	a -> Vd.2S 0 <= lane1 <= 1 b -> Vn.2S 0 <= lane2 <= 1	INS Vd.S[lane1],Vn.S[lane2]	Vd.2S -> result	A64
float32x4_t vcopyq_lane_f32(float32x4_t a, const int lane1, float32x2_t b, const int lane2)	a -> Vd.4S 0 <= lane1 <= 3 b -> Vn.2S 0 <= lane2 <= 1	INS Vd.S[lane1],Vn.S[lane2]	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x1_t vcopy_lane_f64(float64x1_t a, const int lane1, float64x1_t b, const int lane2)	a -> UNUSED 0 <= lane1 <= 0 b -> Vn.1D 0 <= lane2 <= 0	DUP Dd,Vn.D[lane2]	Dd -> result	A64
float64x2_t vcopyq_lane_f64(float64x2_t a, const int lane1, float64x1_t b, const int lane2)	a -> Vd.2D 0 <= lane1 <= 1 b -> Vn.1D 0 <= lane2 <= 0	INS Vd.D[lane1],Vn.D[lane2]	Vd.2D -> result	A64
poly8x8_t vcopy_lane_p8(poly8x8_t a, const int lane1, poly8x8_t b, const int lane2)	a -> Vd.8B 0 <= lane1 <= 7 b -> Vn.8B 0 <= lane2 <= 7	INS Vd.B[lane1],Vn.B[lane2]	Vd.8B -> result	A64
poly8x16_t vcopyq_lane_p8(poly8x16_t a, const int lane1, poly8x8_t b, const int lane2)	a -> Vd.16B 0 <= lane1 <= 15 b -> Vn.8B 0 <= lane2 <= 7	INS Vd.B[lane1],Vn.B[lane2]	Vd.16B -> result	A64
poly16x4_t vcopy_lane_p16(poly16x4_t a, const int lane1, poly16x4_t b, const int lane2)	a -> Vd.4H 0 <= lane1 <= 3 b -> Vn.4H 0 <= lane2 <= 3	INS Vd.H[lane1],Vn.H[lane2]	Vd.4H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly16x8_t vcopyq_lane_p16(poly16x8_t a, const int lane1, poly16x4_t b, const int lane2)	a -> Vd.8H 0 <= lane1 <= 7 b -> Vn.4H 0 <= lane2 <= 3	INS Vd.H[lane1],Vn.H[lane2]	Vd.8H -> result	A64
mfloat8x8_t vcopy_lane_mf8(mfloat8x8_t a, const int lane1, mfloat8x8_t b, const int lane2)	a -> Vd.8B 0 <= lane1 <= 7 b -> Vn.8B 0 <= lane2 <= 7	INS Vd.B[lane1],Vn.B[lane2]	Vd.8B -> result	A64
mfloat8x16_t vcopyq_lane_mf8(mfloat8x16_t a, const int lane1, mfloat8x8_t b, const int lane2)	a -> Vd.16B 0 <= lane1 <= 15 b -> Vn.8B 0 <= lane2 <= 7	INS Vd.B[lane1],Vn.B[lane2]	Vd.16B -> result	A64
int8x8_t vcopy_laneq_s8(int8x8_t a, const int lane1, int8x16_t b, const int lane2)	a -> Vd.8B 0 <= lane1 <= 7 b -> Vn.16B 0 <= lane2 <= 15	INS Vd.B[lane1],Vn.B[lane2]	Vd.8B -> result	A64
int8x16_t vcopyq_laneq_s8(int8x16_t a, const int lane1, int8x16_t b, const int lane2)	a -> Vd.16B 0 <= lane1 <= 15 b -> Vn.16B 0 <= lane2 <= 15	INS Vd.B[lane1],Vn.B[lane2]	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vcopy_laneq_s16(int16x4_t a, const int lane1, int16x8_t b, const int lane2)</code>	<code>a -> Vd.4H 0 <= lane1 <= 3 b -> Vn.8H 0 <= lane2 <= 7</code>	INS Vd.H[lane1],Vn.H[lane2]	Vd.4H -> result	A64
<code>int16x8_t vcopyq_laneq_s16(int16x8_t a, const int lane1, int16x8_t b, const int lane2)</code>	<code>a -> Vd.8H 0 <= lane1 <= 7 b -> Vn.8H 0 <= lane2 <= 7</code>	INS Vd.H[lane1],Vn.H[lane2]	Vd.8H -> result	A64
<code>int32x2_t vcopy_laneq_s32(int32x2_t a, const int lane1, int32x4_t b, const int lane2)</code>	<code>a -> Vd.2S 0 <= lane1 <= 1 b -> Vn.4S 0 <= lane2 <= 3</code>	INS Vd.S[lane1],Vn.S[lane2]	Vd.2S -> result	A64
<code>int32x4_t vcopyq_laneq_s32(int32x4_t a, const int lane1, int32x4_t b, const int lane2)</code>	<code>a -> Vd.4S 0 <= lane1 <= 3 b -> Vn.4S 0 <= lane2 <= 3</code>	INS Vd.S[lane1],Vn.S[lane2]	Vd.4S -> result	A64
<code>int64x1_t vcopy_laneq_s64(int64x1_t a, const int lane1, int64x2_t b, const int lane2)</code>	<code>a -> UNUSED 0 <= lane1 <= 0 b -> Vn.2D 0 <= lane2 <= 1</code>	DUP Dd,Vn.D[lane2]	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>int64x2_t vcopyq_laneq_s64(int64x2_t a, const int lane1, int64x2_t b, const int lane2)</pre>	<pre>a -> Vd.2D 0 <= lane1 <= 1 b -> Vn.2D 0 <= lane2 <= 1</pre>	INS Vd.D[lane1],Vn.D[lane2]	Vd.2D -> result	A64
<pre>uint8x8_t vcopy_laneq_u8(uint8x8_t a, const int lane1, uint8x16_t b, const int lane2)</pre>	<pre>a -> Vd.8B 0 <= lane1 <= 7 b -> Vn.16B 0 <= lane2 <= 15</pre>	INS Vd.B[lane1],Vn.B[lane2]	Vd.8B -> result	A64
<pre>uint8x16_t vcopyq_laneq_u8(uint8x16_t a, const int lane1, uint8x16_t b, const int lane2)</pre>	<pre>a -> Vd.16B 0 <= lane1 <= 15 b -> Vn.16B 0 <= lane2 <= 15</pre>	INS Vd.B[lane1],Vn.B[lane2]	Vd.16B -> result	A64
<pre>uint16x4_t vcopy_laneq_u16(uint16x4_t a, const int lane1, uint16x8_t b, const int lane2)</pre>	<pre>a -> Vd.4H 0 <= lane1 <= 3 b -> Vn.8H 0 <= lane2 <= 7</pre>	INS Vd.H[lane1],Vn.H[lane2]	Vd.4H -> result	A64
<pre>uint16x8_t vcopyq_laneq_u16(uint16x8_t a, const int lane1, uint16x8_t b, const int lane2)</pre>	<pre>a -> Vd.8H 0 <= lane1 <= 7 b -> Vn.8H 0 <= lane2 <= 7</pre>	INS Vd.H[lane1],Vn.H[lane2]	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint32x2_t vcopy_laneq_u32(uint32x2_t a, const int lane1, uint32x4_t b, const int lane2)	a -> Vd.2S 0 <= lane1 <= 1 b -> Vn.4S 0 <= lane2 <= 3	INS Vd.S[lane1],Vn.S[lane2]	Vd.2S -> result	A64
uint32x4_t vcopyq_laneq_u32(uint32x4_t a, const int lane1, uint32x4_t b, const int lane2)	a -> Vd.4S 0 <= lane1 <= 3 b -> Vn.4S 0 <= lane2 <= 3	INS Vd.S[lane1],Vn.S[lane2]	Vd.4S -> result	A64
uint64x1_t vcopy_laneq_u64(uint64x1_t a, const int lane1, uint64x2_t b, const int lane2)	a -> UNUSED 0 <= lane1 <= 0 b -> Vn.2D 0 <= lane2 <= 1	DUP Dd,Vn.D[lane2]	Dd -> result	A64
uint64x2_t vcopyq_laneq_u64(uint64x2_t a, const int lane1, uint64x2_t b, const int lane2)	a -> Vd.2D 0 <= lane1 <= 1 b -> Vn.2D 0 <= lane2 <= 1	INS Vd.D[lane1],Vn.D[lane2]	Vd.2D -> result	A64
poly64x1_t vcopy_laneq_p64(poly64x1_t a, const int lane1, poly64x2_t b, const int lane2)	a -> UNUSED 0 <= lane1 <= 0 b -> Vn.2D 0 <= lane2 <= 1	DUP Dd,Vn.D[lane2]	Dd -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x2_t vcopyq_laneq_p64(poly64x2_t a, const int lane1, poly64x2_t b, const int lane2)	a -> Vd.2D 0 <= lane1 <= 1 b -> Vn.2D 0 <= lane2 <= 1	INS Vd.D[lane1],Vn.D[lane2]	Vd.2D -> result	A32/A64
float32x2_t vcopy_laneq_f32(float32x2_t a, const int lane1, float32x4_t b, const int lane2)	a -> Vd.2S 0 <= lane1 <= 1 b -> Vn.4S 0 <= lane2 <= 3	INS Vd.S[lane1],Vn.S[lane2]	Vd.2S -> result	A64
float32x4_t vcopyq_laneq_f32(float32x4_t a, const int lane1, float32x4_t b, const int lane2)	a -> Vd.4S 0 <= lane1 <= 3 b -> Vn.4S 0 <= lane2 <= 3	INS Vd.S[lane1],Vn.S[lane2]	Vd.4S -> result	A64
float64x1_t vcopy_laneq_f64(float64x1_t a, const int lane1, float64x2_t b, const int lane2)	a -> UNUSED 0 <= lane1 <= 0 b -> Vn.2D 0 <= lane2 <= 1	DUP Dd,Vn.D[lane2]	Dd -> result	A64
float64x2_t vcopyq_laneq_f64(float64x2_t a, const int lane1, float64x2_t b, const int lane2)	a -> Vd.2D 0 <= lane1 <= 1 b -> Vn.2D 0 <= lane2 <= 1	INS Vd.D[lane1],Vn.D[lane2]	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x8_t vcopy_laneq_p8(poly8x8_t a, const int lane1, poly8x16_t b, const int lane2)	a -> Vd.8B 0 <= lane1 <= 7 b -> Vn.16B 0 <= lane2 <= 15	INS Vd.B[lane1],Vn.B[lane2]	Vd.8B -> result	A64
poly8x16_t vcopyq_laneq_p8(poly8x16_t a, const int lane1, poly8x16_t b, const int lane2)	a -> Vd.16B 0 <= lane1 <= 15 b -> Vn.16B 0 <= lane2 <= 15	INS Vd.B[lane1],Vn.B[lane2]	Vd.16B -> result	A64
poly16x4_t vcopy_laneq_p16(poly16x4_t a, const int lane1, poly16x8_t b, const int lane2)	a -> Vd.4H 0 <= lane1 <= 3 b -> Vn.8H 0 <= lane2 <= 7	INS Vd.H[lane1],Vn.H[lane2]	Vd.4H -> result	A64
poly16x8_t vcopyq_laneq_p16(poly16x8_t a, const int lane1, poly16x8_t b, const int lane2)	a -> Vd.8H 0 <= lane1 <= 7 b -> Vn.8H 0 <= lane2 <= 7	INS Vd.H[lane1],Vn.H[lane2]	Vd.8H -> result	A64
mfloat8x8_t vcopy_laneq_mf8(mfloat8x8_t a, const int lane1, mfloat8x16_t b, const int lane2)	a -> Vd.8B 0 <= lane1 <= 7 b -> Vn.16B 0 <= lane2 <= 15	INS Vd.B[lane1],Vn.B[lane2]	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x16_t vcopyq_laneq_mf8(mfloat8x16_t a, const int lane1, mfloat8x16_t b, const int lane2)</code>	<code>a -> Vd.16B 0 <= lane1 <= 15 b -> Vn.16B 0 <= lane2 <= 15</code>	<code>INS Vd.B[lane1], Vn.B[lane2]</code>	<code>Vd.16B -> result</code>	A64

2.1.9.2 Reverse bits within elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vrbit_s8(int8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>RBIT Vd.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x16_t vrbitq_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>RBIT Vd.16B, Vn.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint8x8_t vrbit_u8(uint8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>RBIT Vd.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vrbitq_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>RBIT Vd.16B, Vn.16B</code>	<code>Vd.16B -> result</code>	A64
<code>poly8x8_t vrbit_p8(poly8x8_t a)</code>	<code>a -> Vn.8B</code>	<code>RBIT Vd.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x16_t vrbitq_p8(poly8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>RBIT Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	A64

2.1.9.3 Create vector

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vcreate_s8(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x4_t vcreate_s16(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int32x2_t vcreate_s32(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int64x1_t vcreate_s64(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.1D -> result</code>	v7/A32/A64
<code>uint8x8_t vcreate_u8(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint16x4_t vcreate_u16(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vcreate_u32(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>uint64x1_t vcreate_u64(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.1D -> result</code>	<code>v7/A32/A64</code>
<code>poly64x1_t vcreate_p64(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.1D -> result</code>	<code>A32/A64</code>
<code>float16x4_t vcreate_f16(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>float32x2_t vcreate_f32(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>poly8x8_t vcreate_p8(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>poly16x4_t vcreate_p16(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>float64x1_t vcreate_f64(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.1D -> result</code>	<code>A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x8_t vcreate_mf8(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.8B -> result</code>	A64

2.1.9.4 Set all lanes to the same value

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vdup_n_s8(int8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8B,rn</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vdupq_n_s8(int8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.16B,rn</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vdup_n_s16(int16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4H,rn</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vdupq_n_s16(int16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8H,rn</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vdup_n_s32(int32_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.2S,rn</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vdupq_n_s32(int32_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4S,rn</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1_t vdup_n_s64(int64_t value)</code>	<code>value -> rn</code>	<code>INS Dd.D[0],xn</code>	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x2_t vdupq_n_s64(int64_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.2D,rn</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vdup_n_u8(uint8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8B,rn</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vdupq_n_u8(uint8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.16B,rn</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vdup_n_u16(uint16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4H,rn</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vdupq_n_u16(uint16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8H,rn</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vdup_n_u32(uint32_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.2S,rn</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vdupq_n_u32(uint32_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4S,rn</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vdup_n_u64(uint64_t value)</code>	<code>value -> rn</code>	<code>INS Dd.D[0],xn</code>	<code>Vd.1D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint64x2_t vdupq_n_u64(uint64_t value)	value -> rn	DUP Vd.2D,rn	Vd.2D -> result	v7/A32/A64
poly64x1_t vdup_n_p64(poly64_t value)	value -> rn	INS Dd.D[0],xn	Vd.1D -> result	A32/A64
poly64x2_t vdupq_n_p64(poly64_t value)	value -> rn	DUP Vd.2D,rn	Vd.2D -> result	A32/A64
float32x2_t vdup_n_f32(float32_t value)	value -> rn	DUP Vd.2S,rn	Vd.2S -> result	v7/A32/A64
float32x4_t vdupq_n_f32(float32_t value)	value -> rn	DUP Vd.4S,rn	Vd.4S -> result	v7/A32/A64
poly8x8_t vdup_n_p8(poly8_t value)	value -> rn	DUP Vd.8B,rn	Vd.8B -> result	v7/A32/A64
poly8x16_t vdupq_n_p8(poly8_t value)	value -> rn	DUP Vd.16B,rn	Vd.16B -> result	v7/A32/A64
poly16x4_t vdup_n_p16(poly16_t value)	value -> rn	DUP Vd.4H,rn	Vd.4H -> result	v7/A32/A64
poly16x8_t vdupq_n_p16(poly16_t value)	value -> rn	DUP Vd.8H,rn	Vd.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x1_t vdup_n_f64(float64_t value)</code>	<code>value -> rn</code>	<code>INS Dd.D[0],xn</code>	<code>Vd.1D -> result</code>	A64
<code>float64x2_t vdupq_n_f64(float64_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.2D,rn</code>	<code>Vd.2D -> result</code>	A64
<code>mfloat8x8_t vdup_n_mf8(mfloat8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8B,rn</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vdupq_n_mf8(mfloat8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.16B,rn</code>	<code>Vd.16B -> result</code>	A64
<code>int8x8_t vmov_n_s8(int8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8B,rn</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vmovq_n_s8(int8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.16B,rn</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vmov_n_s16(int16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4H,rn</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vmovq_n_s16(int16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8H,rn</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>int32x2_t vmov_n_s32(int32_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.2S,rn</code>	<code>Vd.2S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vmovq_n_s32(int32_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4S,rn</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vmov_n_s64(int64_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.1D,rn</code>	<code>Vd.1D -> result</code>	v7/A32/A64
<code>int64x2_t vmovq_n_s64(int64_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.2D,rn</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vmov_n_u8(uint8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8B,rn</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vmovq_n_u8(uint8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.16B,rn</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vmov_n_u16(uint16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4H,rn</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vmovq_n_u16(uint16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8H,rn</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vmov_n_u32(uint32_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.2S,rn</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vmovq_n_u32(uint32_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4S,rn</code>	<code>Vd.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint64x1_t vmov_n_u64(uint64_t value)	value -> rn	DUP Vd.1D,rn	Vd.1D -> result	v7/A32/A64
uint64x2_t vmovq_n_u64(uint64_t value)	value -> rn	DUP Vd.2D,rn	Vd.2D -> result	v7/A32/A64
float32x2_t vmov_n_f32(float32_t value)	value -> rn	DUP Vd.2S,rn	Vd.2S -> result	v7/A32/A64
float32x4_t vmovq_n_f32(float32_t value)	value -> rn	DUP Vd.4S,rn	Vd.4S -> result	v7/A32/A64
poly8x8_t vmov_n_p8(poly8_t value)	value -> rn	DUP Vd.8B,rn	Vd.8B -> result	v7/A32/A64
poly8x16_t vmovq_n_p8(poly8_t value)	value -> rn	DUP Vd.16B,rn	Vd.16B -> result	v7/A32/A64
poly16x4_t vmov_n_p16(poly16_t value)	value -> rn	DUP Vd.4H,rn	Vd.4H -> result	v7/A32/A64
poly16x8_t vmovq_n_p16(poly16_t value)	value -> rn	DUP Vd.8H,rn	Vd.8H -> result	v7/A32/A64
float64x1_t vmov_n_f64(float64_t value)	value -> rn	DUP Vd.1D,rn	Vd.1D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2_t vmovq_n_f64(float64_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.2D,rn</code>	<code>Vd.2D -> result</code>	A64
<code>mfloat8x8_t vmov_n_mf8(mfloat8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8B,rn</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vmovq_n_mf8(mfloat8_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.16B,rn</code>	<code>Vd.16B -> result</code>	A64
<code>int8x8_t vdup_lane_s8(int8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Vd.8B,Vn.B[lane]</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vdupq_lane_s8(int8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Vd.16B,Vn.B[lane]</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vdup_lane_s16(int16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vdupq_lane_s16(int16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vdup_lane_s32(int32x2_t vec, const int lane)</code>	<code>vec -> Vn.2S 0 <= lane <= 1</code>	<code>DUP Vd.2S,Vn.S[lane]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>int32x4_t vdupq_lane_s32(int32x2_t vec, const int lane)</code>	<code>vec -> Vn.2S 0 <= lane <= 1</code>	<code>DUP Vd.4S,Vn.S[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>int64x1_t vdup_lane_s64(int64x1_t vec, const int lane)</code>	<code>vec -> Vn.1D 0 <= lane <= 0</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	v7/A32/A64
<code>int64x2_t vdupq_lane_s64(int64x1_t vec, const int lane)</code>	<code>vec -> Vn.1D 0 <= lane <= 0</code>	<code>DUP Vd.2D,Vn.D[lane]</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vdup_lane_u8(uint8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Vd.8B,Vn.B[lane]</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vdupq_lane_u8(uint8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Vd.16B,Vn.B[lane]</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vdup_lane_u16(uint16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vdupq_lane_u16(uint16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vdup_lane_u32(uint32x2_t vec, const int lane)</code>	<code>vec -> Vn.2S 0 <= lane <= 1</code>	<code>DUP Vd.2S,Vn.S[lane]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vdupq_lane_u32(uint32x2_t vec, const int lane)</code>	<code>vec -> Vn.2S 0 <= lane <= 1</code>	<code>DUP Vd.4S,Vn.S[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vdup_lane_u64(uint64x1_t vec, const int lane)</code>	<code>vec -> Vn.1D 0 <= lane <= 0</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	v7/A32/A64
<code>uint64x2_t vdupq_lane_u64(uint64x1_t vec, const int lane)</code>	<code>vec -> Vn.1D 0 <= lane <= 0</code>	<code>DUP Vd.2D,Vn.D[lane]</code>	<code>Vd.2D -> result</code>	v7/A32/A64
<code>poly64x1_t vdup_lane_p64(poly64x1_t vec, const int lane)</code>	<code>vec -> Vn.1D 0 <= lane <= 0</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	A32/A64
<code>poly64x2_t vdupq_lane_p64(poly64x1_t vec, const int lane)</code>	<code>vec -> Vn.1D 0 <= lane <= 0</code>	<code>DUP Vd.2D,Vn.D[lane]</code>	<code>Vd.2D -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vdup_lane_f32(float32x2_t vec, const int lane)</code>	<code>vec -> Vn.2S 0 <= lane <= 1</code>	<code>DUP Vd.2S,Vn.S[lane]</code>	<code>Vd.2S -> result</code>	v7/A32/A64
<code>float32x4_t vdupq_lane_f32(float32x2_t vec, const int lane)</code>	<code>vec -> Vn.2S 0 <= lane <= 1</code>	<code>DUP Vd.4S,Vn.S[lane]</code>	<code>Vd.4S -> result</code>	v7/A32/A64
<code>poly8x8_t vdup_lane_p8(poly8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Vd.8B,Vn.B[lane]</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x16_t vdupq_lane_p8(poly8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Vd.16B,Vn.B[lane]</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly16x4_t vdup_lane_p16(poly16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>poly16x8_t vdupq_lane_p16(poly16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>float64x1_t vdup_lane_f64(float64x1_t vec, const int lane)</code>	<code>vec -> Vn.1D 0 <= lane <= 0</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2_t vdupq_lane_f64(float64x1_t vec, const int lane)</code>	<code>vec -> Vn.1D 0 <= lane <= 0</code>	<code>DUP Vd.2D,Vn.D[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>mfloat8x8_t vdup_lane_mf8(mfloat8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Vd.8B,Vn.B[lane]</code>	<code>Vd.8B -> result</code>	/A64
<code>mfloat8x16_t vdupq_lane_mf8(mfloat8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Vd.16B,Vn.B[lane]</code>	<code>Vd.16B -> result</code>	A64
<code>int8x8_t vdup_laneq_s8(int8x16_t vec, const int lane)</code>	<code>vec -> Vn.16B 0 <= lane <= 15</code>	<code>DUP Vd.8B,Vn.B[lane]</code>	<code>Vd.8B -> result</code>	A64
<code>int8x16_t vdupq_laneq_s8(int8x16_t vec, const int lane)</code>	<code>vec -> Vn.16B 0 <= lane <= 15</code>	<code>DUP Vd.16B,Vn.B[lane]</code>	<code>Vd.16B -> result</code>	A64
<code>int16x4_t vdup_laneq_s16(int16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vdupq_laneq_s16(int16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vdup_laneq_s32(int32x4_t vec, const int lane)</code>	vec -> Vn.4S 0 <= lane <= 3	DUP Vd.2S,Vn.S[lane]	Vd.2S -> result	A64
<code>int32x4_t vdupq_laneq_s32(int32x4_t vec, const int lane)</code>	vec -> Vn.4S 0 <= lane <= 3	DUP Vd.4S,Vn.S[lane]	Vd.4S -> result	A64
<code>int64x1_t vdup_laneq_s64(int64x2_t vec, const int lane)</code>	vec -> Vn.2D 0 <= lane <= 1	DUP Dd,Vn.D[lane]	Dd -> result	A64
<code>int64x2_t vdupq_laneq_s64(int64x2_t vec, const int lane)</code>	vec -> Vn.2D 0 <= lane <= 1	DUP Vd.2D,Vn.D[lane]	Vd.2D -> result	A64
<code>uint8x8_t vdup_laneq_u8(uint8x16_t vec, const int lane)</code>	vec -> Vn.16B 0 <= lane <= 15	DUP Vd.8B,Vn.B[lane]	Vd.8B -> result	A64
<code>uint8x16_t vdupq_laneq_u8(uint8x16_t vec, const int lane)</code>	vec -> Vn.16B 0 <= lane <= 15	DUP Vd.16B,Vn.B[lane]	Vd.16B -> result	A64
<code>uint16x4_t vdup_laneq_u16(uint16x8_t vec, const int lane)</code>	vec -> Vn.8H 0 <= lane <= 7	DUP Vd.4H,Vn.H[lane]	Vd.4H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vdupq_laneq_u16(uint16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x2_t vdup_laneq_u32(uint32x4_t vec, const int lane)</code>	<code>vec -> Vn.4S 0 <= lane <= 3</code>	<code>DUP Vd.2S,Vn.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vdupq_laneq_u32(uint32x4_t vec, const int lane)</code>	<code>vec -> Vn.4S 0 <= lane <= 3</code>	<code>DUP Vd.4S,Vn.S[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x1_t vdup_laneq_u64(uint64x2_t vec, const int lane)</code>	<code>vec -> Vn.2D 0 <= lane <= 1</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	A64
<code>uint64x2_t vdupq_laneq_u64(uint64x2_t vec, const int lane)</code>	<code>vec -> Vn.2D 0 <= lane <= 1</code>	<code>DUP Vd.2D,Vn.D[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>poly64x1_t vdup_laneq_p64(poly64x2_t vec, const int lane)</code>	<code>vec -> Vn.2D 0 <= lane <= 1</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	A64
<code>poly64x2_t vdupq_laneq_p64(poly64x2_t vec, const int lane)</code>	<code>vec -> Vn.2D 0 <= lane <= 1</code>	<code>DUP Vd.2D,Vn.D[lane]</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vdup_laneq_f32(float32x4_t vec, const int lane)</code>	<code>vec -> Vn.4S 0 <= lane <= 3</code>	<code>DUP Vd.2S,Vn.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vdupq_laneq_f32(float32x4_t vec, const int lane)</code>	<code>vec -> Vn.4S 0 <= lane <= 3</code>	<code>DUP Vd.4S,Vn.S[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>poly8x8_t vdup_laneq_p8(poly8x16_t vec, const int lane)</code>	<code>vec -> Vn.16B 0 <= lane <= 15</code>	<code>DUP Vd.8B,Vn.B[lane]</code>	<code>Vd.8B -> result</code>	A64
<code>poly8x16_t vdupq_laneq_p8(poly8x16_t vec, const int lane)</code>	<code>vec -> Vn.16B 0 <= lane <= 15</code>	<code>DUP Vd.16B,Vn.B[lane]</code>	<code>Vd.16B -> result</code>	A64
<code>poly16x4_t vdup_laneq_p16(poly16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>poly16x8_t vdupq_laneq_p16(poly16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>float64x1_t vdup_laneq_f64(float64x2_t vec, const int lane)</code>	<code>vec -> Vn.2D 0 <= lane <= 1</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2_t vdupq_laneq_f64(float64x2_t vec, const int lane)</code>	<code>vec -> Vn.2D 0 <= lane <= 1</code>	<code>DUP Vd.2D,Vn.D[lane]</code>	<code>Vd.2D -> result</code>	A64
<code>mfloat8x8_t vdup_laneq_mf8(mfloat8x16_t vec, const int lane)</code>	<code>vec -> Vn.16B 0 <= lane <= 15</code>	<code>DUP Vd.8B,Vn.B[lane]</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vdupq_laneq_mf8(mfloat8x16_t vec, const int lane)</code>	<code>vec -> Vn.16B 0 <= lane <= 15</code>	<code>DUP Vd.16B,Vn.B[lane]</code>	<code>Vd.16B -> result</code>	A64

2.1.9.5 Combine vectors

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vcombine_s8(int8x8_t low, int8x8_t high)</code>	<code>low -> Vn.8B high -> Vm.8B</code>	<code>DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x8_t vcombine_s16(int16x4_t low, int16x4_t high)</code>	<code>low -> Vn.4H high -> Vm.4H</code>	<code>DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vcombine_s32(int32x2_t low, int32x2_t high)</code>	low -> Vn.2S high -> Vm.2S	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.4S -> result	v7/A32/A64
<code>int64x2_t vcombine_s64(int64x1_t low, int64x1_t high)</code>	low -> Vn.1D high -> Vm.1D	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.2D -> result	v7/A32/A64
<code>uint8x16_t vcombine_u8(uint8x8_t low, uint8x8_t high)</code>	low -> Vn.8B high -> Vm.8B	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.16B -> result	v7/A32/A64
<code>uint16x8_t vcombine_u16(uint16x4_t low, uint16x4_t high)</code>	low -> Vn.4H high -> Vm.4H	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.8H -> result	v7/A32/A64
<code>uint32x4_t vcombine_u32(uint32x2_t low, uint32x2_t high)</code>	low -> Vn.2S high -> Vm.2S	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.4S -> result	v7/A32/A64
<code>uint64x2_t vcombine_u64(uint64x1_t low, uint64x1_t high)</code>	low -> Vn.1D high -> Vm.1D	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.2D -> result	v7/A32/A64
<code>poly64x2_t vcombine_p64(poly64x1_t low, poly64x1_t high)</code>	low -> Vn.1D high -> Vm.1D	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.2D -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vcombine_f16(float16x4_t low, float16x4_t high)</code>	low -> Vn.4H high -> Vm.4H	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.8H -> result	v7/A32/A64
<code>float32x4_t vcombine_f32(float32x2_t low, float32x2_t high)</code>	low -> Vn.2S high -> Vm.2S	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.4S -> result	v7/A32/A64
<code>poly8x16_t vcombine_p8(poly8x8_t low, poly8x8_t high)</code>	low -> Vn.8B high -> Vm.8B	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.16B -> result	v7/A32/A64
<code>poly16x8_t vcombine_p16(poly16x4_t low, poly16x4_t high)</code>	low -> Vn.4H high -> Vm.4H	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.8H -> result	v7/A32/A64
<code>float64x2_t vcombine_f64(float64x1_t low, float64x1_t high)</code>	low -> Vn.1D high -> Vm.1D	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.2D -> result	A64
<code>mfloat8x16_t vcombine_mf8(mfloat8x8_t low, mfloat8x8_t high)</code>	low -> Vn.8B high -> Vm.8B	DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]	Vd.16B -> result	A64

2.1.9.6 Split vectors

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vget_high_s8(int8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>int16x4_t vget_high_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>int32x2_t vget_high_s32(int32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>int64x1_t vget_high_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.1D -> result</code>	<code>v7/A32/A64</code>
<code>uint8x8_t vget_high_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>uint16x4_t vget_high_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>uint32x2_t vget_high_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>uint64x1_t vget_high_u64(uint64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.1D -> result</code>	<code>v7/A32/A64</code>
<code>poly64x1_t vget_high_p64(poly64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.1D -> result</code>	<code>A32/A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vget_high_f16(float16x8_t a)	a -> Vn.8H	DUP Vd.1D,Vn.D[1]	Vd.4H -> result	v7/A32/A64
float32x2_t vget_high_f32(float32x4_t a)	a -> Vn.4S	DUP Vd.1D,Vn.D[1]	Vd.2S -> result	v7/A32/A64
poly8x8_t vget_high_p8(poly8x16_t a)	a -> Vn.16B	DUP Vd.1D,Vn.D[1]	Vd.8B -> result	v7/A32/A64
poly16x4_t vget_high_p16(poly16x8_t a)	a -> Vn.8H	DUP Vd.1D,Vn.D[1]	Vd.4H -> result	v7/A32/A64
float64x1_t vget_high_f64(float64x2_t a)	a -> Vn.2D	DUP Vd.1D,Vn.D[1]	Vd.1D -> result	A64
mfloat8x8_t vget_high_mf8(mfloat8x16_t a)	a -> Vn.16B	DUP Vd.1D,Vn.D[1]	Vd.8B -> result	A64
int8x8_t vget_low_s8(int8x16_t a)	a -> Vn.16B	DUP Vd.1D,Vn.D[0]	Vd.8B -> result	v7/A32/A64
int16x4_t vget_low_s16(int16x8_t a)	a -> Vn.8H	DUP Vd.1D,Vn.D[0]	Vd.4H -> result	v7/A32/A64
int32x2_t vget_low_s32(int32x4_t a)	a -> Vn.4S	DUP Vd.1D,Vn.D[0]	Vd.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1_t vget_low_s64(int64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.1D -> result</code>	<code>v7/A32/A64</code>
<code>uint8x8_t vget_low_u8(uint8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>uint16x4_t vget_low_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>uint32x2_t vget_low_u32(uint32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>uint64x1_t vget_low_u64(uint64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.1D -> result</code>	<code>v7/A32/A64</code>
<code>poly64x1_t vget_low_p64(poly64x2_t a)</code>	<code>a -> Vn.2D</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.1D -> result</code>	<code>A32/A64</code>
<code>float16x4_t vget_low_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.4H -> result</code>	<code>v7/A32/A64</code>
<code>float32x2_t vget_low_f32(float32x4_t a)</code>	<code>a -> Vn.4S</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.2S -> result</code>	<code>v7/A32/A64</code>
<code>poly8x8_t vget_low_p8(poly8x16_t a)</code>	<code>a -> Vn.16B</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly16x4_t vget_low_p16(poly16x8_t a)	a -> Vn.8H	DUP Vd.1D,Vn.D[0]	Vd.4H -> result	v7/A32/A64
float64x1_t vget_low_f64(float64x2_t a)	a -> Vn.2D	DUP Vd.1D,Vn.D[0]	Vd.1D -> result	A64
mfloat8x8_t vget_low_mf8(mfloat8x16_t a)	a -> Vn.16B	DUP Vd.1D,Vn.D[0]	Vd.8B -> result	A64

2.1.9.7 Extract one element from vector

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8_t vdupb_lane_s8(int8x8_t vec, const int lane)	vec -> Vn.8B 0 <= lane <= 7	DUP Bd,Vn.B[lane]	Bd -> result	A64
int16_t vduph_lane_s16(int16x4_t vec, const int lane)	vec -> Vn.4H 0 <= lane <= 3	DUP Hd,Vn.H[lane]	Hd -> result	A64
int32_t vdups_lane_s32(int32x2_t vec, const int lane)	vec -> Vn.2S 0 <= lane <= 1	DUP Sd,Vn.S[lane]	Sd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64_t vdupd_lane_s64(int64x1_t vec, const int lane)</code>	vec -> Vn.1D 0 <= lane <= 0	DUP Dd,Vn.D[lane]	Dd -> result	A64
<code>uint8_t vdupb_lane_u8(uint8x8_t vec, const int lane)</code>	vec -> Vn.8B 0 <= lane <= 7	DUP Bd,Vn.B[lane]	Bd -> result	A64
<code>uint16_t vduph_lane_u16(uint16x4_t vec, const int lane)</code>	vec -> Vn.4H 0 <= lane <= 3	DUP Hd,Vn.H[lane]	Hd -> result	A64
<code>uint32_t vdups_lane_u32(uint32x2_t vec, const int lane)</code>	vec -> Vn.2S 0 <= lane <= 1	DUP Sd,Vn.S[lane]	Sd -> result	A64
<code>uint64_t vdupd_lane_u64(uint64x1_t vec, const int lane)</code>	vec -> Vn.1D 0 <= lane <= 0	DUP Dd,Vn.D[lane]	Dd -> result	A64
<code>float32_t vdups_lane_f32(float32x2_t vec, const int lane)</code>	vec -> Vn.2S 0 <= lane <= 1	DUP Sd,Vn.S[lane]	Sd -> result	A64
<code>float64_t vdupd_lane_f64(float64x1_t vec, const int lane)</code>	vec -> Vn.1D 0 <= lane <= 0	DUP Dd,Vn.D[lane]	Dd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8_t vdupb_lane_p8(poly8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Bd,Vn.B[lane]</code>	<code>Bd -> result</code>	A64
<code>poly16_t vduph_lane_p16(poly16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Hd,Vn.H[lane]</code>	<code>Hd -> result</code>	A64
<code>mfloat8_t vdupb_lane_mf8(mfloat8x8_t vec, const int lane)</code>	<code>vec -> Vn.8B 0 <= lane <= 7</code>	<code>DUP Bd,Vn.B[lane]</code>	<code>Bd -> result</code>	A64
<code>int8_t vdupb_laneq_s8(int8x16_t vec, const int lane)</code>	<code>vec -> Vn.16B 0 <= lane <= 15</code>	<code>DUP Bd,Vn.B[lane]</code>	<code>Bd -> result</code>	A64
<code>int16_t vduph_laneq_s16(int16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Hd,Vn.H[lane]</code>	<code>Hd -> result</code>	A64
<code>int32_t vdups_laneq_s32(int32x4_t vec, const int lane)</code>	<code>vec -> Vn.4S 0 <= lane <= 3</code>	<code>DUP Sd,Vn.S[lane]</code>	<code>Sd -> result</code>	A64
<code>int64_t vdupd_laneq_s64(int64x2_t vec, const int lane)</code>	<code>vec -> Vn.2D 0 <= lane <= 1</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8_t vdupb_laneq_u8(uint8x16_t vec, const int lane)</code>	vec -> Vn.16B 0 <= lane <= 15	DUP Bd,Vn.B[lane]	Bd -> result	A64
<code>uint16_t vduph_laneq_u16(uint16x8_t vec, const int lane)</code>	vec -> Vn.8H 0 <= lane <= 7	DUP Hd,Vn.H[lane]	Hd -> result	A64
<code>uint32_t vdups_laneq_u32(uint32x4_t vec, const int lane)</code>	vec -> Vn.4S 0 <= lane <= 3	DUP Sd,Vn.S[lane]	Sd -> result	A64
<code>uint64_t vdupd_laneq_u64(uint64x2_t vec, const int lane)</code>	vec -> Vn.2D 0 <= lane <= 1	DUP Dd,Vn.D[lane]	Dd -> result	A64
<code>float32_t vdups_laneq_f32(float32x4_t vec, const int lane)</code>	vec -> Vn.4S 0 <= lane <= 3	DUP Sd,Vn.S[lane]	Sd -> result	A64
<code>float64_t vdupd_laneq_f64(float64x2_t vec, const int lane)</code>	vec -> Vn.2D 0 <= lane <= 1	DUP Dd,Vn.D[lane]	Dd -> result	A64
<code>poly8_t vdupb_laneq_p8(poly8x16_t vec, const int lane)</code>	vec -> Vn.16B 0 <= lane <= 15	DUP Bd,Vn.B[lane]	Bd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16_t vduph_laneq_p16(poly16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Hd,Vn.H[lane]</code>	<code>Hd -> result</code>	A64
<code>mfloat8_t vdupb_laneq_mf8(mfloat8x16_t vec, const int lane)</code>	<code>vec -> Vn.16B 0 <= lane <= 15</code>	<code>DUP Bd,Vn.B[lane]</code>	<code>Bd -> result</code>	A64
<code>uint8_t vget_lane_u8(uint8x8_t v, const int lane)</code>	<code>0<=lane<=7 v -> Vn.8B</code>	<code>UMOV Rd,Vn.B[lane]</code>	<code>Rd -> result</code>	v7/A32/A64
<code>uint16_t vget_lane_u16(uint16x4_t v, const int lane)</code>	<code>0<=lane<=3 v -> Vn.4H</code>	<code>UMOV Rd,Vn.H[lane]</code>	<code>Rd -> result</code>	v7/A32/A64
<code>uint32_t vget_lane_u32(uint32x2_t v, const int lane)</code>	<code>0<=lane<=1 v -> Vn.2S</code>	<code>UMOV Rd,Vn.S[lane]</code>	<code>Rd -> result</code>	v7/A32/A64
<code>uint64_t vget_lane_u64(uint64x1_t v, const int lane)</code>	<code>lane==0 v -> Vn.1D</code>	<code>UMOV Rd,Vn.D[lane]</code>	<code>Rd -> result</code>	v7/A32/A64
<code>poly64_t vget_lane_p64(poly64x1_t v, const int lane)</code>	<code>lane==0 v -> Vn.1D</code>	<code>UMOV Rd,Vn.D[lane]</code>	<code>Rd -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8_t vget_lane_s8(int8x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ $v \rightarrow Vn.8B$	SMOV Rd,Vn.B[lane]	Rd -> result	v7/A32/A64
<code>int16_t vget_lane_s16(int16x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ $v \rightarrow Vn.4H$	SMOV Rd,Vn.H[lane]	Rd -> result	v7/A32/A64
<code>int32_t vget_lane_s32(int32x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ $v \rightarrow Vn.2S$	SMOV Rd,Vn.S[lane]	Rd -> result	v7/A32/A64
<code>int64_t vget_lane_s64(int64x1_t v, const int lane)</code>	$\text{lane} == 0$ $v \rightarrow Vn.1D$	UMOV Rd,Vn.D[lane]	Rd -> result	v7/A32/A64
<code>poly8_t vget_lane_p8(poly8x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ $v \rightarrow Vn.8B$	UMOV Rd,Vn.B[lane]	Rd -> result	v7/A32/A64
<code>poly16_t vget_lane_p16(poly16x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ $v \rightarrow Vn.4H$	UMOV Rd,Vn.H[lane]	Rd -> result	v7/A32/A64
<code>mfloat8_t vget_lane_mf8(mfloat8x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ $v \rightarrow Vn.8B$	DUP Bd,Vn.B[lane]	Bd -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vget_lane_f16(float16x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ <code>v -> Vn.4H</code>	<code>DUP Hd,Vn.H[lane]</code>	<code>Hd -> result</code>	v7/A32/A64
<code>float32_t vget_lane_f32(float32x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ <code>v -> Vn.2S</code>	<code>DUP Sd,Vn.S[lane]</code>	<code>Sd -> result</code>	v7/A32/A64
<code>float64_t vget_lane_f64(float64x1_t v, const int lane)</code>	<code>lane==0</code> <code>v -> Vn.1D</code>	<code>DUP Dd,Vn.D[lane]</code>	<code>Dd -> result</code>	A64
<code>uint8_t vgetq_lane_u8(uint8x16_t v, const int lane)</code>	$0 \leq \text{lane} \leq 15$ <code>v -> Vn.16B</code>	<code>UMOV Rd,Vn.B[lane]</code>	<code>Rd -> result</code>	v7/A32/A64
<code>uint16_t vgetq_lane_u16(uint16x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ <code>v -> Vn.8H</code>	<code>UMOV Rd,Vn.H[lane]</code>	<code>Rd -> result</code>	v7/A32/A64
<code>uint32_t vgetq_lane_u32(uint32x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ <code>v -> Vn.4S</code>	<code>UMOV Rd,Vn.S[lane]</code>	<code>Rd -> result</code>	v7/A32/A64
<code>uint64_t vgetq_lane_u64(uint64x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ <code>v -> Vn.2D</code>	<code>UMOV Rd,Vn.D[lane]</code>	<code>Rd -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly64_t vgetq_lane_p64(poly64x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ $v \rightarrow Vn.2D$	UMOV Rd,Vn.D[lane]	Rd -> result	A32/A64
<code>int8_t vgetq_lane_s8(int8x16_t v, const int lane)</code>	$0 \leq \text{lane} \leq 15$ $v \rightarrow Vn.16B$	SMOV Rd,Vn.B[lane]	Rd -> result	v7/A32/A64
<code>int16_t vgetq_lane_s16(int16x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ $v \rightarrow Vn.8H$	SMOV Rd,Vn.H[lane]	Rd -> result	v7/A32/A64
<code>int32_t vgetq_lane_s32(int32x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ $v \rightarrow Vn.4S$	SMOV Rd,Vn.S[lane]	Rd -> result	v7/A32/A64
<code>int64_t vgetq_lane_s64(int64x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ $v \rightarrow Vn.2D$	UMOV Rd,Vn.D[lane]	Rd -> result	v7/A32/A64
<code>poly8_t vgetq_lane_p8(poly8x16_t v, const int lane)</code>	$0 \leq \text{lane} \leq 15$ $v \rightarrow Vn.16B$	UMOV Rd,Vn.B[lane]	Rd -> result	v7/A32/A64
<code>poly16_t vgetq_lane_p16(poly16x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ $v \rightarrow Vn.8H$	UMOV Rd,Vn.H[lane]	Rd -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8_t vgetq_lane_mf8(mfloat8x16_t v, const int lane)</code>	$0 \leq \text{lane} \leq 15$ <code>v -> Vn.16B</code>	DUP Bd,Vn.B[lane]	Bd -> result	v7/A32/A64
<code>float16_t vgetq_lane_f16(float16x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ <code>v -> Vn.8H</code>	DUP Hd,Vn.H[lane]	Hd -> result	v7/A32/A64
<code>float32_t vgetq_lane_f32(float32x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ <code>v -> Vn.4S</code>	DUP Sd,Vn.S[lane]	Sd -> result	v7/A32/A64
<code>float64_t vgetq_lane_f64(float64x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ <code>v -> Vn.2D</code>	DUP Dd,Vn.D[lane]	Dd -> result	A64

2.1.9.8 Extract vector from a pair of vectors

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vext_s8(int8x8_t a, int8x8_t b, const int n)</code>	<code>a -> Vn.8B</code> <code>b -> Vm.8B</code> $0 \leq n \leq 7$	EXT Vd.8B,Vn.8B,Vm.8B,#n	Vd.8B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vextq_s8(int8x16_t a, int8x16_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 15</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vext_s16(int16x4_t a, int16x4_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B 0 <= n <= 3</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#(n<<1)</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int16x8_t vextq_s16(int16x8_t a, int16x8_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 7</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#(n<<1)</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int32x2_t vext_s32(int32x2_t a, int32x2_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B 0 <= n <= 1</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#(n<<2)</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int32x4_t vextq_s32(int32x4_t a, int32x4_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 3</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#(n<<2)</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1_t vext_s64(int64x1_t a, int64x1_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B n == 0</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#(n<<3)</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int64x2_t vextq_s64(int64x2_t a, int64x2_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 1</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#(n<<3)</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint8x8_t vext_u8(uint8x8_t a, uint8x8_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B 0 <= n <= 7</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#n</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vextq_u8(uint8x16_t a, uint8x16_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 15</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#n</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vext_u16(uint16x4_t a, uint16x4_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B 0 <= n <= 3</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#(n<<1)</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vextq_u16(uint16x8_t a, uint16x8_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 7</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#(n<<1)</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint32x2_t vext_u32(uint32x2_t a, uint32x2_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B 0 <= n <= 1</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#(n<<2)</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint32x4_t vextq_u32(uint32x4_t a, uint32x4_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 3</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#(n<<2)</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint64x1_t vext_u64(uint64x1_t a, uint64x1_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B n == 0</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#(n<<3)</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint64x2_t vextq_u64(uint64x2_t a, uint64x2_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 1</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#(n<<3)</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x1_t vext_p64(poly64x1_t a, poly64x1_t b, const int n)	a -> Vn.8B b -> Vm.8B n == 0	EXT Vd.8B,Vn.8B,Vm.8B,#(n<<3)	Vd.8B -> result	A32/A64
poly64x2_t vextq_p64(poly64x2_t a, poly64x2_t b, const int n)	a -> Vn.16B b -> Vm.16B 0 <= n <= 1	EXT Vd.16B,Vn.16B,Vm.16B,#(n<<3)	Vd.16B -> result	A32/A64
float32x2_t vext_f32(float32x2_t a, float32x2_t b, const int n)	a -> Vn.8B b -> Vm.8B 0 <= n <= 1	EXT Vd.8B,Vn.8B,Vm.8B,#(n<<2)	Vd.8B -> result	v7/A32/A64
float32x4_t vextq_f32(float32x4_t a, float32x4_t b, const int n)	a -> Vn.16B b -> Vm.16B 0 <= n <= 3	EXT Vd.16B,Vn.16B,Vm.16B,#(n<<2)	Vd.16B -> result	v7/A32/A64
float64x1_t vext_f64(float64x1_t a, float64x1_t b, const int n)	a -> Vn.8B b -> Vm.8B n == 0	EXT Vd.8B,Vn.8B,Vm.8B,#(n<<3)	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x2_t vextq_f64(float64x2_t a, float64x2_t b, const int n)	a -> Vn.16B b -> Vm.16B 0 <= n <= 1	EXT Vd.16B,Vn.16B,Vm.16B,#(n<<3)	Vd.16B -> result	A64
poly8x8_t vext_p8(poly8x8_t a, poly8x8_t b, const int n)	a -> Vn.8B b -> Vm.8B 0 <= n <= 7	EXT Vd.8B,Vn.8B,Vm.8B,#n	Vd.8B -> result	v7/A32/A64
poly8x16_t vextq_p8(poly8x16_t a, poly8x16_t b, const int n)	a -> Vn.16B b -> Vm.16B 0 <= n <= 15	EXT Vd.16B,Vn.16B,Vm.16B,#n	Vd.16B -> result	v7/A32/A64
poly16x4_t vext_p16(poly16x4_t a, poly16x4_t b, const int n)	a -> Vn.8B b -> Vm.8B 0 <= n <= 3	EXT Vd.8B,Vn.8B,Vm.8B,#(n<<1)	Vd.8B -> result	v7/A32/A64
poly16x8_t vextq_p16(poly16x8_t a, poly16x8_t b, const int n)	a -> Vn.16B b -> Vm.16B 0 <= n <= 7	EXT Vd.16B,Vn.16B,Vm.16B,#(n<<1)	Vd.16B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x8_t vext_mf8(mfloat8x8_t a, mfloat8x8_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B 0 <= n <= 7</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#n</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vextq_mf8(mfloat8x16_t a, mfloat8x16_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 15</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#n</code>	<code>Vd.16B -> result</code>	A64

2.1.9.9 Reverse elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vrev64_s8(int8x8_t vec)</code>	<code>vec -> Vn.8B</code>	<code>REV64 Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vrev64q_s8(int8x16_t vec)</code>	<code>vec -> Vn.16B</code>	<code>REV64 Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>int16x4_t vrev64_s16(int16x4_t vec)</code>	<code>vec -> Vn.4H</code>	<code>REV64 Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>int16x8_t vrev64q_s16(int16x8_t vec)</code>	<code>vec -> Vn.8H</code>	<code>REV64 Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vrev64_s32(int32x2_t vec)</code>	<code>vec -> Vn.2S</code>	REV64 Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
<code>int32x4_t vrev64q_s32(int32x4_t vec)</code>	<code>vec -> Vn.4S</code>	REV64 Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
<code>uint8x8_t vrev64_u8(uint8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV64 Vd.8B,Vn.8B	Vd.8B -> result	v7/A32/A64
<code>uint8x16_t vrev64q_u8(uint8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV64 Vd.16B,Vn.16B	Vd.16B -> result	v7/A32/A64
<code>uint16x4_t vrev64_u16(uint16x4_t vec)</code>	<code>vec -> Vn.4H</code>	REV64 Vd.4H,Vn.4H	Vd.4H -> result	v7/A32/A64
<code>uint16x8_t vrev64q_u16(uint16x8_t vec)</code>	<code>vec -> Vn.8H</code>	REV64 Vd.8H,Vn.8H	Vd.8H -> result	v7/A32/A64
<code>uint32x2_t vrev64_u32(uint32x2_t vec)</code>	<code>vec -> Vn.2S</code>	REV64 Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64
<code>uint32x4_t vrev64q_u32(uint32x4_t vec)</code>	<code>vec -> Vn.4S</code>	REV64 Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
<code>float32x2_t vrev64_f32(float32x2_t vec)</code>	<code>vec -> Vn.2S</code>	REV64 Vd.2S,Vn.2S	Vd.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vrev64q_f32(float32x4_t vec)</code>	<code>vec -> Vn.4S</code>	REV64 Vd.4S,Vn.4S	Vd.4S -> result	v7/A32/A64
<code>poly8x8_t vrev64_p8(poly8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV64 Vd.8B,Vn.8B	Vd.8B -> result	v7/A32/A64
<code>poly8x16_t vrev64q_p8(poly8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV64 Vd.16B,Vn.16B	Vd.16B -> result	v7/A32/A64
<code>poly16x4_t vrev64_p16(poly16x4_t vec)</code>	<code>vec -> Vn.4H</code>	REV64 Vd.4H,Vn.4H	Vd.4H -> result	v7/A32/A64
<code>poly16x8_t vrev64q_p16(poly16x8_t vec)</code>	<code>vec -> Vn.8H</code>	REV64 Vd.8H,Vn.8H	Vd.8H -> result	v7/A32/A64
<code>mfloat8x8_t vrev64_mf8(mfloat8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV64 Vd.8B,Vn.8B	Vd.8B -> result	A64
<code>mfloat8x16_t vrev64q_mf8(mfloat8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV64 Vd.16B,Vn.16B	Vd.16B -> result	A64
<code>int8x8_t vrev32_s8(int8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV32 Vd.8B,Vn.8B	Vd.8B -> result	v7/A32/A64
<code>int8x16_t vrev32q_s8(int8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV32 Vd.16B,Vn.16B	Vd.16B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vrev32_s16(int16x4_t vec)</code>	<code>vec -> Vn.4H</code>	REV32 Vd.4H,Vn.4H	Vd.4H -> result	v7/A32/A64
<code>int16x8_t vrev32q_s16(int16x8_t vec)</code>	<code>vec -> Vn.8H</code>	REV32 Vd.8H,Vn.8H	Vd.8H -> result	v7/A32/A64
<code>uint8x8_t vrev32_u8(uint8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV32 Vd.8B,Vn.8B	Vd.8B -> result	v7/A32/A64
<code>uint8x16_t vrev32q_u8(uint8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV32 Vd.16B,Vn.16B	Vd.16B -> result	v7/A32/A64
<code>uint16x4_t vrev32_u16(uint16x4_t vec)</code>	<code>vec -> Vn.4H</code>	REV32 Vd.4H,Vn.4H	Vd.4H -> result	v7/A32/A64
<code>uint16x8_t vrev32q_u16(uint16x8_t vec)</code>	<code>vec -> Vn.8H</code>	REV32 Vd.8H,Vn.8H	Vd.8H -> result	v7/A32/A64
<code>poly8x8_t vrev32_p8(poly8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV32 Vd.8B,Vn.8B	Vd.8B -> result	v7/A32/A64
<code>poly8x16_t vrev32q_p8(poly8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV32 Vd.16B,Vn.16B	Vd.16B -> result	v7/A32/A64
<code>poly16x4_t vrev32_p16(poly16x4_t vec)</code>	<code>vec -> Vn.4H</code>	REV32 Vd.4H,Vn.4H	Vd.4H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x8_t vrev32q_p16(poly16x8_t vec)</code>	<code>vec -> Vn.8H</code>	REV32 <code>Vd.8H, Vn.8H</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>mfloat8x8_t vrev32_mf8(mfloat8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV32 <code>Vd.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vrev32q_mf8(mfloat8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV32 <code>Vd.16B, Vn.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int8x8_t vrev16_s8(int8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV16 <code>Vd.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>int8x16_t vrev16q_s8(int8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV16 <code>Vd.16B, Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>uint8x8_t vrev16_u8(uint8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV16 <code>Vd.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vrev16q_u8(uint8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV16 <code>Vd.16B, Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64
<code>poly8x8_t vrev16_p8(poly8x8_t vec)</code>	<code>vec -> Vn.8B</code>	REV16 <code>Vd.8B, Vn.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x16_t vrev16q_p8(poly8x16_t vec)</code>	<code>vec -> Vn.16B</code>	REV16 <code>Vd.16B, Vn.16B</code>	<code>Vd.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x8_t vrev16_mf8(mfloat8x8_t vec)</code>	<code>vec -> Vn.8B</code>	<code>REV16 Vd.8B,Vn.8B</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vrev16q_mf8(mfloat8x16_t vec)</code>	<code>vec -> Vn.16B</code>	<code>REV16 Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	A64

2.1.9.10 Zip elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vzip1_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ZIP1 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x16_t vzip1q_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ZIP1 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int16x4_t vzip1_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>ZIP1 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vzip1q_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ZIP1 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vzip1_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>ZIP1 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vzip1q_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ZIP1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vzip1q_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ZIP1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint8x8_t vzip1_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ZIP1 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vzip1q_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ZIP1 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x4_t vzip1_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>ZIP1 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>uint16x8_t vzip1q_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ZIP1 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vzip1_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>ZIP1 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vzip1q_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ZIP1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vzip1q_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ZIP1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vzip1q_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ZIP1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32x2_t vzip1_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>ZIP1 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vzip1q_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ZIP1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float64x2_t vzip1q_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ZIP1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x8_t vzip1_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ZIP1 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
<code>poly8x16_t vzip1q_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ZIP1 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
<code>poly16x4_t vzip1_p16(poly16x4_t a, poly16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	ZIP1 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
<code>poly16x8_t vzip1q_p16(poly16x8_t a, poly16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	ZIP1 Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64
<code>mffloat8x8_t vzip1_mf8(mffloat8x8_t a, mffloat8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ZIP1 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
<code>mffloat8x16_t vzip1q_mf8(mffloat8x16_t a, mffloat8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ZIP1 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
<code>int8x8_t vzip2_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ZIP2 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vzip2q_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>ZIP2 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int16x4_t vzip2_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>ZIP2 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vzip2q_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>ZIP2 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>int32x2_t vzip2_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>ZIP2 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vzip2q_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>ZIP2 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vzip2q_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>ZIP2 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint8x8_t vzip2_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>ZIP2 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vzip2q_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ZIP2 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
<code>uint16x4_t vzip2_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	ZIP2 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
<code>uint16x8_t vzip2q_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	ZIP2 Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64
<code>uint32x2_t vzip2_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	ZIP2 Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	A64
<code>uint32x4_t vzip2q_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	ZIP2 Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	A64
<code>uint64x2_t vzip2q_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	ZIP2 Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64
<code>poly64x2_t vzip2q_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	ZIP2 Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vzip2_f32(float32x2_t a, float32x2_t b)	a -> Vn.2S b -> Vm.2S	ZIP2 Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	A64
float32x4_t vzip2q_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	ZIP2 Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	A64
float64x2_t vzip2q_f64(float64x2_t a, float64x2_t b)	a -> Vn.2D b -> Vm.2D	ZIP2 Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64
poly8x8_t vzip2_p8(poly8x8_t a, poly8x8_t b)	a -> Vn.8B b -> Vm.8B	ZIP2 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
poly8x16_t vzip2q_p8(poly8x16_t a, poly8x16_t b)	a -> Vn.16B b -> Vm.16B	ZIP2 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
poly16x4_t vzip2_p16(poly16x4_t a, poly16x4_t b)	a -> Vn.4H b -> Vm.4H	ZIP2 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
poly16x8_t vzip2q_p16(poly16x8_t a, poly16x8_t b)	a -> Vn.8H b -> Vm.8H	ZIP2 Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x8_t vzip2_mf8(mfloat8x8_t a, mfloat8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ZIP2 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
<code>mfloat8x16_t vzip2q_mf8(mfloat8x16_t a, mfloat8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ZIP2 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
<code>int8x8x2_t vzip_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ZIP1 Vd1.8B,Vn.8B,Vm.8B ZIP2 Vd2.8B,Vn.8B,Vm.8B	Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]	v7/A32/A64
<code>int16x4x2_t vzip_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	ZIP1 Vd1.4H,Vn.4H,Vm.4H ZIP2 Vd2.4H,Vn.4H,Vm.4H	Vd1.4H -> result.val[0] Vd2.4H -> result.val[1]	v7/A32/A64
<code>uint8x8x2_t vzip_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ZIP1 Vd1.8B,Vn.8B,Vm.8B ZIP2 Vd2.8B,Vn.8B,Vm.8B	Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]	v7/A32/A64
<code>uint16x4x2_t vzip_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	ZIP1 Vd1.4H,Vn.4H,Vm.4H ZIP2 Vd2.4H,Vn.4H,Vm.4H	Vd1.4H -> result.val[0] Vd2.4H -> result.val[1]	v7/A32/A64
<code>poly8x8x2_t vzip_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	ZIP1 Vd1.8B,Vn.8B,Vm.8B ZIP2 Vd2.8B,Vn.8B,Vm.8B	Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly16x4x2_t vzip_p16(poly16x4_t a, poly16x4_t b)	a -> Vn.4H b -> Vm.4H	ZIP1 Vd1.4H,Vn.4H,Vm.4H ZIP2 Vd2.4H,Vn.4H,Vm.4H	Vd1.4H -> result.val[0] Vd2.4H -> result.val[1]	v7/A32/A64
mfloat8x8x2_t vzip_mf8(mfloat8x8_t a, mfloat8x8_t b)	a -> Vn.8B b -> Vm.8B	ZIP1 Vd1.8B,Vn.8B,Vm.8B ZIP2 Vd2.8B,Vn.8B,Vm.8B	Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]	A64
int32x2x2_t vzip_s32(int32x2_t a, int32x2_t b)	a -> Vn.2S b -> Vm.2S	ZIP1 Vd1.2S,Vn.2S,Vm.2S ZIP2 Vd2.2S,Vn.2S,Vm.2S	Vd1.2S -> result.val[0] Vd2.2S -> result.val[1]	v7/A32/A64
float32x2x2_t vzip_f32(float32x2_t a, float32x2_t b)	a -> Vn.2S b -> Vm.2S	ZIP1 Vd1.2S,Vn.2S,Vm.2S ZIP2 Vd2.2S,Vn.2S,Vm.2S	Vd1.2S -> result.val[0] Vd2.2S -> result.val[1]	v7/A32/A64
uint32x2x2_t vzip_u32(uint32x2_t a, uint32x2_t b)	a -> Vn.2S b -> Vm.2S	ZIP1 Vd1.2S,Vn.2S,Vm.2S ZIP2 Vd2.2S,Vn.2S,Vm.2S	Vd1.2S -> result.val[0] Vd2.2S -> result.val[1]	v7/A32/A64
int8x16x2_t vzipq_s8(int8x16_t a, int8x16_t b)	a -> Vn.16B b -> Vm.16B	ZIP1 Vd1.16B,Vn.16B,Vm.16B ZIP2 Vd2.16B,Vn.16B,Vm.16B	Vd1.16B -> result.val[0] Vd2.16B -> result.val[1]	v7/A32/A64
int16x8x2_t vzipq_s16(int16x8_t a, int16x8_t b)	a -> Vn.8H b -> Vm.8H	ZIP1 Vd1.8H,Vn.8H,Vm.8H ZIP2 Vd2.8H,Vn.8H,Vm.8H	Vd1.8H -> result.val[0] Vd2.8H -> result.val[1]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4x2_t vzipq_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	ZIP1 <code>Vd1.4S, Vn.4S, Vm.4S</code> ZIP2 <code>Vd2.4S, Vn.4S, Vm.4S</code>	<code>Vd1.4S -> result.val[0]</code> <code>Vd2.4S -> result.val[1]</code>	v7/A32/A64
<code>float32x4x2_t vzipq_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	ZIP1 <code>Vd1.4S, Vn.4S, Vm.4S</code> ZIP2 <code>Vd2.4S, Vn.4S, Vm.4S</code>	<code>Vd1.4S -> result.val[0]</code> <code>Vd2.4S -> result.val[1]</code>	v7/A32/A64
<code>uint8x16x2_t vzipq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B</code> <code>b -> Vm.16B</code>	ZIP1 <code>Vd1.16B, Vn.16B, Vm.16B</code> ZIP2 <code>Vd2.16B, Vn.16B, Vm.16B</code>	<code>Vd1.16B -> result.val[0]</code> <code>Vd2.16B -> result.val[1]</code>	v7/A32/A64
<code>uint16x8x2_t vzipq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	ZIP1 <code>Vd1.8H, Vn.8H, Vm.8H</code> ZIP2 <code>Vd2.8H, Vn.8H, Vm.8H</code>	<code>Vd1.8H -> result.val[0]</code> <code>Vd2.8H -> result.val[1]</code>	v7/A32/A64
<code>uint32x4x2_t vzipq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	ZIP1 <code>Vd1.4S, Vn.4S, Vm.4S</code> ZIP2 <code>Vd2.4S, Vn.4S, Vm.4S</code>	<code>Vd1.4S -> result.val[0]</code> <code>Vd2.4S -> result.val[1]</code>	v7/A32/A64
<code>poly8x16x2_t vzipq_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B</code> <code>b -> Vm.16B</code>	ZIP1 <code>Vd1.16B, Vn.16B, Vm.16B</code> ZIP2 <code>Vd2.16B, Vn.16B, Vm.16B</code>	<code>Vd1.16B -> result.val[0]</code> <code>Vd2.16B -> result.val[1]</code>	v7/A32/A64
<code>poly16x8x2_t vzipq_p16(poly16x8_t a, poly16x8_t b)</code>	<code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	ZIP1 <code>Vd1.8H, Vn.8H, Vm.8H</code> ZIP2 <code>Vd2.8H, Vn.8H, Vm.8H</code>	<code>Vd1.8H -> result.val[0]</code> <code>Vd2.8H -> result.val[1]</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x16x2_t vzipq_mf8(mfloat8x16_t a, mfloat8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	ZIP1 <code>Vd.16B, Vn.16B, Vm.16B</code> ZIP2 <code>Vd2.16B, Vn.16B, Vm.16B</code>	<code>Vd.16B -> result.val[0]</code> <code>Vd2.16B -> result.val[1]</code>	A64

2.1.9.11 Unzip elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vuzp1_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	UZP1 <code>Vd.8B, Vn.8B, Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x16_t vuzp1q_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	UZP1 <code>Vd.16B, Vn.16B, Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int16x4_t vuzp1_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	UZP1 <code>Vd.4H, Vn.4H, Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vuzp1q_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	UZP1 <code>Vd.8H, Vn.8H, Vm.8H</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vuzp1_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UZP1 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vuzp1q_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UZP1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vuzp1q_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UZP1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint8x8_t vuzp1_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP1 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vuzp1q_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UZP1 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x4_t vuzp1_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP1 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>uint16x8_t vuzp1q_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UZP1 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vuzp1_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UZP1 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vuzp1q_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UZP1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vuzp1q_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UZP1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vuzp1q_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UZP1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32x2_t vuzp1_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UZP1 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vuzp1q_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UZP1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float64x2_t vuzp1q_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UZP1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x8_t vuzp1_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP1 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>poly8x16_t vuzp1q_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UZP1 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>poly16x4_t vuzp1_p16(poly16x4_t a, poly16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP1 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>poly16x8_t vuzp1q_p16(poly16x8_t a, poly16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UZP1 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>mfloat8x8_t vuzp1_mf8(mfloat8x8_t a, mfloat8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP1 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vuzp1q_mf8(mfloat8x16_t a, mfloat8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UZP1 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int8x8_t vuzp2_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP2 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vuzp2q_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UZP2 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int16x4_t vuzp2_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP2 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vuzp2q_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UZP2 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>int32x2_t vuzp2_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UZP2 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vuzp2q_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UZP2 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vuzp2q_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UZP2 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint8x8_t vuzp2_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP2 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vuzp2q_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UZP2 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x4_t vuzp2_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP2 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>uint16x8_t vuzp2q_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UZP2 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x2_t vuzp2_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UZP2 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vuzp2q_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>UZP2 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vuzp2q_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UZP2 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vuzp2q_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>UZP2 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vuzp2_f32(float32x2_t a, float32x2_t b)	a -> Vn.2S b -> Vm.2S	UZP2 Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	A64
float32x4_t vuzp2q_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	UZP2 Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	A64
float64x2_t vuzp2q_f64(float64x2_t a, float64x2_t b)	a -> Vn.2D b -> Vm.2D	UZP2 Vd.2D,Vn.2D,Vm.2D	Vd.2D -> result	A64
poly8x8_t vuzp2_p8(poly8x8_t a, poly8x8_t b)	a -> Vn.8B b -> Vm.8B	UZP2 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
poly8x16_t vuzp2q_p8(poly8x16_t a, poly8x16_t b)	a -> Vn.16B b -> Vm.16B	UZP2 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
poly16x4_t vuzp2_p16(poly16x4_t a, poly16x4_t b)	a -> Vn.4H b -> Vm.4H	UZP2 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
poly16x8_t vuzp2q_p16(poly16x8_t a, poly16x8_t b)	a -> Vn.8H b -> Vm.8H	UZP2 Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x8_t vuzp2_mf8(mfloat8x8_t a, mfloat8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP2 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vuzp2q_mf8(mfloat8x16_t a, mfloat8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UZP2 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int8x8x2_t vuzp_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP1 Vd1.8B,Vn.8B,Vm.8B UZP2 Vd2.8B,Vn.8B,Vm.8B</code>	<code>Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]</code>	v7/A32/A64
<code>int16x4x2_t vuzp_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP1 Vd1.4H,Vn.4H,Vm.4H UZP2 Vd2.4H,Vn.4H,Vm.4H</code>	<code>Vd1.4H -> result.val[0] Vd2.4H -> result.val[1]</code>	v7/A32/A64
<code>int32x2x2_t vuzp_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UZP1 Vd1.2S,Vn.2S,Vm.2S UZP2 Vd2.2S,Vn.2S,Vm.2S</code>	<code>Vd1.2S -> result.val[0] Vd2.2S -> result.val[1]</code>	v7/A32/A64
<code>float32x2x2_t vuzp_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UZP1 Vd1.2S,Vn.2S,Vm.2S UZP2 Vd2.2S,Vn.2S,Vm.2S</code>	<code>Vd1.2S -> result.val[0] Vd2.2S -> result.val[1]</code>	v7/A32/A64
<code>uint8x8x2_t vuzp_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP1 Vd1.8B,Vn.8B,Vm.8B UZP2 Vd2.8B,Vn.8B,Vm.8B</code>	<code>Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4x2_t vuzp_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP1 Vd1.4H,Vn.4H,Vm.4H UZP2 Vd2.4H,Vn.4H,Vm.4H</code>	<code>Vd1.4H -> result.val[0] Vd2.4H -> result.val[1]</code>	v7/A32/A64
<code>uint32x2x2_t vuzp_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>UZP1 Vd1.2S,Vn.2S,Vm.2S UZP2 Vd2.2S,Vn.2S,Vm.2S</code>	<code>Vd1.2S -> result.val[0] Vd2.2S -> result.val[1]</code>	v7/A32/A64
<code>poly8x8x2_t vuzp_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP1 Vd1.8B,Vn.8B,Vm.8B UZP2 Vd2.8B,Vn.8B,Vm.8B</code>	<code>Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]</code>	v7/A32/A64
<code>poly16x4x2_t vuzp_p16(poly16x4_t a, poly16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP1 Vd1.4H,Vn.4H,Vm.4H UZP2 Vd2.4H,Vn.4H,Vm.4H</code>	<code>Vd1.4H -> result.val[0] Vd2.4H -> result.val[1]</code>	v7/A32/A64
<code>mfloat8x8x2_t vuzp_mf8(mfloat8x8_t a, mfloat8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>UZP1 Vd1.8B,Vn.8B,Vm.8B UZP2 Vd2.8B,Vn.8B,Vm.8B</code>	<code>Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]</code>	A64
<code>int8x16x2_t vuzpq_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>UZP1 Vd1.16B,Vn.16B,Vm.16B UZP2 Vd2.16B,Vn.16B,Vm.16B</code>	<code>Vd1.16B -> result.val[0] Vd2.16B -> result.val[1]</code>	v7/A32/A64
<code>int16x8x2_t vuzpq_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UZP1 Vd1.8H,Vn.8H,Vm.8H UZP2 Vd2.8H,Vn.8H,Vm.8H</code>	<code>Vd1.8H -> result.val[0] Vd2.8H -> result.val[1]</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int32x4x2_t vuzpq_s32(int32x4_t a, int32x4_t b)	a -> Vn.4S b -> Vm.4S	UZP1 Vd1.4S,Vn.4S,Vm.4S UZP2 Vd2.4S,Vn.4S,Vm.4S	Vd1.4S -> result.val[0] Vd2.4S -> result.val[1]	v7/A32/A64
float32x4x2_t vuzpq_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	UZP1 Vd1.4S,Vn.4S,Vm.4S UZP2 Vd2.4S,Vn.4S,Vm.4S	Vd1.4S -> result.val[0] Vd2.4S -> result.val[1]	v7/A32/A64
uint8x16x2_t vuzpq_u8(uint8x16_t a, uint8x16_t b)	a -> Vn.16B b -> Vm.16B	UZP1 Vd1.16B,Vn.16B,Vm.16B UZP2 Vd2.16B,Vn.16B,Vm.16B	Vd1.16B -> result.val[0] Vd2.16B -> result.val[1]	v7/A32/A64
uint32x4x2_t vuzpq_u32(uint32x4_t a, uint32x4_t b)	a -> Vn.4S b -> Vm.4S	UZP1 Vd1.4S,Vn.4S,Vm.4S UZP2 Vd2.4S,Vn.4S,Vm.4S	Vd1.4S -> result.val[0] Vd2.4S -> result.val[1]	v7/A32/A64
poly8x16x2_t vuzpq_p8(poly8x16_t a, poly8x16_t b)	a -> Vn.16B b -> Vm.16B	UZP1 Vd1.16B,Vn.16B,Vm.16B UZP2 Vd2.16B,Vn.16B,Vm.16B	Vd1.16B -> result.val[0] Vd2.16B -> result.val[1]	v7/A32/A64
poly16x8x2_t vuzpq_p16(poly16x8_t a, poly16x8_t b)	a -> Vn.8H b -> Vm.8H	UZP1 Vd1.8H,Vn.8H,Vm.8H UZP2 Vd2.8H,Vn.8H,Vm.8H	Vd1.8H -> result.val[0] Vd2.8H -> result.val[1]	v7/A32/A64
mfloat8x16x2_t vuzpq_mf8(mfloat8x16_t a, mfloat8x16_t b)	a -> Vn.16B b -> Vm.16B	UZP1 Vd1.16B,Vn.16B,Vm.16B UZP2 Vd2.16B,Vn.16B,Vm.16B	Vd1.16B -> result.val[0] Vd2.16B -> result.val[1]	A64

2.1.9.12 Transpose elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vtrn1_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	TRN1 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
<code>int8x16_t vtrn1q_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	TRN1 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
<code>int16x4_t vtrn1_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	TRN1 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
<code>int16x8_t vtrn1q_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	TRN1 Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64
<code>int32x2_t vtrn1_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	TRN1 Vd.2S,Vn.2S,Vm.2S	Vd.2S -> result	A64
<code>int32x4_t vtrn1q_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	TRN1 Vd.4S,Vn.4S,Vm.4S	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2_t vtrn1q_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>TRN1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint8x8_t vtrn1_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>TRN1 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vtrn1q_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>TRN1 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x4_t vtrn1_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>TRN1 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>uint16x8_t vtrn1q_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>TRN1 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x2_t vtrn1_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>TRN1 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vtrn1q_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>TRN1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vtrn1q_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>TRN1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vtrn1q_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>TRN1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32x2_t vtrn1_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>TRN1 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vtrn1q_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>TRN1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>float64x2_t vtrn1q_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>TRN1 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>poly8x8_t vtrn1_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>TRN1 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>poly8x16_t vtrn1q_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>TRN1 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x4_t vtrn1_p16(poly16x4_t a, poly16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	TRN1 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
<code>poly16x8_t vtrn1q_p16(poly16x8_t a, poly16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	TRN1 Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64
<code>mfloating8x8_t vtrn1_mf8(mfloating8x8_t a, mfloating8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	TRN1 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
<code>mfloating8x16_t vtrn1q_mf8(mfloating8x16_t a, mfloating8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	TRN1 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
<code>int8x8_t vtrn2_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	TRN2 Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	A64
<code>int8x16_t vtrn2q_s8(int8x16_t a, int8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	TRN2 Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	A64
<code>int16x4_t vtrn2_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	TRN2 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vtrn2q_s16(int16x8_t a, int16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>TRN2 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>int32x2_t vtrn2_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>TRN2 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vtrn2q_s32(int32x4_t a, int32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>TRN2 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>int64x2_t vtrn2q_s64(int64x2_t a, int64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>TRN2 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>uint8x8_t vtrn2_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>TRN2 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vtrn2q_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>TRN2 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x4_t vtrn2_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>TRN2 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vtrn2q_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>TRN2 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>uint32x2_t vtrn2_u32(uint32x2_t a, uint32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>TRN2 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>uint32x4_t vtrn2q_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>TRN2 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint64x2_t vtrn2q_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>TRN2 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vtrn2q_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>TRN2 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>float32x2_t vtrn2_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>TRN2 Vd.2S,Vn.2S,Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>float32x4_t vtrn2q_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>TRN2 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2_t vtrn2q_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>TRN2 Vd.2D,Vn.2D,Vm.2D</code>	<code>Vd.2D -> result</code>	A64
<code>poly8x8_t vtrn2_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>TRN2 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>poly8x16_t vtrn2q_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>TRN2 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>poly16x4_t vtrn2_p16(poly16x4_t a, poly16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>TRN2 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>poly16x8_t vtrn2q_p16(poly16x8_t a, poly16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>TRN2 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>mfloat8x8_t vtrn2_mf8(mfloat8x8_t a, mfloat8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	<code>TRN2 Vd.8B,Vn.8B,Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vtrn2q_mf8(mfloat8x16_t a, mfloat8x16_t b)</code>	<code>a -> Vn.16B b -> Vm.16B</code>	<code>TRN2 Vd.16B,Vn.16B,Vm.16B</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8x2_t vtrn_s8(int8x8_t a, int8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	TRN1 <code>Vd1.8B, Vn.8B, Vm.8B</code> TRN2 <code>Vd2.8B, Vn.8B, Vm.8B</code>	<code>Vd1.8B -> result.val[0]</code> <code>Vd2.8B -> result.val[1]</code>	v7/A32/A64
<code>int16x4x2_t vtrn_s16(int16x4_t a, int16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	TRN1 <code>Vd1.4H, Vn.4H, Vm.4H</code> TRN2 <code>Vd2.4H, Vn.4H, Vm.4H</code>	<code>Vd1.4H -> result.val[0]</code> <code>Vd2.4H -> result.val[1]</code>	v7/A32/A64
<code>uint8x8x2_t vtrn_u8(uint8x8_t a, uint8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	TRN1 <code>Vd1.8B, Vn.8B, Vm.8B</code> TRN2 <code>Vd2.8B, Vn.8B, Vm.8B</code>	<code>Vd1.8B -> result.val[0]</code> <code>Vd2.8B -> result.val[1]</code>	v7/A32/A64
<code>uint16x4x2_t vtrn_u16(uint16x4_t a, uint16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	TRN1 <code>Vd1.4H, Vn.4H, Vm.4H</code> TRN2 <code>Vd2.4H, Vn.4H, Vm.4H</code>	<code>Vd1.4H -> result.val[0]</code> <code>Vd2.4H -> result.val[1]</code>	v7/A32/A64
<code>poly8x8x2_t vtrn_p8(poly8x8_t a, poly8x8_t b)</code>	<code>a -> Vn.8B b -> Vm.8B</code>	TRN1 <code>Vd1.8B, Vn.8B, Vm.8B</code> TRN2 <code>Vd2.8B, Vn.8B, Vm.8B</code>	<code>Vd1.8B -> result.val[0]</code> <code>Vd2.8B -> result.val[1]</code>	v7/A32/A64
<code>poly16x4x2_t vtrn_p16(poly16x4_t a, poly16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	TRN1 <code>Vd1.4H, Vn.4H, Vm.4H</code> TRN2 <code>Vd2.4H, Vn.4H, Vm.4H</code>	<code>Vd1.4H -> result.val[0]</code> <code>Vd2.4H -> result.val[1]</code>	v7/A32/A64
<code>int32x2x2_t vtrn_s32(int32x2_t a, int32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	TRN1 <code>Vd1.2S, Vn.2S, Vm.2S</code> TRN2 <code>Vd2.2S, Vn.2S, Vm.2S</code>	<code>Vd1.2S -> result.val[0]</code> <code>Vd2.2S -> result.val[1]</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2x2_t vtrn_f32(float32x2_t a, float32x2_t b)	a -> Vn.2S b -> Vm.2S	TRN1 Vd1.2S, Vn.2S, Vm.2S TRN2 Vd2.2S, Vn.2S, Vm.2S	Vd1.2S -> result.val[0] Vd2.2S -> result.val[1]	v7/A32/A64
uint32x2x2_t vtrn_u32(uint32x2_t a, uint32x2_t b)	a -> Vn.2S b -> Vm.2S	TRN1 Vd1.2S, Vn.2S, Vm.2S TRN2 Vd2.2S, Vn.2S, Vm.2S	Vd1.2S -> result.val[0] Vd2.2S -> result.val[1]	v7/A32/A64
mfloat8x8x2_t vtrn_mf8(mfloat8x8_t a, mfloat8x8_t b)	a -> Vn.8B b -> Vm.8B	TRN1 Vd1.8B, Vn.8B, Vm.8B TRN2 Vd2.8B, Vn.8B, Vm.8B	Vd1.8B -> result.val[0] Vd2.8B -> result.val[1]	A64
int8x16x2_t vtrnq_s8(int8x16_t a, int8x16_t b)	a -> Vn.16B b -> Vm.16B	TRN1 Vd1.16B, Vn.16B, Vm.16B TRN2 Vd2.16B, Vn.16B, Vm.16B	Vd1.16B -> result.val[0] Vd2.16B -> result.val[1]	v7/A32/A64
int16x8x2_t vtrnq_s16(int16x8_t a, int16x8_t b)	a -> Vn.8H b -> Vm.8H	TRN1 Vd1.8H, Vn.8H, Vm.8H TRN2 Vd2.8H, Vn.8H, Vm.8H	Vd1.8H -> result.val[0] Vd2.8H -> result.val[1]	v7/A32/A64
int32x4x2_t vtrnq_s32(int32x4_t a, int32x4_t b)	a -> Vn.4S b -> Vm.4S	TRN1 Vd1.4S, Vn.4S, Vm.4S TRN2 Vd2.4S, Vn.4S, Vm.4S	Vd1.4S -> result.val[0] Vd2.4S -> result.val[1]	v7/A32/A64
float32x4x2_t vtrnq_f32(float32x4_t a, float32x4_t b)	a -> Vn.4S b -> Vm.4S	TRN1 Vd1.4S, Vn.4S, Vm.4S TRN2 Vd2.4S, Vn.4S, Vm.4S	Vd1.4S -> result.val[0] Vd2.4S -> result.val[1]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16x2_t vtrnq_u8(uint8x16_t a, uint8x16_t b)</code>	<code>a -> Vn.16B</code> <code>b -> Vm.16B</code>	TRN1 <code>Vd1.16B, Vn.16B, Vm.16B</code> TRN2 <code>Vd2.16B, Vn.16B, Vm.16B</code>	<code>Vd1.16B -> result.val[0]</code> <code>Vd2.16B -> result.val[1]</code>	v7/A32/A64
<code>uint16x8x2_t vtrnq_u16(uint16x8_t a, uint16x8_t b)</code>	<code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	TRN1 <code>Vd1.8H, Vn.8H, Vm.8H</code> TRN2 <code>Vd2.8H, Vn.8H, Vm.8H</code>	<code>Vd1.8H -> result.val[0]</code> <code>Vd2.8H -> result.val[1]</code>	v7/A32/A64
<code>uint32x4x2_t vtrnq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S</code> <code>b -> Vm.4S</code>	TRN1 <code>Vd1.4S, Vn.4S, Vm.4S</code> TRN2 <code>Vd2.4S, Vn.4S, Vm.4S</code>	<code>Vd1.4S -> result.val[0]</code> <code>Vd2.4S -> result.val[1]</code>	v7/A32/A64
<code>poly8x16x2_t vtrnq_p8(poly8x16_t a, poly8x16_t b)</code>	<code>a -> Vn.16B</code> <code>b -> Vm.16B</code>	TRN1 <code>Vd1.16B, Vn.16B, Vm.16B</code> TRN2 <code>Vd2.16B, Vn.16B, Vm.16B</code>	<code>Vd1.16B -> result.val[0]</code> <code>Vd2.16B -> result.val[1]</code>	v7/A32/A64
<code>poly16x8x2_t vtrnq_p16(poly16x8_t a, poly16x8_t b)</code>	<code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	TRN1 <code>Vd1.8H, Vn.8H, Vm.8H</code> TRN2 <code>Vd2.8H, Vn.8H, Vm.8H</code>	<code>Vd1.8H -> result.val[0]</code> <code>Vd2.8H -> result.val[1]</code>	v7/A32/A64
<code>mfloat8x16x2_t vtrnq_mf8(mfloat8x16_t a, mfloat8x16_t b)</code>	<code>a -> Vn.16B</code> <code>b -> Vm.16B</code>	TRN1 <code>Vd1.16B, Vn.16B, Vm.16B</code> TRN2 <code>Vd2.16B, Vn.16B, Vm.16B</code>	<code>Vd1.16B -> result.val[0]</code> <code>Vd2.16B -> result.val[1]</code>	A64

2.1.9.13 Set vector lane

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vset_lane_u8(uint8_t a, uint8x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ a -> Rn v -> Vd.8B	MOV Vd.B[lane],Rn	Vd.8B -> result	v7/A32/A64
<code>uint16x4_t vset_lane_u16(uint16_t a, uint16x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ a -> Rn v -> Vd.4H	MOV Vd.H[lane],Rn	Vd.4H -> result	v7/A32/A64
<code>uint32x2_t vset_lane_u32(uint32_t a, uint32x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ a -> Rn v -> Vd.2S	MOV Vd.S[lane],Rn	Vd.2S -> result	v7/A32/A64
<code>uint64x1_t vset_lane_u64(uint64_t a, uint64x1_t v, const int lane)</code>	lane==0 a -> Rn v -> Vd.1D	MOV Vd.D[lane],Rn	Vd.1D -> result	v7/A32/A64
<code>poly64x1_t vset_lane_p64(poly64_t a, poly64x1_t v, const int lane)</code>	lane==0 a -> Rn v -> Vd.1D	MOV Vd.D[lane],Rn	Vd.1D -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vset_lane_s8(int8_t a, int8x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ a -> Rn v -> Vd.8B	MOV Vd.B[lane],Rn	Vd.8B -> result	v7/A32/A64
<code>int16x4_t vset_lane_s16(int16_t a, int16x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ a -> Rn v -> Vd.4H	MOV Vd.H[lane],Rn	Vd.4H -> result	v7/A32/A64
<code>int32x2_t vset_lane_s32(int32_t a, int32x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ a -> Rn v -> Vd.2S	MOV Vd.S[lane],Rn	Vd.2S -> result	v7/A32/A64
<code>int64x1_t vset_lane_s64(int64_t a, int64x1_t v, const int lane)</code>	lane==0 a -> Rn v -> Vd.1D	MOV Vd.D[lane],Rn	Vd.1D -> result	v7/A32/A64
<code>poly8x8_t vset_lane_p8(poly8_t a, poly8x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ a -> Rn v -> Vd.8B	MOV Vd.B[lane],Rn	Vd.8B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly16x4_t vset_lane_p16(poly16_t a, poly16x4_t v, const int lane)	0<=lane<=3 a -> Rn v -> Vd.4H	MOV Vd.H[lane],Rn	Vd.4H -> result	v7/A32/A64
float16x4_t vset_lane_f16(float16_t a, float16x4_t v, const int lane)	0<=lane<=3 a -> VnH v -> Vd.4H	MOV Vd.H[lane],Vn.H[0]	Vd.4H -> result	v7/A32/A64
float16x8_t vsetq_lane_f16(float16_t a, float16x8_t v, const int lane)	0<=lane<=7 a -> VnH v -> Vd.8H	MOV Vd.H[lane],Vn.H[0]	Vd.8H -> result	v7/A32/A64
float32x2_t vset_lane_f32(float32_t a, float32x2_t v, const int lane)	0<=lane<=1 a -> Rn v -> Vd.2S	MOV Vd.S[lane],Rn	Vd.2S -> result	v7/A32/A64
float64x1_t vset_lane_f64(float64_t a, float64x1_t v, const int lane)	lane==0 a -> Rn v -> Vd.1D	MOV Vd.D[lane],Rn	Vd.1D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x8_t vset_lane_mf8(mfloat8_t a, mfloat8x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ a -> Rn v -> Vd.8B	MOV Vd.B[lane],Rn	Vd.8B -> result	A64
<code>uint8x16_t vsetq_lane_u8(uint8_t a, uint8x16_t v, const int lane)</code>	$0 \leq \text{lane} \leq 15$ a -> Rn v -> Vd.16B	MOV Vd.B[lane],Rn	Vd.16B -> result	v7/A32/A64
<code>uint16x8_t vsetq_lane_u16(uint16_t a, uint16x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ a -> Rn v -> Vd.8H	MOV Vd.H[lane],Rn	Vd.8H -> result	v7/A32/A64
<code>uint32x4_t vsetq_lane_u32(uint32_t a, uint32x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ a -> Rn v -> Vd.4S	MOV Vd.S[lane],Rn	Vd.4S -> result	v7/A32/A64
<code>uint64x2_t vsetq_lane_u64(uint64_t a, uint64x2_t v, const int lane)</code>	$0 \leq \text{lane} \leq 1$ a -> Rn v -> Vd.2D	MOV Vd.D[lane],Rn	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x2_t vsetq_lane_p64(poly64_t a, poly64x2_t v, const int lane)	0<=lane<=1 a -> Rn v -> Vd.2D	MOV Vd.D[lane],Rn	Vd.2D -> result	A32/A64
int8x16_t vsetq_lane_s8(int8_t a, int8x16_t v, const int lane)	0<=lane<=15 a -> Rn v -> Vd.16B	MOV Vd.B[lane],Rn	Vd.16B -> result	v7/A32/A64
int16x8_t vsetq_lane_s16(int16_t a, int16x8_t v, const int lane)	0<=lane<=7 a -> Rn v -> Vd.8H	MOV Vd.H[lane],Rn	Vd.8H -> result	v7/A32/A64
int32x4_t vsetq_lane_s32(int32_t a, int32x4_t v, const int lane)	0<=lane<=3 a -> Rn v -> Vd.4S	MOV Vd.S[lane],Rn	Vd.4S -> result	v7/A32/A64
int64x2_t vsetq_lane_s64(int64_t a, int64x2_t v, const int lane)	0<=lane<=1 a -> Rn v -> Vd.2D	MOV Vd.D[lane],Rn	Vd.2D -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x16_t vsetq_lane_p8(poly8_t a, poly8x16_t v, const int lane)	0<=lane<=15 a -> Rn v -> Vd.16B	MOV Vd.B[lane],Rn	Vd.16B -> result	v7/A32/A64
poly16x8_t vsetq_lane_p16(poly16_t a, poly16x8_t v, const int lane)	0<=lane<=7 a -> Rn v -> Vd.8H	MOV Vd.H[lane],Rn	Vd.8H -> result	v7/A32/A64
float32x4_t vsetq_lane_f32(float32_t a, float32x4_t v, const int lane)	0<=lane<=3 a -> Rn v -> Vd.4S	MOV Vd.S[lane],Rn	Vd.4S -> result	v7/A32/A64
float64x2_t vsetq_lane_f64(float64_t a, float64x2_t v, const int lane)	0<=lane<=1 a -> Rn v -> Vd.2D	MOV Vd.D[lane],Rn	Vd.2D -> result	A64
mfloating8x16_t vsetq_lane_mf8(mfloating8_t a, mfloating8x16_t v, const int lane)	0<=lane<=15 a -> Rn v -> Vd.16B	MOV Vd.B[lane],Rn	Vd.16B -> result	A64

2.1.9.14 Unzip elements'

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x8x2_t vuzpq_u16(uint16x8_t a, uint16x8_t b)	a -> Vn.8H b -> Vm.8H	UZP1 Vd1.8H,Vn.8H,Vm.8H UZP2 Vd2.8H,Vn.8H,Vm.8H	Vd1.8H -> result.val[0] Vd2.8H -> result.val[1]	v7/A32/A64

2.1.10 Load

2.1.10.1 Stride

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int8x8_t vld1_s8(int8_t const *ptr)	ptr -> Xn	LD1 {Vt.8B},[Xn]	Vt.8B -> result	v7/A32/A64
int8x16_t vld1q_s8(int8_t const *ptr)	ptr -> Xn	LD1 {Vt.16B},[Xn]	Vt.16B -> result	v7/A32/A64
int16x4_t vld1_s16(int16_t const *ptr)	ptr -> Xn	LD1 {Vt.4H},[Xn]	Vt.4H -> result	v7/A32/A64
int16x8_t vld1q_s16(int16_t const *ptr)	ptr -> Xn	LD1 {Vt.8H},[Xn]	Vt.8H -> result	v7/A32/A64
int32x2_t vld1_s32(int32_t const *ptr)	ptr -> Xn	LD1 {Vt.2S},[Xn]	Vt.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vld1q_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S},[Xn]</code>	<code>Vt.4S -> result</code>	v7/A32/A64
<code>int64x1_t vld1_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D},[Xn]</code>	<code>Vt.1D -> result</code>	v7/A32/A64
<code>int64x2_t vld1q_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D},[Xn]</code>	<code>Vt.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vld1_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B},[Xn]</code>	<code>Vt.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vld1q_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B},[Xn]</code>	<code>Vt.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vld1_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H},[Xn]</code>	<code>Vt.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vld1q_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H},[Xn]</code>	<code>Vt.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vld1_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S},[Xn]</code>	<code>Vt.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vld1q_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S},[Xn]</code>	<code>Vt.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x1_t vld1_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D}, [Xn]</code>	<code>Vt.1D -> result</code>	v7/A32/A64
<code>uint64x2_t vld1q_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D}, [Xn]</code>	<code>Vt.2D -> result</code>	v7/A32/A64
<code>poly64x1_t vld1_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D}, [Xn]</code>	<code>Vt.1D -> result</code>	A32/A64
<code>poly64x2_t vld1q_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D}, [Xn]</code>	<code>Vt.2D -> result</code>	A32/A64
<code>float16x4_t vld1_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H}, [Xn]</code>	<code>Vt.4H -> result</code>	v7/A32/A64
<code>float16x8_t vld1q_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H}, [Xn]</code>	<code>Vt.8H -> result</code>	v7/A32/A64
<code>float32x2_t vld1_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S}, [Xn]</code>	<code>Vt.2S -> result</code>	v7/A32/A64
<code>float32x4_t vld1q_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S}, [Xn]</code>	<code>Vt.4S -> result</code>	v7/A32/A64
<code>poly8x8_t vld1_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B}, [Xn]</code>	<code>Vt.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x16_t vld1q_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B},[Xn]</code>	<code>Vt.16B -> result</code>	v7/A32/A64
<code>poly16x4_t vld1_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H},[Xn]</code>	<code>Vt.4H -> result</code>	v7/A32/A64
<code>poly16x8_t vld1q_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H},[Xn]</code>	<code>Vt.8H -> result</code>	v7/A32/A64
<code>float64x1_t vld1_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D},[Xn]</code>	<code>Vt.1D -> result</code>	A64
<code>float64x2_t vld1q_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D},[Xn]</code>	<code>Vt.2D -> result</code>	A64
<code>mfloat8x8_t vld1_mf8(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B},[Xn]</code>	<code>Vt.8B -> result</code>	A64
<code>mfloat8x16_t vld1q_mf8(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B},[Xn]</code>	<code>Vt.16B -> result</code>	A64
<code>int8x8_t vld1_lane_s8(int8_t const *ptr, int8x8_t src, const int lane)</code>	<code>ptr -> Xn</code> <code>src -> Vt.8B</code> <code>0 <= lane <= 7</code>	<code>LD1 {Vt.b}[lane],[Xn]</code>	<code>Vt.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vld1q_lane_s8(int8_t const *ptr, int8x16_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.16B 0 <= lane <= 15</code>	<code>LD1 {Vt.b}[lane],[Xn]</code>	<code>Vt.16B -> result</code>	v7/A32/A64
<code>int16x4_t vld1_lane_s16(int16_t const *ptr, int16x4_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.4H 0 <= lane <= 3</code>	<code>LD1 {Vt.H}[lane],[Xn]</code>	<code>Vt.4H -> result</code>	v7/A32/A64
<code>int16x8_t vld1q_lane_s16(int16_t const *ptr, int16x8_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.8H 0 <= lane <= 7</code>	<code>LD1 {Vt.H}[lane],[Xn]</code>	<code>Vt.8H -> result</code>	v7/A32/A64
<code>int32x2_t vld1_lane_s32(int32_t const *ptr, int32x2_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.2S 0 <= lane <= 1</code>	<code>LD1 {Vt.S}[lane],[Xn]</code>	<code>Vt.2S -> result</code>	v7/A32/A64
<code>int32x4_t vld1q_lane_s32(int32_t const *ptr, int32x4_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.4S 0 <= lane <= 3</code>	<code>LD1 {Vt.S}[lane],[Xn]</code>	<code>Vt.4S -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1_t vld1_lane_s64(int64_t const *ptr, int64x1_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.1D 0 <= lane <= 0</code>	<code>LD1 {Vt.D}[lane],[Xn]</code>	<code>Vt.1D -> result</code>	v7/A32/A64
<code>int64x2_t vld1q_lane_s64(int64_t const *ptr, int64x2_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.2D 0 <= lane <= 1</code>	<code>LD1 {Vt.D}[lane],[Xn]</code>	<code>Vt.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vld1_lane_u8(uint8_t const *ptr, uint8x8_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.8B 0 <= lane <= 7</code>	<code>LD1 {Vt.B}[lane],[Xn]</code>	<code>Vt.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vld1q_lane_u8(uint8_t const *ptr, uint8x16_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.16B 0 <= lane <= 15</code>	<code>LD1 {Vt.B}[lane],[Xn]</code>	<code>Vt.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vld1_lane_u16(uint16_t const *ptr, uint16x4_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.4H 0 <= lane <= 3</code>	<code>LD1 {Vt.H}[lane],[Xn]</code>	<code>Vt.4H -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vld1q_lane_u16(uint16_t const *ptr, uint16x8_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.8H 0 <= lane <= 7</code>	<code>LD1 {Vt.H}[lane],[Xn]</code>	<code>Vt.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vld1_lane_u32(uint32_t const *ptr, uint32x2_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.2S 0 <= lane <= 1</code>	<code>LD1 {Vt.S}[lane],[Xn]</code>	<code>Vt.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vld1q_lane_u32(uint32_t const *ptr, uint32x4_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.4S 0 <= lane <= 3</code>	<code>LD1 {Vt.S}[lane],[Xn]</code>	<code>Vt.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vld1_lane_u64(uint64_t const *ptr, uint64x1_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.1D 0 <= lane <= 0</code>	<code>LD1 {Vt.D}[lane],[Xn]</code>	<code>Vt.1D -> result</code>	v7/A32/A64
<code>uint64x2_t vld1q_lane_u64(uint64_t const *ptr, uint64x2_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.2D 0 <= lane <= 1</code>	<code>LD1 {Vt.D}[lane],[Xn]</code>	<code>Vt.2D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x1_t vld1_lane_p64(poly64_t const *ptr, poly64x1_t src, const int lane)	ptr -> Xn src -> Vt.1D 0 <= lane <= 0	LD1 {Vt.D}[lane],[Xn]	Vt.1D -> result	A32/A64
poly64x2_t vld1q_lane_p64(poly64_t const *ptr, poly64x2_t src, const int lane)	ptr -> Xn src -> Vt.2D 0 <= lane <= 1	LD1 {Vt.D}[lane],[Xn]	Vt.2D -> result	A32/A64
float16x4_t vld1_lane_f16(float16_t const *ptr, float16x4_t src, const int lane)	ptr -> Xn src -> Vt.4H 0 <= lane <= 3	LD1 {Vt.H}[lane],[Xn]	Vt.4H -> result	v7/A32/A64
float16x8_t vld1q_lane_f16(float16_t const *ptr, float16x8_t src, const int lane)	ptr -> Xn src -> Vt.8H 0 <= lane <= 7	LD1 {Vt.H}[lane],[Xn]	Vt.8H -> result	v7/A32/A64
float32x2_t vld1_lane_f32(float32_t const *ptr, float32x2_t src, const int lane)	ptr -> Xn src -> Vt.2S 0 <= lane <= 1	LD1 {Vt.S}[lane],[Xn]	Vt.2S -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vld1q_lane_f32(float32_t const *ptr, float32x4_t src, const int lane)	ptr -> Xn src -> Vt.4S 0 <= lane <= 3	LD1 {Vt.S}[lane],[Xn]	Vt.4S -> result	v7/A32/A64
poly8x8_t vld1_lane_p8(poly8_t const *ptr, poly8x8_t src, const int lane)	ptr -> Xn src -> Vt.8B 0 <= lane <= 7	LD1 {Vt.B}[lane],[Xn]	Vt.8B -> result	v7/A32/A64
poly8x16_t vld1q_lane_p8(poly8_t const *ptr, poly8x16_t src, const int lane)	ptr -> Xn src -> Vt.16B 0 <= lane <= 15	LD1 {Vt.B}[lane],[Xn]	Vt.16B -> result	v7/A32/A64
poly16x4_t vld1_lane_p16(poly16_t const *ptr, poly16x4_t src, const int lane)	ptr -> Xn src -> Vt.4H 0 <= lane <= 3	LD1 {Vt.H}[lane],[Xn]	Vt.4H -> result	v7/A32/A64
poly16x8_t vld1q_lane_p16(poly16_t const *ptr, poly16x8_t src, const int lane)	ptr -> Xn src -> Vt.8H 0 <= lane <= 7	LD1 {Vt.H}[lane],[Xn]	Vt.8H -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x1_t vld1_lane_f64(float64_t const *ptr, float64x1_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.1D 0 <= lane <= 0</code>	<code>LD1 {Vt.D}[lane],[Xn]</code>	<code>Vt.1D -> result</code>	A64
<code>float64x2_t vld1q_lane_f64(float64_t const *ptr, float64x2_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.2D 0 <= lane <= 1</code>	<code>LD1 {Vt.D}[lane],[Xn]</code>	<code>Vt.2D -> result</code>	A64
<code>mfloat8x8_t vld1_lane_mf8(mfloat8_t const *ptr, mfloat8x8_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.8B 0 <= lane <= 7</code>	<code>LD1 {Vt.b}[lane],[Xn]</code>	<code>Vt.8B -> result</code>	A64
<code>mfloat8x16_t vld1q_lane_mf8(mfloat8_t const *ptr, mfloat8x16_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.16B 0 <= lane <= 15</code>	<code>LD1 {Vt.b}[lane],[Xn]</code>	<code>Vt.16B -> result</code>	A64
<code>uint64x1_t vldap1_lane_u64(uint64_t const *ptr, uint64x1_t src, const int lane)</code>	<code>ptr -> Xn src -> Vt.1D 0 <= lane <= 0</code>	<code>LDAP1 {Vt.D}[lane],[Xn]</code>	<code>Vt.1D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint64x2_t vldap1q_lane_u64(uint64_t const *ptr, uint64x2_t src, const int lane)	ptr -> Xn src -> Vt.2D 0 <= lane <= 1	LDAP1 {Vt.D}[lane],[Xn]	Vt.2D -> result	A64
int64x1_t vldap1_lane_s64(int64_t const *ptr, int64x1_t src, const int lane)	ptr -> Xn src -> Vt.1D 0 <= lane <= 0	LDAP1 {Vt.D}[lane],[Xn]	Vt.1D -> result	A64
int64x2_t vldap1q_lane_s64(int64_t const *ptr, int64x2_t src, const int lane)	ptr -> Xn src -> Vt.2D 0 <= lane <= 1	LDAP1 {Vt.D}[lane],[Xn]	Vt.2D -> result	A64
float64x1_t vldap1_lane_f64(float64_t const *ptr, float64x1_t src, const int lane)	ptr -> Xn src -> Vt.1D 0 <= lane <= 0	LDAP1 {Vt.D}[lane],[Xn]	Vt.1D -> result	A64
float64x2_t vldap1q_lane_f64(float64_t const *ptr, float64x2_t src, const int lane)	ptr -> Xn src -> Vt.2D 0 <= lane <= 1	LDAP1 {Vt.D}[lane],[Xn]	Vt.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x1_t vldap1_lane_p64(poly64_t const *ptr, poly64x1_t src, const int lane)	ptr -> Xn src -> Vt.1D 0 <= lane <= 0	LDAP1 {Vt.D}[lane],[Xn]	Vt.1D -> result	A64
poly64x2_t vldap1q_lane_p64(poly64_t const *ptr, poly64x2_t src, const int lane)	ptr -> Xn src -> Vt.2D 0 <= lane <= 1	LDAP1 {Vt.D}[lane],[Xn]	Vt.2D -> result	A64
int8x8_t vld1_dup_s8(int8_t const *ptr)	ptr -> Xn	LD1R {Vt.8B},[Xn]	Vt.8B -> result	v7/A32/A64
int8x16_t vld1q_dup_s8(int8_t const *ptr)	ptr -> Xn	LD1R {Vt.16B},[Xn]	Vt.16B -> result	v7/A32/A64
int16x4_t vld1_dup_s16(int16_t const *ptr)	ptr -> Xn	LD1R {Vt.4H},[Xn]	Vt.4H -> result	v7/A32/A64
int16x8_t vld1q_dup_s16(int16_t const *ptr)	ptr -> Xn	LD1R {Vt.8H},[Xn]	Vt.8H -> result	v7/A32/A64
int32x2_t vld1_dup_s32(int32_t const *ptr)	ptr -> Xn	LD1R {Vt.2S},[Xn]	Vt.2S -> result	v7/A32/A64
int32x4_t vld1q_dup_s32(int32_t const *ptr)	ptr -> Xn	LD1R {Vt.4S},[Xn]	Vt.4S -> result	v7/A32/A64

Copyright: see section [Copyright](#).

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1_t vld1_dup_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D},[Xn]</code>	<code>Vt.1D -> result</code>	v7/A32/A64
<code>int64x2_t vld1q_dup_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.2D},[Xn]</code>	<code>Vt.2D -> result</code>	v7/A32/A64
<code>uint8x8_t vld1_dup_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.8B},[Xn]</code>	<code>Vt.8B -> result</code>	v7/A32/A64
<code>uint8x16_t vld1q_dup_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.16B},[Xn]</code>	<code>Vt.16B -> result</code>	v7/A32/A64
<code>uint16x4_t vld1_dup_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.4H},[Xn]</code>	<code>Vt.4H -> result</code>	v7/A32/A64
<code>uint16x8_t vld1q_dup_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.8H},[Xn]</code>	<code>Vt.8H -> result</code>	v7/A32/A64
<code>uint32x2_t vld1_dup_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.2S},[Xn]</code>	<code>Vt.2S -> result</code>	v7/A32/A64
<code>uint32x4_t vld1q_dup_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.4S},[Xn]</code>	<code>Vt.4S -> result</code>	v7/A32/A64
<code>uint64x1_t vld1_dup_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D},[Xn]</code>	<code>Vt.1D -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vld1q_dup_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.2D},[Xn]</code>	<code>Vt.2D -> result</code>	v7/A32/A64
<code>poly64x1_t vld1_dup_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D},[Xn]</code>	<code>Vt.1D -> result</code>	A32/A64
<code>poly64x2_t vld1q_dup_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.2D},[Xn]</code>	<code>Vt.2D -> result</code>	A32/A64
<code>float16x4_t vld1_dup_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.4H},[Xn]</code>	<code>Vt.4H -> result</code>	v7/A32/A64
<code>float16x8_t vld1q_dup_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.8H},[Xn]</code>	<code>Vt.8H -> result</code>	v7/A32/A64
<code>float32x2_t vld1_dup_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.2S},[Xn]</code>	<code>Vt.2S -> result</code>	v7/A32/A64
<code>float32x4_t vld1q_dup_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.4S},[Xn]</code>	<code>Vt.4S -> result</code>	v7/A32/A64
<code>poly8x8_t vld1_dup_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.8B},[Xn]</code>	<code>Vt.8B -> result</code>	v7/A32/A64
<code>poly8x16_t vld1q_dup_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.16B},[Xn]</code>	<code>Vt.16B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x4_t vld1_dup_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.4H},[Xn]</code>	<code>Vt.4H -> result</code>	v7/A32/A64
<code>poly16x8_t vld1q_dup_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.8H},[Xn]</code>	<code>Vt.8H -> result</code>	v7/A32/A64
<code>float64x1_t vld1_dup_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D},[Xn]</code>	<code>Vt.1D -> result</code>	A64
<code>float64x2_t vld1q_dup_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.2D},[Xn]</code>	<code>Vt.2D -> result</code>	A64
<code>mfloat8x8_t vld1_dup_mf8(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.8B},[Xn]</code>	<code>Vt.8B -> result</code>	A64
<code>mfloat8x16_t vld1q_dup_mf8(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.16B},[Xn]</code>	<code>Vt.16B -> result</code>	A64
<code>void vstl1_lane_u64(uint64_t *ptr, uint64x1_t val, const int lane)</code>	<code>val -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	<code>STL1 {Vt.d}[lane],[Xn]</code>		A64
<code>void vstl1q_lane_u64(uint64_t *ptr, uint64x2_t val, const int lane)</code>	<code>val -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	<code>STL1 {Vt.d}[lane],[Xn]</code>		A64

Copyright: see section [Copyright](#).

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vstl1_lane_s64(int64_t *ptr, int64x1_t val, const int lane)</pre>	<pre>val -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	STL1 {Vt.d}[lane],[Xn]		A64
<pre>void vstl1q_lane_s64(int64_t *ptr, int64x2_t val, const int lane)</pre>	<pre>val -> Vt.2D ptr -> Xn 0 <= lane <= 1</pre>	STL1 {Vt.d}[lane],[Xn]		A64
<pre>void vstl1_lane_f64(float64_t *ptr, float64x1_t val, const int lane)</pre>	<pre>val -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	STL1 {Vt.d}[lane],[Xn]		A64
<pre>void vstl1q_lane_f64(float64_t *ptr, float64x2_t val, const int lane)</pre>	<pre>val -> Vt.2D ptr -> Xn 0 <= lane <= 1</pre>	STL1 {Vt.d}[lane],[Xn]		A64
<pre>void vstl1_lane_p64(poly64_t *ptr, poly64x1_t val, const int lane)</pre>	<pre>val -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	STL1 {Vt.d}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_lane_p64(poly64_t *ptr, poly64x2_t val, const int lane)</code>	<code>val -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	STL1 {Vt.d}[lane],[Xn]		A64
<code>int8x8x2_t vld2_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	LD2 {Vt.8B - Vt2.8B},[Xn]	Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64
<code>int8x16x2_t vld2q_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	LD2 {Vt.16B - Vt2.16B},[Xn]	Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A32/A64
<code>int16x4x2_t vld2_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	LD2 {Vt.4H - Vt2.4H},[Xn]	Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>int16x8x2_t vld2q_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	LD2 {Vt.8H - Vt2.8H},[Xn]	Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
<code>int32x2x2_t vld2_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	LD2 {Vt.2S - Vt2.2S},[Xn]	Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4x2_t vld2q_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.4S - Vt2.4S}, [Xn]</code>	<code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64
<code>uint8x8x2_t vld2_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.8B - Vt2.8B}, [Xn]</code>	<code>Vt2.8B -> result.val[1]</code> <code>Vt.8B -> result.val[0]</code>	A32/A64
<code>uint8x16x2_t vld2q_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.16B - Vt2.16B}, [Xn]</code>	<code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>uint16x4x2_t vld2_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.4H - Vt2.4H}, [Xn]</code>	<code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>uint16x8x2_t vld2q_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>uint32x2x2_t vld2_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.2S - Vt2.2S}, [Xn]</code>	<code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>uint32x4x2_t vld2q_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.4S - Vt2.4S}, [Xn]</code>	<code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4x2_t vld2_f16(float16_t const *ptr)	ptr -> Xn	LD2 {Vt.4H - Vt2.4H}, [Xn]	Vt2.4H -> result.val[1]7/A32/A64 Vt.4H -> result.val[0]	
float16x8x2_t vld2q_f16(float16_t const *ptr)	ptr -> Xn	LD2 {Vt.8H - Vt2.8H}, [Xn]	Vt2.8H -> result.val[1]7/A32/A64 Vt.8H -> result.val[0]	
float32x2x2_t vld2_f32(float32_t const *ptr)	ptr -> Xn	LD2 {Vt.2S - Vt2.2S}, [Xn]	Vt2.2S -> result.val[1]7/A32/A64 Vt.2S -> result.val[0]	
float32x4x2_t vld2q_f32(float32_t const *ptr)	ptr -> Xn	LD2 {Vt.4S - Vt2.4S}, [Xn]	Vt2.4S -> result.val[1]7/A32/A64 Vt.4S -> result.val[0]	
poly8x8x2_t vld2_p8(poly8_t const *ptr)	ptr -> Xn	LD2 {Vt.8B - Vt2.8B}, [Xn]	Vt2.8B -> result.val[1]7/A32/A64 Vt.8B -> result.val[0]	
poly8x16x2_t vld2q_p8(poly8_t const *ptr)	ptr -> Xn	LD2 {Vt.16B - Vt2.16B}, [Xn]	Vt2.16B -> result.val[1]7/A32/A64 Vt.16B -> result.val[0]	
poly16x4x2_t vld2_p16(poly16_t const *ptr)	ptr -> Xn	LD2 {Vt.4H - Vt2.4H}, [Xn]	Vt2.4H -> result.val[1]7/A32/A64 Vt.4H -> result.val[0]	

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x8x2_t vld2q_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>int64x1x2_t vld2_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>uint64x1x2_t vld2_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>poly64x1x2_t vld2_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>int64x2x2_t vld2q_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A64
<code>uint64x2x2_t vld2q_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A64
<code>poly64x2x2_t vld2q_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x1x2_t vld2_f64(float64_t const *ptr)	ptr -> Xn	LD1 {Vt.1D - Vt2.1D}, [Xn]	Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64
float64x2x2_t vld2q_f64(float64_t const *ptr)	ptr -> Xn	LD2 {Vt.2D - Vt2.2D}, [Xn]	Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
mfloat8x8x2_t vld2_mf8(mfloat8_t const *ptr)	ptr -> Xn	LD2 {Vt.8B - Vt2.8B}, [Xn]	Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
mfloat8x16x2_t vld2q_mf8(mfloat8_t const *ptr)	ptr -> Xn	LD2 {Vt.16B - Vt2.16B}, [Xn]	Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64
int8x8x3_t vld3_s8(int8_t const *ptr)	ptr -> Xn	LD3 {Vt.8B - Vt3.8B}, [Xn]	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64
int8x16x3_t vld3q_s8(int8_t const *ptr)	ptr -> Xn	LD3 {Vt.16B - Vt3.16B}, [Xn]	Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4x3_t vld3_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
<code>int16x8x3_t vld3q_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
<code>int32x2x3_t vld3_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.2S - Vt3.2S}, [Xn]</code>	Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64
<code>int32x4x3_t vld3q_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.4S - Vt3.4S}, [Xn]</code>	Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
<code>uint8x8x3_t vld3_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16x3_t vld3q_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.16B - Vt3.16B}, [Xn]</code>	<code>Vt3.16B -> result.val[2]</code> <code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>uint16x4x3_t vld3_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.4H - Vt3.4H}, [Xn]</code>	<code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>uint16x8x3_t vld3q_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.8H - Vt3.8H}, [Xn]</code>	<code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>uint32x2x3_t vld3_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.2S - Vt3.2S}, [Xn]</code>	<code>Vt3.2S -> result.val[2]</code> <code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>uint32x4x3_t vld3q_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.4S - Vt3.4S}, [Xn]</code>	<code>Vt3.4S -> result.val[2]</code> <code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4x3_t vld3_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.4H - Vt3.4H}, [Xn]</code>	<code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	v7/A32/A64
<code>float16x8x3_t vld3q_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.8H - Vt3.8H}, [Xn]</code>	<code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	v7/A32/A64
<code>float32x2x3_t vld3_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.2S - Vt3.2S}, [Xn]</code>	<code>Vt3.2S -> result.val[2]</code> <code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	v7/A32/A64
<code>float32x4x3_t vld3q_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.4S - Vt3.4S}, [Xn]</code>	<code>Vt3.4S -> result.val[2]</code> <code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	v7/A32/A64
<code>poly8x8x3_t vld3_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.8B - Vt3.8B}, [Xn]</code>	<code>Vt3.8B -> result.val[2]</code> <code>Vt2.8B -> result.val[1]</code> <code>Vt.8B -> result.val[0]</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x16x3_t vld3q_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.16B - Vt3.16B}, [Xn]</code>	<code>Vt3.16B -> result.val[2]</code> <code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>poly16x4x3_t vld3_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.4H - Vt3.4H}, [Xn]</code>	<code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>poly16x8x3_t vld3q_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.8H - Vt3.8H}, [Xn]</code>	<code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>int64x1x3_t vld3_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt3.1D}, [Xn]</code>	<code>Vt3.1D -> result.val[2]</code> <code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>uint64x1x3_t vld3_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt3.1D}, [Xn]</code>	<code>Vt3.1D -> result.val[2]</code> <code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly64x1x3_t vld3_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt3.1D}, [Xn]</code>	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A32/A64
<code>int64x2x3_t vld3q_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>uint64x2x3_t vld3q_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>poly64x2x3_t vld3q_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>float64x1x3_t vld3_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt3.1D}, [Xn]</code>	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2x3_t vld3q_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>mfloating8x8x3_t vld3_mf8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
<code>mfloating8x16x3_t vld3q_mf8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.16B - Vt3.16B}, [Xn]</code>	Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64
<code>int8x8x4_t vld4_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A7/A32/A64
<code>int8x16x4_t vld4q_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.16B - Vt4.16B}, [Xn]</code>	Vt4.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4x4_t vld4_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
<code>int16x8x4_t vld4q_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
<code>int32x2x4_t vld4_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.2S - Vt4.2S}, [Xn]</code>	Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64
<code>int32x4x4_t vld4q_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.4S - Vt4.4S}, [Xn]</code>	Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
<code>uint8x8x4_t vld4_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16x4_t vld4q_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.16B - Vt4.16B}, [Xn]</code>	<code>Vt4.16B -> result.val[3]</code> <code>Vt3.16B -> result.val[2]</code> <code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>uint16x4x4_t vld4_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.4H - Vt4.4H}, [Xn]</code>	<code>Vt4.4H -> result.val[3]</code> <code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>uint16x8x4_t vld4q_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8H - Vt4.8H}, [Xn]</code>	<code>Vt4.8H -> result.val[3]</code> <code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>uint32x2x4_t vld4_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.2S - Vt4.2S}, [Xn]</code>	<code>Vt4.2S -> result.val[3]</code> <code>Vt3.2S -> result.val[2]</code> <code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>uint32x4x4_t vld4q_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.4S - Vt4.4S}, [Xn]</code>	<code>Vt4.4S -> result.val[3]</code> <code>Vt3.4S -> result.val[2]</code> <code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4x4_t vld4_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>float16x8x4_t vld4q_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
<code>float32x2x4_t vld4_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.2S - Vt4.2S}, [Xn]</code>	Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	A32/A64
<code>float32x4x4_t vld4q_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.4S - Vt4.4S}, [Xn]</code>	Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	A32/A64
<code>poly8x8x4_t vld4_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x16x4_t vld4q_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.16B - Vt4.16B}, [Xn]</code>	<code>Vt4.16B -> result.val[3]</code> <code>Vt3.16B -> result.val[2]</code> <code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>poly16x4x4_t vld4_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.4H - Vt4.4H}, [Xn]</code>	<code>Vt4.4H -> result.val[3]</code> <code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>poly16x8x4_t vld4q_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8H - Vt4.8H}, [Xn]</code>	<code>Vt4.8H -> result.val[3]</code> <code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>int64x1x4_t vld4_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt4.1D}, [Xn]</code>	<code>Vt4.1D -> result.val[3]</code> <code>Vt3.1D -> result.val[2]</code> <code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>uint64x1x4_t vld4_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt4.1D}, [Xn]</code>	<code>Vt4.1D -> result.val[3]</code> <code>Vt3.1D -> result.val[2]</code> <code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly64x1x4_t vld4_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt4.1D}, [Xn]</code>	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A32/A64
<code>int64x2x4_t vld4q_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>uint64x2x4_t vld4q_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>poly64x2x4_t vld4q_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>float64x1x4_t vld4_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt4.1D}, [Xn]</code>	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2x4_t vld4q_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>mfloat8x8x4_t vld4_mf8(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
<code>mfloat8x16x4_t vld4q_mf8(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.16B - Vt4.16B}, [Xn]</code>	Vt4.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64
<code>int8x8x2_t vld2_dup_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8B - Vt2.8B}, [Xn]</code>	Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A7/A32/A64
<code>int8x16x2_t vld2q_dup_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.16B - Vt2.16B}, [Xn]</code>	Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4x2_t vld2_dup_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.4H - Vt2.4H}, [Xn]</code>	<code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>int16x8x2_t vld2q_dup_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>int32x2x2_t vld2_dup_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.2S - Vt2.2S}, [Xn]</code>	<code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>int32x4x2_t vld2q_dup_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.4S - Vt2.4S}, [Xn]</code>	<code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64
<code>uint8x8x2_t vld2_dup_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8B - Vt2.8B}, [Xn]</code>	<code>Vt2.8B -> result.val[1]</code> <code>Vt.8B -> result.val[0]</code>	A32/A64
<code>uint8x16x2_t vld2q_dup_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.16B - Vt2.16B}, [Xn]</code>	<code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>uint16x4x2_t vld2_dup_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.4H - Vt2.4H}, [Xn]</code>	<code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8x2_t vld2q_dup_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>uint32x2x2_t vld2_dup_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.2S - Vt2.2S}, [Xn]</code>	<code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>uint32x4x2_t vld2q_dup_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.4S - Vt2.4S}, [Xn]</code>	<code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64
<code>float16x4x2_t vld2_dup_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.4H - Vt2.4H}, [Xn]</code>	<code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>float16x8x2_t vld2q_dup_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>float32x2x2_t vld2_dup_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.2S - Vt2.2S}, [Xn]</code>	<code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>float32x4x2_t vld2q_dup_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.4S - Vt2.4S}, [Xn]</code>	<code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x8x2_t vld2_dup_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8B - Vt2.8B}, [Xn]</code>	<code>Vt2.8B -> result.val[1]</code> <code>Vt.8B -> result.val[0]</code>	A32/A64
<code>poly8x16x2_t vld2q_dup_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.16B - Vt2.16B}, [Xn]</code>	<code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>poly16x4x2_t vld2_dup_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.4H - Vt2.4H}, [Xn]</code>	<code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>poly16x8x2_t vld2q_dup_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>int64x1x2_t vld2_dup_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>uint64x1x2_t vld2_dup_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>poly64x1x2_t vld2_dup_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x2x2_t vld2q_dup_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A64
<code>uint64x2x2_t vld2q_dup_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A64
<code>poly64x2x2_t vld2q_dup_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A64
<code>float64x1x2_t vld2_dup_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A64
<code>float64x2x2_t vld2q_dup_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A64
<code>mfloat8x8x2_t vld2_dup_mf8(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8B - Vt2.8B}, [Xn]</code>	<code>Vt2.8B -> result.val[1]</code> <code>Vt.8B -> result.val[0]</code>	A64
<code>mfloat8x16x2_t vld2q_dup_mf8(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.16B - Vt2.16B}, [Xn]</code>	<code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8x3_t vld3_dup_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64
<code>int8x16x3_t vld3q_dup_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.16B - Vt3.16B}, [Xn]</code>	Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A32/A64
<code>int16x4x3_t vld3_dup_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>int16x8x3_t vld3q_dup_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
<code>int32x2x3_t vld3_dup_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.2S - Vt3.2S}, [Xn]</code>	Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4x3_t vld3q_dup_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.4S - Vt3.4S}, [Xn]</code>	Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
<code>uint8x8x3_t vld3_dup_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64
<code>uint8x16x3_t vld3q_dup_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.16B - Vt3.16B}, [Xn]</code>	Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	v7/A32/A64
<code>uint16x4x3_t vld3_dup_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
<code>uint16x8x3_t vld3q_dup_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2x3_t vld3_dup_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.2S - Vt3.2S}, [Xn]</code>	Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64
<code>uint32x4x3_t vld3q_dup_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.4S - Vt3.4S}, [Xn]</code>	Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
<code>float16x4x3_t vld3_dup_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
<code>float16x8x3_t vld3q_dup_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
<code>float32x2x3_t vld3_dup_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.2S - Vt3.2S}, [Xn]</code>	Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4x3_t vld3q_dup_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.4S - Vt3.4S}, [Xn]</code>	Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
<code>poly8x8x3_t vld3_dup_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64
<code>poly8x16x3_t vld3q_dup_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.16B - Vt3.16B}, [Xn]</code>	Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	v7/A32/A64
<code>poly16x4x3_t vld3_dup_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
<code>poly16x8x3_t vld3q_dup_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1x3_t vld3_dup_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.1D - Vt3.1D}, [Xn]</code>	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A32/A64
<code>uint64x1x3_t vld3_dup_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.1D - Vt3.1D}, [Xn]</code>	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A32/A64
<code>poly64x1x3_t vld3_dup_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.1D - Vt3.1D}, [Xn]</code>	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A32/A64
<code>int64x2x3_t vld3q_dup_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>uint64x2x3_t vld3q_dup_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly64x2x3_t vld3q_dup_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>float64x1x3_t vld3_dup_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.1D - Vt3.1D}, [Xn]</code>	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64
<code>float64x2x3_t vld3q_dup_f64(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>mfloating8x8x3_t vld3_dup_mf8(mfloating8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
<code>mfloating8x16x3_t vld3q_dup_mf8(mfloating8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.16B - Vt3.16B}, [Xn]</code>	Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8x4_t vld4_dup_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64
<code>int8x16x4_t vld4q_dup_s8(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.16B - Vt4.16B}, [Xn]</code>	Vt4.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A32/A64
<code>int16x4x4_t vld4_dup_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>int16x8x4_t vld4q_dup_s16(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
<code>int32x2x4_t vld4_dup_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.2S - Vt4.2S}, [Xn]</code>	Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4x4_t vld4q_dup_s32(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.4S - Vt4.4S}, [Xn]</code>	Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	A32/A64
<code>uint8x8x4_t vld4_dup_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64
<code>uint8x16x4_t vld4q_dup_u8(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.16B - Vt4.16B}, [Xn]</code>	Vt4.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A32/A64
<code>uint16x4x4_t vld4_dup_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>uint16x8x4_t vld4q_dup_u16(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2x4_t vld4_dup_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.2S - Vt4.2S}, [Xn]</code>	Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	A32/A64
<code>uint32x4x4_t vld4q_dup_u32(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.4S - Vt4.4S}, [Xn]</code>	Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	A32/A64
<code>float16x4x4_t vld4_dup_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>float16x8x4_t vld4q_dup_f16(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
<code>float32x2x4_t vld4_dup_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.2S - Vt4.2S}, [Xn]</code>	Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4x4_t vld4q_dup_f32(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.4S - Vt4.4S}, [Xn]</code>	Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	A32/A64
<code>poly8x8x4_t vld4_dup_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64
<code>poly8x16x4_t vld4q_dup_p8(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.16B - Vt4.16B}, [Xn]</code>	Vt4.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A32/A64
<code>poly16x4x4_t vld4_dup_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>poly16x8x4_t vld4q_dup_p16(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1x4_t vld4_dup_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.1D - Vt4.1D}, [Xn]</code>	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A77/A32/A64
<code>uint64x1x4_t vld4_dup_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.1D - Vt4.1D}, [Xn]</code>	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A77/A32/A64
<code>poly64x1x4_t vld4_dup_p64(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.1D - Vt4.1D}, [Xn]</code>	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A32/A64
<code>int64x2x4_t vld4q_dup_s64(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>uint64x2x4_t vld4q_dup_u64(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x2x4_t vld4q_dup_p64(poly64_t const *ptr)	ptr -> Xn	LD4R {Vt.2D - Vt4.2D}, [Xn]	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
float64x1x4_t vld4_dup_f64(float64_t const *ptr)	ptr -> Xn	LD4R {Vt.1D - Vt4.1D}, [Xn]	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64
float64x2x4_t vld4q_dup_f64(float64_t const *ptr)	ptr -> Xn	LD4R {Vt.2D - Vt4.2D}, [Xn]	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
mfloating8x8x4_t vld4_dup_mf8(mfloating8_t const *ptr)	ptr -> Xn	LD4R {Vt.8B - Vt4.8B}, [Xn]	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
mfloating8x16x4_t vld4q_dup_mf8(mfloating8_t const *ptr)	ptr -> Xn	LD4R {Vt.16B - Vt4.16B}, [Xn]	Vt4.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4x2_t vld2_lane_s16(int16_t const *ptr, int16x4x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3</code>	LD2 {Vt.h - Vt2.h}[lane], [Xm1]	<code>Vt2.4H -> result.val[1] Vt.4H -> result.val[0]</code>	7/A32/A64
<code>int16x8x2_t vld2q_lane_s16(int16_t const *ptr, int16x8x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7</code>	LD2 {Vt.h - Vt2.h}[lane], [Xm1]	<code>Vt2.8H -> result.val[1] Vt.8H -> result.val[0]</code>	7/A32/A64
<code>int32x2x2_t vld2_lane_s32(int32_t const *ptr, int32x2x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1</code>	LD2 {Vt.s - Vt2.s}[lane], [Xm1]	<code>Vt2.2S -> result.val[1] Vt.2S -> result.val[0]</code>	7/A32/A64
<code>int32x4x2_t vld2q_lane_s32(int32_t const *ptr, int32x4x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3</code>	LD2 {Vt.s - Vt2.s}[lane], [Xm1]	<code>Vt2.4S -> result.val[1] Vt.4S -> result.val[0]</code>	7/A32/A64
<code>uint16x4x2_t vld2_lane_u16(uint16_t const *ptr, uint16x4x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3</code>	LD2 {Vt.h - Vt2.h}[lane], [Xm1]	<code>Vt2.4H -> result.val[1] Vt.4H -> result.val[0]</code>	7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>uint16x8x2_t vld2q_lane_u16(uint16_t const *ptr, uint16x8x2_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7</pre>	LD2 {Vt.h - Vt2.h}[lane], [Xn]	<pre>Vt2.8H -> result.val[1] Vt.8H -> result.val[0]</pre>	v7/A32/A64
<pre>uint32x2x2_t vld2_lane_u32(uint32_t const *ptr, uint32x2x2_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1</pre>	LD2 {Vt.s - Vt2.s}[lane], [Xn]	<pre>Vt2.2S -> result.val[1] Vt.2S -> result.val[0]</pre>	v7/A32/A64
<pre>uint32x4x2_t vld2q_lane_u32(uint32_t const *ptr, uint32x4x2_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3</pre>	LD2 {Vt.s - Vt2.s}[lane], [Xn]	<pre>Vt2.4S -> result.val[1] Vt.4S -> result.val[0]</pre>	v7/A32/A64
<pre>float16x4x2_t vld2_lane_f16(float16_t const *ptr, float16x4x2_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3</pre>	LD2 {Vt.h - Vt2.h}[lane], [Xn]	<pre>Vt2.4H -> result.val[1] Vt.4H -> result.val[0]</pre>	v7/A32/A64
<pre>float16x8x2_t vld2q_lane_f16(float16_t const *ptr, float16x8x2_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7</pre>	LD2 {Vt.h - Vt2.h}[lane], [Xn]	<pre>Vt2.8H -> result.val[1] Vt.8H -> result.val[0]</pre>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2x2_t vld2_lane_f32(float32_t const *ptr, float32x2x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1	LD2 {Vt.s - Vt2.s}[lane], [Xn], #2	Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64
float32x4x2_t vld2q_lane_f32(float32_t const *ptr, float32x4x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3	LD2 {Vt.s - Vt2.s}[lane], [Xn], #4	Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
poly16x4x2_t vld2_lane_p16(poly16_t const *ptr, poly16x4x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3	LD2 {Vt.h - Vt2.h}[lane], [Xn], #4	Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
poly16x8x2_t vld2q_lane_p16(poly16_t const *ptr, poly16x8x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7	LD2 {Vt.h - Vt2.h}[lane], [Xn], #8	Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
int8x8x2_t vld2_lane_s8(int8_t const *ptr, int8x8x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7	LD2 {Vt.b - Vt2.b}[lane], [Xn], #8	Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8x2_t vld2_lane_u8(uint8_t const *ptr, uint8x8x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7</code>	<code>LD2 {Vt.b - Vt2.b}[lane], [Xm1]</code>	<code>Vt2.8B -> result.val[1] Vt.8B -> result.val[0]</code>	A32/A64
<code>poly8x8x2_t vld2_lane_p8(poly8_t const *ptr, poly8x8x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7</code>	<code>LD2 {Vt.b - Vt2.b}[lane], [Xm1]</code>	<code>Vt2.8B -> result.val[1] Vt.8B -> result.val[0]</code>	A32/A64
<code>int8x16x2_t vld2q_lane_s8(int8_t const *ptr, int8x16x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</code>	<code>LD2 {Vt.b - Vt2.b}[lane], [Xm1]</code>	<code>Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</code>	A64
<code>uint8x16x2_t vld2q_lane_u8(uint8_t const *ptr, uint8x16x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</code>	<code>LD2 {Vt.b - Vt2.b}[lane], [Xm1]</code>	<code>Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</code>	A64
<code>poly8x16x2_t vld2q_lane_p8(poly8_t const *ptr, poly8x16x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</code>	<code>LD2 {Vt.b - Vt2.b}[lane], [Xm1]</code>	<code>Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1x2_t vld2_lane_s64(int64_t const *ptr, int64x1x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0</code>	<code>LD2 {Vt.d - Vt2.d}[lane], [Xn], #2</code>	<code>ptr -> Xn Vt2.1D -> result.val[1] Vt.1D -> result.val[0]</code>	A64
<code>int64x2x2_t vld2q_lane_s64(int64_t const *ptr, int64x2x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1</code>	<code>LD2 {Vt.d - Vt2.d}[lane], [Xn], #4</code>	<code>ptr -> Xn Vt2.2D -> result.val[1] Vt.2D -> result.val[0]</code>	A64
<code>uint64x1x2_t vld2_lane_u64(uint64_t const *ptr, uint64x1x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0</code>	<code>LD2 {Vt.d - Vt2.d}[lane], [Xn], #2</code>	<code>ptr -> Xn Vt2.1D -> result.val[1] Vt.1D -> result.val[0]</code>	A64
<code>uint64x2x2_t vld2q_lane_u64(uint64_t const *ptr, uint64x2x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1</code>	<code>LD2 {Vt.d - Vt2.d}[lane], [Xn], #4</code>	<code>ptr -> Xn Vt2.2D -> result.val[1] Vt.2D -> result.val[0]</code>	A64
<code>poly64x1x2_t vld2_lane_p64(poly64_t const *ptr, poly64x1x2_t src, const int lane)</code>	<code>ptr -> Xn src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0</code>	<code>LD2 {Vt.d - Vt2.d}[lane], [Xn], #2</code>	<code>ptr -> Xn Vt2.1D -> result.val[1] Vt.1D -> result.val[0]</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x2x2_t vld2q_lane_p64(poly64_t const *ptr, poly64x2x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1	LD2 {Vt.d - Vt2.d}[lane], [Xn], #2	Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
float64x1x2_t vld2_lane_f64(float64_t const *ptr, float64x1x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0	LD2 {Vt.d - Vt2.d}[lane], [Xn], #1	Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64
float64x2x2_t vld2q_lane_f64(float64_t const *ptr, float64x2x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1	LD2 {Vt.d - Vt2.d}[lane], [Xn], #2	Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
mfloat8x8x2_t vld2_lane_mf8(mfloat8_t const *ptr, mfloat8x8x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7	LD2 {Vt.b - Vt2.b}[lane], [Xn], #2.8B	Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
mfloat8x16x2_t vld2q_lane_mf8(mfloat8_t const *ptr, mfloat8x16x2_t src, const int lane)	ptr -> Xn src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15	LD2 {Vt.b - Vt2.b}[lane], [Xn], #2.16B	Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>int16x4x3_t vld3_lane_s16(int16_t const *ptr, int16x4x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3</pre>	LD3 {Vt.h - Vt3.h}[lane], [Xm1]	<pre>Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]</pre>	v7/A32/A64
<pre>int16x8x3_t vld3q_lane_s16(int16_t const *ptr, int16x8x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7</pre>	LD3 {Vt.h - Vt3.h}[lane], [Xm1]	<pre>Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]</pre>	v7/A32/A64
<pre>int32x2x3_t vld3_lane_s32(int32_t const *ptr, int32x2x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.2S src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1</pre>	LD3 {Vt.s - Vt3.s}[lane], [Xm1]	<pre>Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]</pre>	v7/A32/A64
<pre>int32x4x3_t vld3q_lane_s32(int32_t const *ptr, int32x4x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.4S src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3</pre>	LD3 {Vt.s - Vt3.s}[lane], [Xm1]	<pre>Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]</pre>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>uint16x4x3_t vld3_lane_u16(uint16_t const *ptr, uint16x4x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3</pre>	LD3 {Vt.h - Vt3.h}[lane], [Xm1]	<pre>Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]</pre>	v7/A32/A64
<pre>uint16x8x3_t vld3q_lane_u16(uint16_t const *ptr, uint16x8x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7</pre>	LD3 {Vt.h - Vt3.h}[lane], [Xm1]	<pre>Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]</pre>	v7/A32/A64
<pre>uint32x2x3_t vld3_lane_u32(uint32_t const *ptr, uint32x2x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.2S src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1</pre>	LD3 {Vt.s - Vt3.s}[lane], [Xm1]	<pre>Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]</pre>	v7/A32/A64
<pre>uint32x4x3_t vld3q_lane_u32(uint32_t const *ptr, uint32x4x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.4S src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3</pre>	LD3 {Vt.s - Vt3.s}[lane], [Xm1]	<pre>Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]</pre>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4x3_t vld3_lane_f16(float16_t const *ptr, float16x4x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3	LD3 {Vt.h - Vt3.h}[lane], [Xn]	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
float16x8x3_t vld3q_lane_f16(float16_t const *ptr, float16x8x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7	LD3 {Vt.h - Vt3.h}[lane], [Xn]	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
float32x2x3_t vld3_lane_f32(float32_t const *ptr, float32x2x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.2S src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1	LD3 {Vt.s - Vt3.s}[lane], [Xn]	Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64
float32x4x3_t vld3q_lane_f32(float32_t const *ptr, float32x4x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.4S src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3	LD3 {Vt.s - Vt3.s}[lane], [Xn]	Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly16x4x3_t vld3_lane_p16(poly16_t const *ptr, poly16x4x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3	LD3 {Vt.h - Vt3.h}[lane], [Xn], #3	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
poly16x8x3_t vld3q_lane_p16(poly16_t const *ptr, poly16x8x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7	LD3 {Vt.h - Vt3.h}[lane], [Xn], #3	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
int8x8x3_t vld3_lane_s8(int8_t const *ptr, int8x8x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.8B src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7	LD3 {Vt.b - Vt3.b}[lane], [Xn], #3	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64
uint8x8x3_t vld3_lane_u8(uint8_t const *ptr, uint8x8x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.8B src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7	LD3 {Vt.b - Vt3.b}[lane], [Xn], #3	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>poly8x8x3_t vld3_lane_p8(poly8_t const *ptr, poly8x8x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.8B src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7</pre>	LD3 {Vt.b - Vt3.b}[lane], [Xm1]	<pre>Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]</pre>	A7/A32/A64
<pre>int8x16x3_t vld3q_lane_s8(int8_t const *ptr, int8x16x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.16B src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</pre>	LD3 {Vt.b - Vt3.b}[lane], [Xm1]	<pre>Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</pre>	A64
<pre>uint8x16x3_t vld3q_lane_u8(uint8_t const *ptr, uint8x16x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.16B src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</pre>	LD3 {Vt.b - Vt3.b}[lane], [Xm1]	<pre>Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</pre>	A64
<pre>poly8x16x3_t vld3q_lane_p8(poly8_t const *ptr, poly8x16x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.16B src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</pre>	LD3 {Vt.b - Vt3.b}[lane], [Xm1]	<pre>Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</pre>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>int64x1x3_t vld3_lane_s64(int64_t const *ptr, int64x1x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.1D src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0</pre>	LD3 {Vt.d - Vt3.d}[lane], [Xm1]	<pre>Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]</pre>	A64
<pre>int64x2x3_t vld3q_lane_s64(int64_t const *ptr, int64x2x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.2D src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1</pre>	LD3 {Vt.d - Vt3.d}[lane], [Xm1]	<pre>Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]</pre>	A64
<pre>uint64x1x3_t vld3_lane_u64(uint64_t const *ptr, uint64x1x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.1D src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0</pre>	LD3 {Vt.d - Vt3.d}[lane], [Xm1]	<pre>Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]</pre>	A64
<pre>uint64x2x3_t vld3q_lane_u64(uint64_t const *ptr, uint64x2x3_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[2] -> Vt3.2D src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1</pre>	LD3 {Vt.d - Vt3.d}[lane], [Xm1]	<pre>Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]</pre>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x1x3_t vld3_lane_p64(poly64_t const *ptr, poly64x1x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.1D src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0	LD3 {Vt.d - Vt3.d}[lane], [Xn]	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64
poly64x2x3_t vld3q_lane_p64(poly64_t const *ptr, poly64x2x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.2D src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1	LD3 {Vt.d - Vt3.d}[lane], [Xn]	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
float64x1x3_t vld3_lane_f64(float64_t const *ptr, float64x1x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.1D src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0	LD3 {Vt.d - Vt3.d}[lane], [Xn]	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64
float64x2x3_t vld3q_lane_f64(float64_t const *ptr, float64x2x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.2D src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1	LD3 {Vt.d - Vt3.d}[lane], [Xn]	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre> mfloat8x8x3_t vld3_lane_mf8(mfloat8_t const *ptr, mfloat8x8x3_t src, const int lane) </pre>	<pre> ptr -> Xn src.val[2] -> Vt3.8B src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7 </pre>	LD3 {Vt.b - Vt3.b}[lane], [Xm1]	<pre> Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0] </pre>	A64
<pre> mfloat8x16x3_t vld3q_lane_mf8(mfloat8_t const *ptr, mfloat8x16x3_t src, const int lane) </pre>	<pre> ptr -> Xn src.val[2] -> Vt3.16B src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15 </pre>	LD3 {Vt.b - Vt3.b}[lane], [Xm1]	<pre> Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0] </pre>	A64
<pre> int16x4x4_t vld4_lane_s16(int16_t const *ptr, int16x4x4_t src, const int lane) </pre>	<pre> ptr -> Xn src.val[3] -> Vt4.4H src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3 </pre>	LD4 {Vt.h - Vt4.h}[lane], [Xm1]	<pre> Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0] </pre>	A7/A32/A64
<pre> int16x8x4_t vld4q_lane_s16(int16_t const *ptr, int16x8x4_t src, const int lane) </pre>	<pre> ptr -> Xn src.val[3] -> Vt4.8H src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7 </pre>	LD4 {Vt.h - Vt4.h}[lane], [Xm1]	<pre> Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0] </pre>	A7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>int32x2x4_t vld4_lane_s32(int32_t const *ptr, int32x2x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.2S src.val[2] -> Vt3.2S src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1</pre>	LD4 {Vt.s - Vt4.s}[lane], [Xn], #4	<pre>Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]</pre>	v7/A32/A64
<pre>int32x4x4_t vld4q_lane_s32(int32_t const *ptr, int32x4x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.4S src.val[2] -> Vt3.4S src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3</pre>	LD4 {Vt.s - Vt4.s}[lane], [Xn], #4	<pre>Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]</pre>	v7/A32/A64
<pre>uint16x4x4_t vld4_lane_u16(uint16_t const *ptr, uint16x4x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.4H src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3</pre>	LD4 {Vt.h - Vt4.h}[lane], [Xn], #4	<pre>Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]</pre>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>uint16x8x4_t vld4q_lane_u16(uint16_t const *ptr, uint16x8x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.8H src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7</pre>	LD4 {Vt.h - Vt4.h}[lane], [Xn]	<pre>Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]</pre>	v7/A32/A64
<pre>uint32x2x4_t vld4_lane_u32(uint32_t const *ptr, uint32x2x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.2S src.val[2] -> Vt3.2S src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1</pre>	LD4 {Vt.s - Vt4.s}[lane], [Xn]	<pre>Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]</pre>	v7/A32/A64
<pre>uint32x4x4_t vld4q_lane_u32(uint32_t const *ptr, uint32x4x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.4S src.val[2] -> Vt3.4S src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3</pre>	LD4 {Vt.s - Vt4.s}[lane], [Xn]	<pre>Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]</pre>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4x4_t vld4_lane_f16(float16_t const *ptr, float16x4x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.4H src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3	LD4 {Vt.h - Vt4.h}[lane], [Xn]	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
float16x8x4_t vld4q_lane_f16(float16_t const *ptr, float16x8x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.8H src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7	LD4 {Vt.h - Vt4.h}[lane], [Xn]	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
float32x2x4_t vld4_lane_f32(float32_t const *ptr, float32x2x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.2S src.val[2] -> Vt3.2S src.val[1] -> Vt2.2S src.val[0] -> Vt.2S 0 <= lane <= 1	LD4 {Vt.s - Vt4.s}[lane], [Xn]	Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4x4_t vld4q_lane_f32(float32_t const *ptr, float32x4x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.4S src.val[2] -> Vt3.4S src.val[1] -> Vt2.4S src.val[0] -> Vt.4S 0 <= lane <= 3	LD4 {Vt.s - Vt4.s}[lane], [Xn]	Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
poly16x4x4_t vld4_lane_p16(poly16_t const *ptr, poly16x4x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.4H src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3	LD4 {Vt.h - Vt4.h}[lane], [Xn]	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
poly16x8x4_t vld4q_lane_p16(poly16_t const *ptr, poly16x8x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.8H src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7	LD4 {Vt.h - Vt4.h}[lane], [Xn]	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>int8x8x4_t vld4_lane_s8(int8_t const *ptr, int8x8x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.8B src.val[2] -> Vt3.8B src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7</pre>	LD4 {Vt.b - Vt4.b}[lane], [Xn]	<pre>Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]</pre>	v7/A32/A64
<pre>uint8x8x4_t vld4_lane_u8(uint8_t const *ptr, uint8x8x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.8B src.val[2] -> Vt3.8B src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7</pre>	LD4 {Vt.b - Vt4.b}[lane], [Xn]	<pre>Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]</pre>	v7/A32/A64
<pre>poly8x8x4_t vld4_lane_p8(poly8_t const *ptr, poly8x8x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.8B src.val[2] -> Vt3.8B src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7</pre>	LD4 {Vt.b - Vt4.b}[lane], [Xn]	<pre>Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]</pre>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16x4_t vld4q_lane_s8(int8_t const *ptr, int8x16x4_t src, const int lane)</code>	<code>ptr -> Xn src.val[3] -> Vt4.16B src.val[2] -> Vt3.16B src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</code>	<code>LD4 {Vt.b - Vt4.b}[lane], [Xn]</code>	<code>Vt.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</code>	<code>AArch64</code>
<code>uint8x16x4_t vld4q_lane_u8(uint8_t const *ptr, uint8x16x4_t src, const int lane)</code>	<code>ptr -> Xn src.val[3] -> Vt4.16B src.val[2] -> Vt3.16B src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</code>	<code>LD4 {Vt.b - Vt4.b}[lane], [Xn]</code>	<code>Vt.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</code>	<code>AArch64</code>
<code>poly8x16x4_t vld4q_lane_p8(poly8_t const *ptr, poly8x16x4_t src, const int lane)</code>	<code>ptr -> Xn src.val[3] -> Vt4.16B src.val[2] -> Vt3.16B src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</code>	<code>LD4 {Vt.b - Vt4.b}[lane], [Xn]</code>	<code>Vt.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</code>	<code>AArch64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>int64x1x4_t vld4_lane_s64(int64_t const *ptr, int64x1x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.1D src.val[2] -> Vt3.1D src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0</pre>	LD4 {Vt.d - Vt4.d}[lane], [Xn], #4	<pre>Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]</pre>	A64
<pre>int64x2x4_t vld4q_lane_s64(int64_t const *ptr, int64x2x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.2D src.val[2] -> Vt3.2D src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1</pre>	LD4 {Vt.d - Vt4.d}[lane], [Xn], #8	<pre>Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]</pre>	A64
<pre>uint64x1x4_t vld4_lane_u64(uint64_t const *ptr, uint64x1x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.1D src.val[2] -> Vt3.1D src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0</pre>	LD4 {Vt.d - Vt4.d}[lane], [Xn], #4	<pre>Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]</pre>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint64x2x4_t vld4q_lane_u64(uint64_t const *ptr, uint64x2x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.2D src.val[2] -> Vt3.2D src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1	LD4 {Vt.d - Vt4.d}[lane], [Xn]	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
poly64x1x4_t vld4_lane_p64(poly64_t const *ptr, poly64x1x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.1D src.val[2] -> Vt3.1D src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0	LD4 {Vt.d - Vt4.d}[lane], [Xn]	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64
poly64x2x4_t vld4q_lane_p64(poly64_t const *ptr, poly64x2x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.2D src.val[2] -> Vt3.2D src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1	LD4 {Vt.d - Vt4.d}[lane], [Xn]	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>float64x1x4_t vld4_lane_f64(float64_t const *ptr, float64x1x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.1D src.val[2] -> Vt3.1D src.val[1] -> Vt2.1D src.val[0] -> Vt.1D 0 <= lane <= 0</pre>	LD4 {Vt.d - Vt4.d}[lane], [Xn], #4	<pre>Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]</pre>	A64
<pre>float64x2x4_t vld4q_lane_f64(float64_t const *ptr, float64x2x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.2D src.val[2] -> Vt3.2D src.val[1] -> Vt2.2D src.val[0] -> Vt.2D 0 <= lane <= 1</pre>	LD4 {Vt.d - Vt4.d}[lane], [Xn], #8	<pre>Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]</pre>	A64
<pre>mfloat8x8x4_t vld4_lane_mf8(mfloat8_t const *ptr, mfloat8x8x4_t src, const int lane)</pre>	<pre>ptr -> Xn src.val[3] -> Vt4.8B src.val[2] -> Vt3.8B src.val[1] -> Vt2.8B src.val[0] -> Vt.8B 0 <= lane <= 7</pre>	LD4 {Vt.b - Vt4.b}[lane], [Xn], #16	<pre>Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]</pre>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x16x4_t vld4q_lane_mf8(mfloat8_t const *ptr, mfloat8x16x4_t src, const int lane)</code>	<code>ptr -> Xn src.val[3] -> Vt4.16B src.val[2] -> Vt3.16B src.val[1] -> Vt2.16B src.val[0] -> Vt.16B 0 <= lane <= 15</code>	<code>LD4 {Vt.b - Vt4.b}[lane], [Xn]</code>	<code>Vt.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</code>	A32/A64
<code>int8x8x2_t vld1_s8_x2(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt2.8B}, [Xn]</code>	<code>Vt2.8B -> result.val[1] Vt.8B -> result.val[0]</code>	A32/A64
<code>int8x16x2_t vld1q_s8_x2(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt2.16B}, [Xn]</code>	<code>Vt2.16B -> result.val[1] Vt.16B -> result.val[0]</code>	A32/A64
<code>int16x4x2_t vld1_s16_x2(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt2.4H}, [Xn]</code>	<code>Vt2.4H -> result.val[1] Vt.4H -> result.val[0]</code>	A32/A64
<code>int16x8x2_t vld1q_s16_x2(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1] Vt.8H -> result.val[0]</code>	A32/A64
<code>int32x2x2_t vld1_s32_x2(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S - Vt2.2S}, [Xn]</code>	<code>Vt2.2S -> result.val[1] Vt.2S -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4x2_t vld1q_s32_x2(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S - Vt2.4S}, [Xn]</code>	<code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64
<code>uint8x8x2_t vld1_u8_x2(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt2.8B}, [Xn]</code>	<code>Vt2.8B -> result.val[1]</code> <code>Vt.8B -> result.val[0]</code>	A32/A64
<code>uint8x16x2_t vld1q_u8_x2(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt2.16B}, [Xn]</code>	<code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>uint16x4x2_t vld1_u16_x2(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt2.4H}, [Xn]</code>	<code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>uint16x8x2_t vld1q_u16_x2(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>uint32x2x2_t vld1_u32_x2(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S - Vt2.2S}, [Xn]</code>	<code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>uint32x4x2_t vld1q_u32_x2(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S - Vt2.4S}, [Xn]</code>	<code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4x2_t vld1_f16_x2(float16_t const *ptr)	ptr -> Xn	LD1 {Vt.4H - Vt2.4H}, [Xn]	Vt2.4H -> result.val[1]7/A32/A64 Vt.4H -> result.val[0]	
float16x8x2_t vld1q_f16_x2(float16_t const *ptr)	ptr -> Xn	LD1 {Vt.8H - Vt2.8H}, [Xn]	Vt2.8H -> result.val[1]7/A32/A64 Vt.8H -> result.val[0]	
float32x2x2_t vld1_f32_x2(float32_t const *ptr)	ptr -> Xn	LD1 {Vt.2S - Vt2.2S}, [Xn]	Vt2.2S -> result.val[1]7/A32/A64 Vt.2S -> result.val[0]	
float32x4x2_t vld1q_f32_x2(float32_t const *ptr)	ptr -> Xn	LD1 {Vt.4S - Vt2.4S}, [Xn]	Vt2.4S -> result.val[1]7/A32/A64 Vt.4S -> result.val[0]	
poly8x8x2_t vld1_p8_x2(poly8_t const *ptr)	ptr -> Xn	LD1 {Vt.8B - Vt2.8B}, [Xn]	Vt2.8B -> result.val[1]7/A32/A64 Vt.8B -> result.val[0]	
poly8x16x2_t vld1q_p8_x2(poly8_t const *ptr)	ptr -> Xn	LD1 {Vt.16B - Vt2.16B}, [Xn]	Vt2.16B -> result.val[1]7/A32/A64 Vt.16B -> result.val[0]	
poly16x4x2_t vld1_p16_x2(poly16_t const *ptr)	ptr -> Xn	LD1 {Vt.4H - Vt2.4H}, [Xn]	Vt2.4H -> result.val[1]7/A32/A64 Vt.4H -> result.val[0]	

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x8x2_t vld1q_p16_x2(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>int64x1x2_t vld1_s64_x2(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>uint64x1x2_t vld1_u64_x2(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>poly64x1x2_t vld1_p64_x2(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt2.1D}, [Xn]</code>	<code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>int64x2x2_t vld1q_s64_x2(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A32/A64
<code>uint64x2x2_t vld1q_u64_x2(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A32/A64
<code>poly64x2x2_t vld1q_p64_x2(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D - Vt2.2D}, [Xn]</code>	<code>Vt2.2D -> result.val[1]</code> <code>Vt.2D -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x1x2_t vld1_f64_x2(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt2.1D}, [Xn]</code>	Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64
<code>float64x2x2_t vld1q_f64_x2(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D - Vt2.2D}, [Xn]</code>	Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>mfloat8x8x2_t vld1_mf8_x2(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt2.8B}, [Xn]</code>	Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
<code>mfloat8x16x2_t vld1q_mf8_x2(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt2.16B}, [Xn]</code>	Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64
<code>int8x8x3_t vld1_s8_x3(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64
<code>int8x16x3_t vld1q_s8_x3(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt3.16B}, [Xn]</code>	Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4x3_t vld1_s16_x3(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
<code>int16x8x3_t vld1q_s16_x3(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
<code>int32x2x3_t vld1_s32_x3(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S - Vt3.2S}, [Xn]</code>	Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64
<code>int32x4x3_t vld1q_s32_x3(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S - Vt3.4S}, [Xn]</code>	Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
<code>uint8x8x3_t vld1_u8_x3(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16x3_t vld1q_u8_x3(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt3.16B}, [Xn]</code>	<code>Vt3.16B -> result.val[2]</code> <code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>uint16x4x3_t vld1_u16_x3(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt3.4H}, [Xn]</code>	<code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>uint16x8x3_t vld1q_u16_x3(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt3.8H}, [Xn]</code>	<code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>uint32x2x3_t vld1_u32_x3(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S - Vt3.2S}, [Xn]</code>	<code>Vt3.2S -> result.val[2]</code> <code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>uint32x4x3_t vld1q_u32_x3(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S - Vt3.4S}, [Xn]</code>	<code>Vt3.4S -> result.val[2]</code> <code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4x3_t vld1_f16_x3(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	v7/A32/A64
<code>float16x8x3_t vld1q_f16_x3(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	v7/A32/A64
<code>float32x2x3_t vld1_f32_x3(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S - Vt3.2S}, [Xn]</code>	Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	v7/A32/A64
<code>float32x4x3_t vld1q_f32_x3(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S - Vt3.4S}, [Xn]</code>	Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	v7/A32/A64
<code>poly8x8x3_t vld1_p8_x3(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x16x3_t vld1q_p8_x3(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt3.16B}, [Xn]</code>	<code>Vt3.16B -> result.val[2]</code> <code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>poly16x4x3_t vld1_p16_x3(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt3.4H}, [Xn]</code>	<code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>poly16x8x3_t vld1q_p16_x3(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt3.8H}, [Xn]</code>	<code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>int64x1x3_t vld1_s64_x3(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt3.1D}, [Xn]</code>	<code>Vt3.1D -> result.val[2]</code> <code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>uint64x1x3_t vld1_u64_x3(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt3.1D}, [Xn]</code>	<code>Vt3.1D -> result.val[2]</code> <code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly64x1x3_t vld1_p64_x3(poly64_t const *ptr)	ptr -> Xn	LD1 {Vt.1D - Vt3.1D}, [Xn]	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A32/A64
int64x2x3_t vld1q_s64_x3(int64_t const *ptr)	ptr -> Xn	LD1 {Vt.2D - Vt3.2D}, [Xn]	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A32/A64
uint64x2x3_t vld1q_u64_x3(uint64_t const *ptr)	ptr -> Xn	LD1 {Vt.2D - Vt3.2D}, [Xn]	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A32/A64
poly64x2x3_t vld1q_p64_x3(poly64_t const *ptr)	ptr -> Xn	LD1 {Vt.2D - Vt3.2D}, [Xn]	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A32/A64
float64x1x3_t vld1_f64_x3(float64_t const *ptr)	ptr -> Xn	LD1 {Vt.1D - Vt3.1D}, [Xn]	Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float64x2x3_t vld1q_f64_x3(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D - Vt3.2D}, [Xn]</code>	Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
<code>mffloat8x8x3_t vld1_mf8_x3(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt3.8B}, [Xn]</code>	Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
<code>mffloat8x16x3_t vld1q_mf8_x3(mfloat8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt3.16B}, [Xn]</code>	Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64
<code>int8x8x4_t vld1_s8_x4(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A7/A32/A64
<code>int8x16x4_t vld1q_s8_x4(int8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt4.16B}, [Xn]</code>	Vt4.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4x4_t vld1_s16_x4(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>int16x8x4_t vld1q_s16_x4(int16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
<code>int32x2x4_t vld1_s32_x4(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S - Vt4.2S}, [Xn]</code>	Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	A32/A64
<code>int32x4x4_t vld1q_s32_x4(int32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S - Vt4.4S}, [Xn]</code>	Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	A32/A64
<code>uint8x8x4_t vld1_u8_x4(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16x4_t vld1q_u8_x4(uint8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt4.16B}, [Xn]</code>	<code>Vt4.16B -> result.val[3]</code> <code>Vt3.16B -> result.val[2]</code> <code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>uint16x4x4_t vld1_u16_x4(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt4.4H}, [Xn]</code>	<code>Vt4.4H -> result.val[3]</code> <code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>uint16x8x4_t vld1q_u16_x4(uint16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt4.8H}, [Xn]</code>	<code>Vt4.8H -> result.val[3]</code> <code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>uint32x2x4_t vld1_u32_x4(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S - Vt4.2S}, [Xn]</code>	<code>Vt4.2S -> result.val[3]</code> <code>Vt3.2S -> result.val[2]</code> <code>Vt2.2S -> result.val[1]</code> <code>Vt.2S -> result.val[0]</code>	A32/A64
<code>uint32x4x4_t vld1q_u32_x4(uint32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S - Vt4.4S}, [Xn]</code>	<code>Vt4.4S -> result.val[3]</code> <code>Vt3.4S -> result.val[2]</code> <code>Vt2.4S -> result.val[1]</code> <code>Vt.4S -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4x4_t vld1_f16_x4(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
<code>float16x8x4_t vld1q_f16_x4(float16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
<code>float32x2x4_t vld1_f32_x4(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2S - Vt4.2S}, [Xn]</code>	Vt4.2S -> result.val[3] Vt3.2S -> result.val[2] Vt2.2S -> result.val[1] Vt.2S -> result.val[0]	A32/A64
<code>float32x4x4_t vld1q_f32_x4(float32_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4S - Vt4.4S}, [Xn]</code>	Vt4.4S -> result.val[3] Vt3.4S -> result.val[2] Vt2.4S -> result.val[1] Vt.4S -> result.val[0]	A32/A64
<code>poly8x8x4_t vld1_p8_x4(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8B - Vt4.8B}, [Xn]</code>	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x16x4_t vld1q_p8_x4(poly8_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.16B - Vt4.16B}, [Xn]</code>	<code>Vt4.16B -> result.val[3]</code> <code>Vt3.16B -> result.val[2]</code> <code>Vt2.16B -> result.val[1]</code> <code>Vt.16B -> result.val[0]</code>	A32/A64
<code>poly16x4x4_t vld1_p16_x4(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt4.4H}, [Xn]</code>	<code>Vt4.4H -> result.val[3]</code> <code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>poly16x8x4_t vld1q_p16_x4(poly16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt4.8H}, [Xn]</code>	<code>Vt4.8H -> result.val[3]</code> <code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>int64x1x4_t vld1_s64_x4(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt4.1D}, [Xn]</code>	<code>Vt4.1D -> result.val[3]</code> <code>Vt3.1D -> result.val[2]</code> <code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64
<code>uint64x1x4_t vld1_u64_x4(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt4.1D}, [Xn]</code>	<code>Vt4.1D -> result.val[3]</code> <code>Vt3.1D -> result.val[2]</code> <code>Vt2.1D -> result.val[1]</code> <code>Vt.1D -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly64x1x4_t vld1_p64_x4(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt4.1D}, [Xn]</code>	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A32/A64
<code>int64x2x4_t vld1q_s64_x4(int64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A32/A64
<code>uint64x2x4_t vld1q_u64_x4(uint64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A32/A64
<code>poly64x2x4_t vld1q_p64_x4(poly64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.2D - Vt4.2D}, [Xn]</code>	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A32/A64
<code>float64x1x4_t vld1_f64_x4(float64_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.1D - Vt4.1D}, [Xn]</code>	Vt4.1D -> result.val[3] Vt3.1D -> result.val[2] Vt2.1D -> result.val[1] Vt.1D -> result.val[0]	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float64x2x4_t vld1q_f64_x4(float64_t const *ptr)	ptr -> Xn	LD1 {Vt.2D - Vt4.2D},[Xn]	Vt4.2D -> result.val[3] Vt3.2D -> result.val[2] Vt2.2D -> result.val[1] Vt.2D -> result.val[0]	A64
mfloat8x8x4_t vld1q_mf8_x4(mfloat8_t const *ptr)	ptr -> Xn	LD1 {Vt.8B - Vt4.8B},[Xn]	Vt4.8B -> result.val[3] Vt3.8B -> result.val[2] Vt2.8B -> result.val[1] Vt.8B -> result.val[0]	A64
mfloat8x16x4_t vld1q_mf8_x4(mfloat8_t const *ptr)	ptr -> Xn	LD1 {Vt.16B - Vt4.16B},[Xn]	Vt4.16B -> result.val[3] Vt3.16B -> result.val[2] Vt2.16B -> result.val[1] Vt.16B -> result.val[0]	A64

2.1.10.2 Load

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly128_t vldrq_p128(poly128_t const *ptr)	ptr -> Xn	LDR Qd,[Xn]	Qd -> result	A32/A64

2.1.11 Store

2.1.11.1 Stride

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_s8(int8_t *ptr, int8x8_t val)</code>	<code>val -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B}, [Xn]		v7/A32/A64
<code>void vst1q_s8(int8_t *ptr, int8x16_t val)</code>	<code>val -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B}, [Xn]		v7/A32/A64
<code>void vst1_s16(int16_t *ptr, int16x4_t val)</code>	<code>val -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H}, [Xn]		v7/A32/A64
<code>void vst1q_s16(int16_t *ptr, int16x8_t val)</code>	<code>val -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H}, [Xn]		v7/A32/A64
<code>void vst1_s32(int32_t *ptr, int32x2_t val)</code>	<code>val -> Vt.2S ptr -> Xn</code>	ST1 {Vt.2S}, [Xn]		v7/A32/A64
<code>void vst1q_s32(int32_t *ptr, int32x4_t val)</code>	<code>val -> Vt.4S ptr -> Xn</code>	ST1 {Vt.4S}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_s64(int64_t *ptr, int64x1_t val)</code>	<code>val -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D}, [Xn]		v7/A32/A64
<code>void vst1q_s64(int64_t *ptr, int64x2_t val)</code>	<code>val -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D}, [Xn]		v7/A32/A64
<code>void vst1_u8(uint8_t *ptr, uint8x8_t val)</code>	<code>val -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B}, [Xn]		v7/A32/A64
<code>void vst1q_u8(uint8_t *ptr, uint8x16_t val)</code>	<code>val -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B}, [Xn]		v7/A32/A64
<code>void vst1_u16(uint16_t *ptr, uint16x4_t val)</code>	<code>val -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H}, [Xn]		v7/A32/A64
<code>void vst1q_u16(uint16_t *ptr, uint16x8_t val)</code>	<code>val -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H}, [Xn]		v7/A32/A64
<code>void vst1_u32(uint32_t *ptr, uint32x2_t val)</code>	<code>val -> Vt.2S ptr -> Xn</code>	ST1 {Vt.2S}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_u32(uint32_t *ptr, uint32x4_t val)</code>	<code>val -> Vt.4S ptr -> Xn</code>	ST1 {Vt.4S}, [Xn]		v7/A32/A64
<code>void vst1_u64(uint64_t *ptr, uint64x1_t val)</code>	<code>val -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D}, [Xn]		v7/A32/A64
<code>void vst1q_u64(uint64_t *ptr, uint64x2_t val)</code>	<code>val -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D}, [Xn]		v7/A32/A64
<code>void vst1_p64(poly64_t *ptr, poly64x1_t val)</code>	<code>val -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D}, [Xn]		A32/A64
<code>void vst1q_p64(poly64_t *ptr, poly64x2_t val)</code>	<code>val -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D}, [Xn]		A32/A64
<code>void vst1_f16(float16_t *ptr, float16x4_t val)</code>	<code>val -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H}, [Xn]		v7/A32/A64
<code>void vst1q_f16(float16_t *ptr, float16x8_t val)</code>	<code>val -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_f32(float32_t *ptr, float32x2_t val)</code>	<code>val -> Vt.2S ptr -> Xn</code>	ST1 {Vt.2S}, [Xn]		v7/A32/A64
<code>void vst1q_f32(float32_t *ptr, float32x4_t val)</code>	<code>val -> Vt.4S ptr -> Xn</code>	ST1 {Vt.4S}, [Xn]		v7/A32/A64
<code>void vst1_p8(poly8_t *ptr, poly8x8_t val)</code>	<code>val -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B}, [Xn]		v7/A32/A64
<code>void vst1q_p8(poly8_t *ptr, poly8x16_t val)</code>	<code>val -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B}, [Xn]		v7/A32/A64
<code>void vst1_p16(poly16_t *ptr, poly16x4_t val)</code>	<code>val -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H}, [Xn]		v7/A32/A64
<code>void vst1q_p16(poly16_t *ptr, poly16x8_t val)</code>	<code>val -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H}, [Xn]		v7/A32/A64
<code>void vst1_f64(float64_t *ptr, float64x1_t val)</code>	<code>val -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D}, [Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_f64(float64_t *ptr, float64x2_t val)</code>	<code>val -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D},[Xn]		A64
<code>void vst1_mf8(mfloat8_t *ptr, mfloat8x8_t val)</code>	<code>val -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B},[Xn]		A64
<code>void vst1q_mf8(mfloat8_t *ptr, mfloat8x16_t val)</code>	<code>val -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B},[Xn]		A64
<code>void vst1_lane_s8(int8_t *ptr, int8x8_t val, const int lane)</code>	<code>val -> Vt.8B ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.b}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_s8(int8_t *ptr, int8x16_t val, const int lane)</code>	<code>val -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	ST1 {Vt.b}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_s16(int16_t *ptr, int16x4_t val, const int lane)</code>	<code>val -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST1 {Vt.h}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_lane_s16(int16_t *ptr, int16x8_t val, const int lane)</code>	<code>val -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_s32(int32_t *ptr, int32x2_t val, const int lane)</code>	<code>val -> Vt.2S ptr -> Xn 0 <= lane <= 1</code>	ST1 {Vt.s}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_s32(int32_t *ptr, int32x4_t val, const int lane)</code>	<code>val -> Vt.4S ptr -> Xn 0 <= lane <= 3</code>	ST1 {Vt.s}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_s64(int64_t *ptr, int64x1_t val, const int lane)</code>	<code>val -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	ST1 {Vt.d}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_s64(int64_t *ptr, int64x2_t val, const int lane)</code>	<code>val -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	ST1 {Vt.d}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_lane_u8(uint8_t *ptr, uint8x8_t val, const int lane)</code>	<code>val -> Vt.8B ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.b}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_u8(uint8_t *ptr, uint8x16_t val, const int lane)</code>	<code>val -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	ST1 {Vt.b}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_u16(uint16_t *ptr, uint16x4_t val, const int lane)</code>	<code>val -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST1 {Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_u16(uint16_t *ptr, uint16x8_t val, const int lane)</code>	<code>val -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_u32(uint32_t *ptr, uint32x2_t val, const int lane)</code>	<code>val -> Vt.2S ptr -> Xn 0 <= lane <= 1</code>	ST1 {Vt.s}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_lane_u32(uint32_t *ptr, uint32x4_t val, const int lane)</code>	<code>val -> Vt.4S ptr -> Xn 0 <= lane <= 3</code>	ST1 {Vt.s}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_u64(uint64_t *ptr, uint64x1_t val, const int lane)</code>	<code>val -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	ST1 {Vt.d}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_u64(uint64_t *ptr, uint64x2_t val, const int lane)</code>	<code>val -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	ST1 {Vt.d}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_p64(poly64_t *ptr, poly64x1_t val, const int lane)</code>	<code>val -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	ST1 {Vt.d}[lane],[Xn]		A32/A64
<code>void vst1q_lane_p64(poly64_t *ptr, poly64x2_t val, const int lane)</code>	<code>val -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	ST1 {Vt.d}[lane],[Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_lane_f16(float16_t *ptr, float16x4_t val, const int lane)</code>	<code>val -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST1 {Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_f16(float16_t *ptr, float16x8_t val, const int lane)</code>	<code>val -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_f32(float32_t *ptr, float32x2_t val, const int lane)</code>	<code>val -> Vt.2S ptr -> Xn 0 <= lane <= 1</code>	ST1 {Vt.s}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_f32(float32_t *ptr, float32x4_t val, const int lane)</code>	<code>val -> Vt.4S ptr -> Xn 0 <= lane <= 3</code>	ST1 {Vt.s}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_p8(poly8_t *ptr, poly8x8_t val, const int lane)</code>	<code>val -> Vt.8B ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.b}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_lane_p8(poly8_t *ptr, poly8x16_t val, const int lane)</code>	<code>val -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	ST1 {Vt.b}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_p16(poly16_t *ptr, poly16x4_t val, const int lane)</code>	<code>val -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST1 {Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst1q_lane_p16(poly16_t *ptr, poly16x8_t val, const int lane)</code>	<code>val -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst1_lane_f64(float64_t *ptr, float64x1_t val, const int lane)</code>	<code>val -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	ST1 {Vt.d}[lane],[Xn]		A64
<code>void vst1q_lane_f64(float64_t *ptr, float64x2_t val, const int lane)</code>	<code>val -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	ST1 {Vt.d}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_lane_mf8(mfloat8_t *ptr, mfloat8x8_t val, const int lane)</code>	<code>val -> Vt.8B ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.b}[lane],[Xn]		A64
<code>void vst1q_lane_mf8(mfloat8_t *ptr, mfloat8x16_t val, const int lane)</code>	<code>val -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	ST1 {Vt.b}[lane],[Xn]		A64
<code>void vst2_s8(int8_t *ptr, int8x8x2_t val)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST2 {Vt.8B - Vt2.8B},[Xn]		v7/A32/A64
<code>void vst2q_s8(int8_t *ptr, int8x16x2_t val)</code>	<code>val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST2 {Vt.16B - Vt2.16B},[Xn]		v7/A32/A64
<code>void vst2_s16(int16_t *ptr, int16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST2 {Vt.4H - Vt2.4H},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2q_s16(int16_t *ptr, int16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST2 {Vt.8H - Vt2.8H}, [Xn]		v7/A32/A64
<code>void vst2_s32(int32_t *ptr, int32x2x2_t val)</code>	<code>val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST2 {Vt.2S - Vt2.2S}, [Xn]		v7/A32/A64
<code>void vst2q_s32(int32_t *ptr, int32x4x2_t val)</code>	<code>val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST2 {Vt.4S - Vt2.4S}, [Xn]		v7/A32/A64
<code>void vst2_u8(uint8_t *ptr, uint8x8x2_t val)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST2 {Vt.8B - Vt2.8B}, [Xn]		v7/A32/A64
<code>void vst2q_u8(uint8_t *ptr, uint8x16x2_t val)</code>	<code>val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST2 {Vt.16B - Vt2.16B}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2_u16(uint16_t *ptr, uint16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST2 {Vt.4H - Vt2.4H}, [Xn]		v7/A32/A64
<code>void vst2q_u16(uint16_t *ptr, uint16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST2 {Vt.8H - Vt2.8H}, [Xn]		v7/A32/A64
<code>void vst2_u32(uint32_t *ptr, uint32x2x2_t val)</code>	<code>val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST2 {Vt.2S - Vt2.2S}, [Xn]		v7/A32/A64
<code>void vst2q_u32(uint32_t *ptr, uint32x4x2_t val)</code>	<code>val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST2 {Vt.4S - Vt2.4S}, [Xn]		v7/A32/A64
<code>void vst2_f16(float16_t *ptr, float16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST2 {Vt.4H - Vt2.4H}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2q_f16(float16_t *ptr, float16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST2 {Vt.8H - Vt2.8H}, [Xn]		v7/A32/A64
<code>void vst2_f32(float32_t *ptr, float32x2x2_t val)</code>	<code>val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST2 {Vt.2S - Vt2.2S}, [Xn]		v7/A32/A64
<code>void vst2q_f32(float32_t *ptr, float32x4x2_t val)</code>	<code>val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST2 {Vt.4S - Vt2.4S}, [Xn]		v7/A32/A64
<code>void vst2_p8(poly8_t *ptr, poly8x8x2_t val)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST2 {Vt.8B - Vt2.8B}, [Xn]		v7/A32/A64
<code>void vst2q_p8(poly8_t *ptr, poly8x16x2_t val)</code>	<code>val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST2 {Vt.16B - Vt2.16B}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2_p16(poly16_t *ptr, poly16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST2 {Vt.4H - Vt2.4H}, [Xn]		v7/A32/A64
<code>void vst2q_p16(poly16_t *ptr, poly16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST2 {Vt.8H - Vt2.8H}, [Xn]		v7/A32/A64
<code>void vst2_s64(int64_t *ptr, int64x1x2_t val)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt2.1D}, [Xn]		v7/A32/A64
<code>void vst2_u64(uint64_t *ptr, uint64x1x2_t val)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt2.1D}, [Xn]		v7/A32/A64
<code>void vst2_p64(poly64_t *ptr, poly64x1x2_t val)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt2.1D}, [Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2q_s64(int64_t *ptr, int64x2x2_t val)</code>	<code>val.val[1] -> Vt.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST2 {Vt.2D - Vt.2D},[Xn]		A64
<code>void vst2q_u64(uint64_t *ptr, uint64x2x2_t val)</code>	<code>val.val[1] -> Vt.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST2 {Vt.2D - Vt.2D},[Xn]		A64
<code>void vst2q_p64(poly64_t *ptr, poly64x2x2_t val)</code>	<code>val.val[1] -> Vt.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST2 {Vt.2D - Vt.2D},[Xn]		A64
<code>void vst2_f64(float64_t *ptr, float64x1x2_t val)</code>	<code>val.val[1] -> Vt.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt.1D},[Xn]		A64
<code>void vst2q_f64(float64_t *ptr, float64x2x2_t val)</code>	<code>val.val[1] -> Vt.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST2 {Vt.2D - Vt.2D},[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2_mf8(mfloat8_t *ptr, mfloat8x8x2_t val)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST2 {Vt.8B - Vt2.8B},[Xn]		A64
<code>void vst2q_mf8(mfloat8_t *ptr, mfloat8x16x2_t val)</code>	<code>val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST2 {Vt.16B - Vt2.16B},[Xn]		A64
<code>void vst3_s8(int8_t *ptr, int8x8x3_t val)</code>	<code>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST3 {Vt.8B - Vt3.8B},[Xn]		v7/A32/A64
<code>void vst3q_s8(int8_t *ptr, int8x16x3_t val)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST3 {Vt.16B - Vt3.16B},[Xn]		v7/A32/A64
<code>void vst3_s16(int16_t *ptr, int16x4x3_t val)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST3 {Vt.4H - Vt3.4H},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3q_s16(int16_t *ptr, int16x8x3_t val)</code>	val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn	ST3 {Vt.8H - Vt3.8H}, [Xn]		v7/A32/A64
<code>void vst3_s32(int32_t *ptr, int32x2x3_t val)</code>	val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn	ST3 {Vt.2S - Vt3.2S}, [Xn]		v7/A32/A64
<code>void vst3q_s32(int32_t *ptr, int32x4x3_t val)</code>	val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn	ST3 {Vt.4S - Vt3.4S}, [Xn]		v7/A32/A64
<code>void vst3_u8(uint8_t *ptr, uint8x8x3_t val)</code>	val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn	ST3 {Vt.8B - Vt3.8B}, [Xn]		v7/A32/A64
<code>void vst3q_u8(uint8_t *ptr, uint8x16x3_t val)</code>	val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn	ST3 {Vt.16B - Vt3.16B}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3_u16(uint16_t *ptr, uint16x4x3_t val)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST3 {Vt.4H - Vt3.4H}, [Xn]		v7/A32/A64
<code>void vst3q_u16(uint16_t *ptr, uint16x8x3_t val)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST3 {Vt.8H - Vt3.8H}, [Xn]		v7/A32/A64
<code>void vst3_u32(uint32_t *ptr, uint32x2x3_t val)</code>	<code>val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST3 {Vt.2S - Vt3.2S}, [Xn]		v7/A32/A64
<code>void vst3q_u32(uint32_t *ptr, uint32x4x3_t val)</code>	<code>val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST3 {Vt.4S - Vt3.4S}, [Xn]		v7/A32/A64
<code>void vst3_f16(float16_t *ptr, float16x4x3_t val)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST3 {Vt.4H - Vt3.4H}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3q_f16(float16_t *ptr, float16x8x3_t val)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST3 {Vt.8H - Vt3.8H}, [Xn]		v7/A32/A64
<code>void vst3_f32(float32_t *ptr, float32x2x3_t val)</code>	<code>val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST3 {Vt.2S - Vt3.2S}, [Xn]		v7/A32/A64
<code>void vst3q_f32(float32_t *ptr, float32x4x3_t val)</code>	<code>val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST3 {Vt.4S - Vt3.4S}, [Xn]		v7/A32/A64
<code>void vst3_p8(poly8_t *ptr, poly8x8x3_t val)</code>	<code>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST3 {Vt.8B - Vt3.8B}, [Xn]		v7/A32/A64
<code>void vst3q_p8(poly8_t *ptr, poly8x16x3_t val)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST3 {Vt.16B - Vt3.16B}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3_p16(poly16_t *ptr, poly16x4x3_t val)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST3 {Vt.4H - Vt3.4H}, [Xn]		v7/A32/A64
<code>void vst3q_p16(poly16_t *ptr, poly16x8x3_t val)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST3 {Vt.8H - Vt3.8H}, [Xn]		v7/A32/A64
<code>void vst3_s64(int64_t *ptr, int64x1x3_t val)</code>	<code>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt3.1D}, [Xn]		v7/A32/A64
<code>void vst3_u64(uint64_t *ptr, uint64x1x3_t val)</code>	<code>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt3.1D}, [Xn]		v7/A32/A64
<code>void vst3_p64(poly64_t *ptr, poly64x1x3_t val)</code>	<code>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt3.1D}, [Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3q_s64(int64_t *ptr, int64x2x3_t val)</code>	val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST3 {Vt.2D - Vt3.2D}, [Xn]		A64
<code>void vst3q_u64(uint64_t *ptr, uint64x2x3_t val)</code>	val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST3 {Vt.2D - Vt3.2D}, [Xn]		A64
<code>void vst3q_p64(poly64_t *ptr, poly64x2x3_t val)</code>	val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST3 {Vt.2D - Vt3.2D}, [Xn]		A64
<code>void vst3_f64(float64_t *ptr, float64x1x3_t val)</code>	val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn	ST1 {Vt.1D - Vt3.1D}, [Xn]		A64
<code>void vst3q_f64(float64_t *ptr, float64x2x3_t val)</code>	val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST3 {Vt.2D - Vt3.2D}, [Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3_mf8(mfloat8_t *ptr, mfloat8x8x3_t val)</code>	<code>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST3 {Vt.8B - Vt3.8B},[Xn]		A64
<code>void vst3q_mf8(mfloat8_t *ptr, mfloat8x16x3_t val)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST3 {Vt.16B - Vt3.16B},[Xn]		A64
<code>void vst4_s8(int8_t *ptr, int8x8x4_t val)</code>	<code>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST4 {Vt.8B - Vt4.8B},[Xn]		v7/A32/A64
<code>void vst4q_s8(int8_t *ptr, int8x16x4_t val)</code>	<code>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST4 {Vt.16B - Vt4.16B},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_s16(int16_t *ptr, int16x4x4_t val)</pre>	<pre>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</pre>	ST4 {Vt.4H - Vt4.4H},[Xn]		v7/A32/A64
<pre>void vst4q_s16(int16_t *ptr, int16x8x4_t val)</pre>	<pre>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</pre>	ST4 {Vt.8H - Vt4.8H},[Xn]		v7/A32/A64
<pre>void vst4_s32(int32_t *ptr, int32x2x4_t val)</pre>	<pre>val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</pre>	ST4 {Vt.2S - Vt4.2S},[Xn]		v7/A32/A64
<pre>void vst4q_s32(int32_t *ptr, int32x4x4_t val)</pre>	<pre>val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</pre>	ST4 {Vt.4S - Vt4.4S},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_u8(uint8_t *ptr, uint8x8x4_t val)</pre>	<pre>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</pre>	ST4 {Vt.8B - Vt4.8B},[Xn]		v7/A32/A64
<pre>void vst4q_u8(uint8_t *ptr, uint8x16x4_t val)</pre>	<pre>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</pre>	ST4 {Vt.16B - Vt4.16B},[Xn]		v7/A32/A64
<pre>void vst4_u16(uint16_t *ptr, uint16x4x4_t val)</pre>	<pre>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</pre>	ST4 {Vt.4H - Vt4.4H},[Xn]		v7/A32/A64
<pre>void vst4q_u16(uint16_t *ptr, uint16x8x4_t val)</pre>	<pre>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</pre>	ST4 {Vt.8H - Vt4.8H},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_u32(uint32_t *ptr, uint32x2x4_t val)</pre>	<pre>val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</pre>	ST4 {Vt.2S - Vt4.2S}, [Xn]		v7/A32/A64
<pre>void vst4q_u32(uint32_t *ptr, uint32x4x4_t val)</pre>	<pre>val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</pre>	ST4 {Vt.4S - Vt4.4S}, [Xn]		v7/A32/A64
<pre>void vst4_f16(float16_t *ptr, float16x4x4_t val)</pre>	<pre>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</pre>	ST4 {Vt.4H - Vt4.4H}, [Xn]		v7/A32/A64
<pre>void vst4q_f16(float16_t *ptr, float16x8x4_t val)</pre>	<pre>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</pre>	ST4 {Vt.8H - Vt4.8H}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst4_f32(float32_t *ptr, float32x4_t val)</code>	<code>val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST4 {Vt.2S - Vt4.2S},[Xn]		v7/A32/A64
<code>void vst4q_f32(float32_t *ptr, float32x4x4_t val)</code>	<code>val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST4 {Vt.4S - Vt4.4S},[Xn]		v7/A32/A64
<code>void vst4_p8(poly8_t *ptr, poly8x8x4_t val)</code>	<code>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST4 {Vt.8B - Vt4.8B},[Xn]		v7/A32/A64
<code>void vst4q_p8(poly8_t *ptr, poly8x16x4_t val)</code>	<code>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST4 {Vt.16B - Vt4.16B},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst4_p16(poly16_t *ptr, poly16x4x4_t val)</code>	<code>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST4 {Vt.4H - Vt4.4H},[Xn]		v7/A32/A64
<code>void vst4q_p16(poly16_t *ptr, poly16x8x4_t val)</code>	<code>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST4 {Vt.8H - Vt4.8H},[Xn]		v7/A32/A64
<code>void vst4_s64(int64_t *ptr, int64x1x4_t val)</code>	<code>val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt4.1D},[Xn]		v7/A32/A64
<code>void vst4_u64(uint64_t *ptr, uint64x1x4_t val)</code>	<code>val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt4.1D},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst4_p64(poly64_t *ptr, poly64x1x4_t val)</code>	val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn	ST1 {Vt.1D - Vt4.1D},[Xn]		A32/A64
<code>void vst4q_s64(int64_t *ptr, int64x2x4_t val)</code>	val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST4 {Vt.2D - Vt4.2D},[Xn]		A64
<code>void vst4q_u64(uint64_t *ptr, uint64x2x4_t val)</code>	val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST4 {Vt.2D - Vt4.2D},[Xn]		A64
<code>void vst4q_p64(poly64_t *ptr, poly64x2x4_t val)</code>	val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST4 {Vt.2D - Vt4.2D},[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst4_f64(float64_t *ptr, float64x1x4_t val)</code>	<code>val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt4.1D},[Xn]		A64
<code>void vst4q_f64(float64_t *ptr, float64x2x4_t val)</code>	<code>val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST4 {Vt.2D - Vt4.2D},[Xn]		A64
<code>void vst4_mf8(mfloat8_t *ptr, mfloat8x8x4_t val)</code>	<code>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST4 {Vt.8B - Vt4.8B},[Xn]		A64
<code>void vst4q_mf8(mfloat8_t *ptr, mfloat8x16x4_t val)</code>	<code>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST4 {Vt.16B - Vt4.16B},[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2_lane_s8(int8_t *ptr, int8x8x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</code>	ST2 {Vt.b - Vt2.b}[lane],[Xn]		v7/A32/A64
<code>void vst2_lane_u8(uint8_t *ptr, uint8x8x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</code>	ST2 {Vt.b - Vt2.b}[lane],[Xn]		v7/A32/A64
<code>void vst2_lane_p8(poly8_t *ptr, poly8x8x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</code>	ST2 {Vt.b - Vt2.b}[lane],[Xn]		v7/A32/A64
<code>void vst2_lane_mf8(mfloat8_t *ptr, mfloat8x8x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</code>	ST2 {Vt.b - Vt2.b}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst3_lane_s8(int8_t *ptr, int8x8x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</pre>	ST3 {Vt.b - Vt3.b}[lane],[Xn]		v7/A32/A64
<pre>void vst3_lane_u8(uint8_t *ptr, uint8x8x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</pre>	ST3 {Vt.b - Vt3.b}[lane],[Xn]		v7/A32/A64
<pre>void vst3_lane_p8(poly8_t *ptr, poly8x8x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</pre>	ST3 {Vt.b - Vt3.b}[lane],[Xn]		v7/A32/A64
<pre>void vst3_lane_mf8(mfloat8_t *ptr, mfloat8x8x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</pre>	ST3 {Vt.b - Vt3.b}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_lane_s8(int8_t *ptr, int8x8x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</pre>	ST4 {Vt.b - Vt4.b}[lane],[Xn]		v7/A32/A64
<pre>void vst4_lane_u8(uint8_t *ptr, uint8x8x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</pre>	ST4 {Vt.b - Vt4.b}[lane],[Xn]		v7/A32/A64
<pre>void vst4_lane_p8(poly8_t *ptr, poly8x8x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</pre>	ST4 {Vt.b - Vt4.b}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_lane_mf8(mfloat8_t *ptr, mfloat8x8x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn 0 <= lane <= 7</pre>	ST4 {Vt.b - Vt4.b}[lane],[Xn]		A64
<pre>void vst2_lane_s16(int16_t *ptr, int16x4x2_t val, const int lane)</pre>	<pre>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</pre>	ST2 {Vt.h - Vt2.h}[lane],[Xn]		v7/A32/A64
<pre>void vst2q_lane_s16(int16_t *ptr, int16x8x2_t val, const int lane)</pre>	<pre>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</pre>	ST2 {Vt.h - Vt2.h}[lane],[Xn]		v7/A32/A64
<pre>void vst2_lane_s32(int32_t *ptr, int32x2x2_t val, const int lane)</pre>	<pre>val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</pre>	ST2 {Vt.s - Vt2.s}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2q_lane_s32(int32_t *ptr, int32x4x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</code>	ST2 {Vt.s - Vt2.s}[lane],[Xn]		v7/A32/A64
<code>void vst2_lane_u16(uint16_t *ptr, uint16x4x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST2 {Vt.h - Vt2.h}[lane],[Xn]		v7/A32/A64
<code>void vst2q_lane_u16(uint16_t *ptr, uint16x8x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST2 {Vt.h - Vt2.h}[lane],[Xn]		v7/A32/A64
<code>void vst2_lane_u32(uint32_t *ptr, uint32x2x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</code>	ST2 {Vt.s - Vt2.s}[lane],[Xn]		v7/A32/A64
<code>void vst2q_lane_u32(uint32_t *ptr, uint32x4x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</code>	ST2 {Vt.s - Vt2.s}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2_lane_f16(float16_t *ptr, float16x4x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST2 {Vt.h - Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst2q_lane_f16(float16_t *ptr, float16x8x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST2 {Vt.h - Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst2_lane_f32(float32_t *ptr, float32x2x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</code>	ST2 {Vt.s - Vt.s}[lane],[Xn]		v7/A32/A64
<code>void vst2q_lane_f32(float32_t *ptr, float32x4x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</code>	ST2 {Vt.s - Vt.s}[lane],[Xn]		v7/A32/A64
<code>void vst2_lane_p16(poly16_t *ptr, poly16x4x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST2 {Vt.h - Vt.h}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2q_lane_p16(poly16_t *ptr, poly16x8x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST2 {Vt.h - Vt.h}[lane],[Xn]		v7/A32/A64
<code>void vst2q_lane_s8(int8_t *ptr, int8x16x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	ST2 {Vt.b - Vt.b}[lane],[Xn]		A64
<code>void vst2q_lane_u8(uint8_t *ptr, uint8x16x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	ST2 {Vt.b - Vt.b}[lane],[Xn]		A64
<code>void vst2q_lane_p8(poly8_t *ptr, poly8x16x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	ST2 {Vt.b - Vt.b}[lane],[Xn]		A64
<code>void vst2q_lane_mf8(mfloat8_t *ptr, mfloat8x16x2_t val, const int lane)</code>	<code>val.val[1] -> Vt.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	ST2 {Vt.b - Vt.b}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2_lane_s64(int64_t *ptr, int64x1x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	ST2 {Vt.d - Vt2.d}[lane],[Xn]		A64
<code>void vst2q_lane_s64(int64_t *ptr, int64x2x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	ST2 {Vt.d - Vt2.d}[lane],[Xn]		A64
<code>void vst2_lane_u64(uint64_t *ptr, uint64x1x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	ST2 {Vt.d - Vt2.d}[lane],[Xn]		A64
<code>void vst2q_lane_u64(uint64_t *ptr, uint64x2x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	ST2 {Vt.d - Vt2.d}[lane],[Xn]		A64
<code>void vst2_lane_p64(poly64_t *ptr, poly64x1x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	ST2 {Vt.d - Vt2.d}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2q_lane_p64(poly64_t *ptr, poly64x2x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	ST2 {Vt.d - Vt2.d}[lane],[Xn]		A64
<code>void vst2_lane_f64(float64_t *ptr, float64x1x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	ST2 {Vt.d - Vt2.d}[lane],[Xn]		A64
<code>void vst2q_lane_f64(float64_t *ptr, float64x2x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	ST2 {Vt.d - Vt2.d}[lane],[Xn]		A64
<code>void vst3_lane_s16(int16_t *ptr, int16x4x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST3 {Vt.h - Vt3.h}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst3q_lane_s16(int16_t *ptr, int16x8x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</pre>	ST3 {Vt.h - Vt3.h}[lane],[Xn]		v7/A32/A64
<pre>void vst3_lane_s32(int32_t *ptr, int32x2x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</pre>	ST3 {Vt.s - Vt3.s}[lane],[Xn]		v7/A32/A64
<pre>void vst3q_lane_s32(int32_t *ptr, int32x4x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</pre>	ST3 {Vt.s - Vt3.s}[lane],[Xn]		v7/A32/A64
<pre>void vst3_lane_u16(uint16_t *ptr, uint16x4x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</pre>	ST3 {Vt.h - Vt3.h}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst3q_lane_u16(uint16_t *ptr, uint16x8x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</pre>	ST3 {Vt.h - Vt3.h}[lane],[Xn]		v7/A32/A64
<pre>void vst3_lane_u32(uint32_t *ptr, uint32x2x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</pre>	ST3 {Vt.s - Vt3.s}[lane],[Xn]		v7/A32/A64
<pre>void vst3q_lane_u32(uint32_t *ptr, uint32x4x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</pre>	ST3 {Vt.s - Vt3.s}[lane],[Xn]		v7/A32/A64
<pre>void vst3_lane_f16(float16_t *ptr, float16x4x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</pre>	ST3 {Vt.h - Vt3.h}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3q_lane_f16(float16_t *ptr, float16x8x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	<code>ST3 {Vt.h - Vt3.h}[lane],[Xn]</code>		v7/A32/A64
<code>void vst3_lane_f32(float32_t *ptr, float32x2x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</code>	<code>ST3 {Vt.s - Vt3.s}[lane],[Xn]</code>		v7/A32/A64
<code>void vst3q_lane_f32(float32_t *ptr, float32x4x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</code>	<code>ST3 {Vt.s - Vt3.s}[lane],[Xn]</code>		v7/A32/A64
<code>void vst3_lane_p16(poly16_t *ptr, poly16x4x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	<code>ST3 {Vt.h - Vt3.h}[lane],[Xn]</code>		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3q_lane_p16(poly16_t *ptr, poly16x8x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	<code>ST3 {Vt.h - Vt3.h}[lane],[Xn]</code>		v7/A32/A64
<code>void vst3q_lane_s8(int8_t *ptr, int8x16x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	<code>ST3 {Vt.b - Vt3.b}[lane],[Xn]</code>		v7/A32/A64
<code>void vst3q_lane_u8(uint8_t *ptr, uint8x16x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	<code>ST3 {Vt.b - Vt3.b}[lane],[Xn]</code>		v7/A32/A64
<code>void vst3q_lane_p8(poly8_t *ptr, poly8x16x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</code>	<code>ST3 {Vt.b - Vt3.b}[lane],[Xn]</code>		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst3_lane_s64(int64_t *ptr, int64x1x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	ST3 {Vt.d - Vt3.d}[lane],[Xn]		A64
<pre>void vst3q_lane_s64(int64_t *ptr, int64x2x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</pre>	ST3 {Vt.d - Vt3.d}[lane],[Xn]		A64
<pre>void vst3_lane_u64(uint64_t *ptr, uint64x1x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	ST3 {Vt.d - Vt3.d}[lane],[Xn]		A64
<pre>void vst3q_lane_u64(uint64_t *ptr, uint64x2x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</pre>	ST3 {Vt.d - Vt3.d}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3_lane_p64(poly64_t *ptr, poly64x1x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	<code>ST3 {Vt.d - Vt3.d}[lane],[Xn]</code>		A64
<code>void vst3q_lane_p64(poly64_t *ptr, poly64x2x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	<code>ST3 {Vt.d - Vt3.d}[lane],[Xn]</code>		A64
<code>void vst3_lane_f64(float64_t *ptr, float64x1x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</code>	<code>ST3 {Vt.d - Vt3.d}[lane],[Xn]</code>		A64
<code>void vst3q_lane_f64(float64_t *ptr, float64x2x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</code>	<code>ST3 {Vt.d - Vt3.d}[lane],[Xn]</code>		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst3q_lane_mf8(mfloat8_t *ptr, mfloat8x16x3_t val, const int lane)</pre>	<pre>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</pre>	ST3 {Vt.b - Vt3.b}[lane],[Xn]		A64
<pre>void vst4_lane_s16(int16_t *ptr, int16x4x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</pre>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		v7/A32/A64
<pre>void vst4q_lane_s16(int16_t *ptr, int16x8x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</pre>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_lane_s32(int32_t *ptr, int32x2x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</pre>	ST4 {Vt.s - Vt4.s}[lane],[Xn]		v7/A32/A64
<pre>void vst4q_lane_s32(int32_t *ptr, int32x4x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</pre>	ST4 {Vt.s - Vt4.s}[lane],[Xn]		v7/A32/A64
<pre>void vst4_lane_u16(uint16_t *ptr, uint16x4x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</pre>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4q_lane_u16(uint16_t *ptr, uint16x8x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</pre>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		v7/A32/A64
<pre>void vst4_lane_u32(uint32_t *ptr, uint32x2x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</pre>	ST4 {Vt.s - Vt4.s}[lane],[Xn]		v7/A32/A64
<pre>void vst4q_lane_u32(uint32_t *ptr, uint32x4x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</pre>	ST4 {Vt.s - Vt4.s}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_lane_f16(float16_t *ptr, float16x4x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</pre>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		v7/A32/A64
<pre>void vst4q_lane_f16(float16_t *ptr, float16x8x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</pre>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		v7/A32/A64
<pre>void vst4_lane_f32(float32_t *ptr, float32x2x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn 0 <= lane <= 1</pre>	ST4 {Vt.s - Vt4.s}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4q_lane_f32(float32_t *ptr, float32x4x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn 0 <= lane <= 3</pre>	ST4 {Vt.s - Vt4.s}[lane],[Xn]		v7/A32/A64
<pre>void vst4_lane_p16(poly16_t *ptr, poly16x4x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</pre>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		v7/A32/A64
<pre>void vst4q_lane_p16(poly16_t *ptr, poly16x8x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</pre>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4q_lane_s8(int8_t *ptr, int8x16x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</pre>	ST4 {Vt.b - Vt4.b}[lane],[Xn]		A64
<pre>void vst4q_lane_u8(uint8_t *ptr, uint8x16x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</pre>	ST4 {Vt.b - Vt4.b}[lane],[Xn]		A64
<pre>void vst4q_lane_p8(poly8_t *ptr, poly8x16x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</pre>	ST4 {Vt.b - Vt4.b}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_lane_s64(int64_t *ptr, int64x1x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	ST4 {Vt.d - Vt4.d}[lane],[Xn]		A64
<pre>void vst4q_lane_s64(int64_t *ptr, int64x2x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</pre>	ST4 {Vt.d - Vt4.d}[lane],[Xn]		A64
<pre>void vst4_lane_u64(uint64_t *ptr, uint64x1x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	ST4 {Vt.d - Vt4.d}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4q_lane_u64(uint64_t *ptr, uint64x2x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</pre>	ST4 {Vt.d - Vt4.d}[lane],[Xn]		A64
<pre>void vst4_lane_p64(poly64_t *ptr, poly64x1x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	ST4 {Vt.d - Vt4.d}[lane],[Xn]		A64
<pre>void vst4q_lane_p64(poly64_t *ptr, poly64x2x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</pre>	ST4 {Vt.d - Vt4.d}[lane],[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst4_lane_f64(float64_t *ptr, float64x1x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn 0 <= lane <= 0</pre>	ST4 {Vt.d - Vt4.d}[lane],[Xn]		A64
<pre>void vst4q_lane_f64(float64_t *ptr, float64x2x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn 0 <= lane <= 1</pre>	ST4 {Vt.d - Vt4.d}[lane],[Xn]		A64
<pre>void vst4q_lane_mf8(mfloat8_t *ptr, mfloat8x16x4_t val, const int lane)</pre>	<pre>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn 0 <= lane <= 15</pre>	ST4 {Vt.b - Vt4.b}[lane],[Xn]		A64
<pre>void vst1_s8_x2(int8_t *ptr, int8x8x2_t val)</pre>	<pre>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</pre>	ST1 {Vt.8B - Vt2.8B},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_s8_x2(int8_t *ptr, int8x16x2_t val)</code>	val.val[1] -> Vt.16B val.val[0] -> Vt.16B ptr -> Xn	ST1 {Vt.16B - Vt.16B},[Xn]		v7/A32/A64
<code>void vst1_s16_x2(int16_t *ptr, int16x4x2_t val)</code>	val.val[1] -> Vt.4H val.val[0] -> Vt.4H ptr -> Xn	ST1 {Vt.4H - Vt.4H},[Xn]		v7/A32/A64
<code>void vst1q_s16_x2(int16_t *ptr, int16x8x2_t val)</code>	val.val[1] -> Vt.8H val.val[0] -> Vt.8H ptr -> Xn	ST1 {Vt.8H - Vt.8H},[Xn]		v7/A32/A64
<code>void vst1_s32_x2(int32_t *ptr, int32x2x2_t val)</code>	val.val[1] -> Vt.2S val.val[0] -> Vt.2S ptr -> Xn	ST1 {Vt.2S - Vt.2S},[Xn]		v7/A32/A64
<code>void vst1q_s32_x2(int32_t *ptr, int32x4x2_t val)</code>	val.val[1] -> Vt.4S val.val[0] -> Vt.4S ptr -> Xn	ST1 {Vt.4S - Vt.4S},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_u8_x2(uint8_t *ptr, uint8x8x2_t val)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt2.8B},[Xn]		v7/A32/A64
<code>void vst1q_u8_x2(uint8_t *ptr, uint8x16x2_t val)</code>	<code>val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B - Vt2.16B},[Xn]		v7/A32/A64
<code>void vst1_u16_x2(uint16_t *ptr, uint16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt2.4H},[Xn]		v7/A32/A64
<code>void vst1q_u16_x2(uint16_t *ptr, uint16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt2.8H},[Xn]		v7/A32/A64
<code>void vst1_u32_x2(uint32_t *ptr, uint32x2x2_t val)</code>	<code>val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST1 {Vt.2S - Vt2.2S},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_u32_x2(uint32_t *ptr, uint32x4x2_t val)</code>	<code>val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST1 {Vt.4S - Vt2.4S}, [Xn]		v7/A32/A64
<code>void vst1_f16_x2(float16_t *ptr, float16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt2.4H}, [Xn]		v7/A32/A64
<code>void vst1q_f16_x2(float16_t *ptr, float16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt2.8H}, [Xn]		v7/A32/A64
<code>void vst1_f32_x2(float32_t *ptr, float32x2x2_t val)</code>	<code>val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST1 {Vt.2S - Vt2.2S}, [Xn]		v7/A32/A64
<code>void vst1q_f32_x2(float32_t *ptr, float32x4x2_t val)</code>	<code>val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST1 {Vt.4S - Vt2.4S}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_p8_x2(poly8_t *ptr, poly8x8x2_t val)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt2.8B},[Xn]		v7/A32/A64
<code>void vst1q_p8_x2(poly8_t *ptr, poly8x16x2_t val)</code>	<code>val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B - Vt2.16B},[Xn]		v7/A32/A64
<code>void vst1_p16_x2(poly16_t *ptr, poly16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt2.4H},[Xn]		v7/A32/A64
<code>void vst1q_p16_x2(poly16_t *ptr, poly16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt2.8H},[Xn]		v7/A32/A64
<code>void vst1_s64_x2(int64_t *ptr, int64x1x2_t val)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt2.1D},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_u64_x2(uint64_t *ptr, uint64x1x2_t val)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt2.1D},[Xn]		v7/A32/A64
<code>void vst1_p64_x2(poly64_t *ptr, poly64x1x2_t val)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt2.1D},[Xn]		A32/A64
<code>void vst1q_s64_x2(int64_t *ptr, int64x2x2_t val)</code>	<code>val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D - Vt2.2D},[Xn]		v7/A32/A64
<code>void vst1q_u64_x2(uint64_t *ptr, uint64x2x2_t val)</code>	<code>val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D - Vt2.2D},[Xn]		v7/A32/A64
<code>void vst1q_p64_x2(poly64_t *ptr, poly64x2x2_t val)</code>	<code>val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D - Vt2.2D},[Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_f64_x2(float64_t *ptr, float64x1x2_t val)</code>	<code>val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt2.1D},[Xn]		A64
<code>void vst1q_f64_x2(float64_t *ptr, float64x2x2_t val)</code>	<code>val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D - Vt2.2D},[Xn]		A64
<code>void vst1_mf8_x2(mfloat8_t *ptr, mfloat8x8x2_t val)</code>	<code>val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt2.8B},[Xn]		A64
<code>void vst1q_mf8_x2(mfloat8_t *ptr, mfloat8x16x2_t val)</code>	<code>val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B - Vt2.16B},[Xn]		A64
<code>void vst1_s8_x3(int8_t *ptr, int8x8x3_t val)</code>	<code>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt3.8B},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_s8_x3(int8_t *ptr, int8x16x3_t val)</code>	val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn	ST1 {Vt.16B - Vt3.16B},[Xn]		v7/A32/A64
<code>void vst1_s16_x3(int16_t *ptr, int16x4x3_t val)</code>	val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn	ST1 {Vt.4H - Vt3.4H},[Xn]		v7/A32/A64
<code>void vst1q_s16_x3(int16_t *ptr, int16x8x3_t val)</code>	val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn	ST1 {Vt.8H - Vt3.8H},[Xn]		v7/A32/A64
<code>void vst1_s32_x3(int32_t *ptr, int32x2x3_t val)</code>	val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn	ST1 {Vt.2S - Vt3.2S},[Xn]		v7/A32/A64
<code>void vst1q_s32_x3(int32_t *ptr, int32x4x3_t val)</code>	val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn	ST1 {Vt.4S - Vt3.4S},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_u8_x3(uint8_t *ptr, uint8x8x3_t val)</code>	<code>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt3.8B},[Xn]		v7/A32/A64
<code>void vst1q_u8_x3(uint8_t *ptr, uint8x16x3_t val)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B - Vt3.16B},[Xn]		v7/A32/A64
<code>void vst1_u16_x3(uint16_t *ptr, uint16x4x3_t val)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt3.4H},[Xn]		v7/A32/A64
<code>void vst1q_u16_x3(uint16_t *ptr, uint16x8x3_t val)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt3.8H},[Xn]		v7/A32/A64
<code>void vst1_u32_x3(uint32_t *ptr, uint32x2x3_t val)</code>	<code>val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST1 {Vt.2S - Vt3.2S},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_u32_x3(uint32_t *ptr, uint32x4x3_t val)</code>	val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn	ST1 {Vt.4S - Vt3.4S}, [Xn]		v7/A32/A64
<code>void vst1_f16_x3(float16_t *ptr, float16x4x3_t val)</code>	val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn	ST1 {Vt.4H - Vt3.4H}, [Xn]		v7/A32/A64
<code>void vst1q_f16_x3(float16_t *ptr, float16x8x3_t val)</code>	val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn	ST1 {Vt.8H - Vt3.8H}, [Xn]		v7/A32/A64
<code>void vst1_f32_x3(float32_t *ptr, float32x2x3_t val)</code>	val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn	ST1 {Vt.2S - Vt3.2S}, [Xn]		v7/A32/A64
<code>void vst1q_f32_x3(float32_t *ptr, float32x4x3_t val)</code>	val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn	ST1 {Vt.4S - Vt3.4S}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_p8_x3(poly8_t *ptr, poly8x8x3_t val)</code>	<code>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt3.8B},[Xn]		v7/A32/A64
<code>void vst1q_p8_x3(poly8_t *ptr, poly8x16x3_t val)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B - Vt3.16B},[Xn]		v7/A32/A64
<code>void vst1_p16_x3(poly16_t *ptr, poly16x4x3_t val)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt3.4H},[Xn]		v7/A32/A64
<code>void vst1q_p16_x3(poly16_t *ptr, poly16x8x3_t val)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt3.8H},[Xn]		v7/A32/A64
<code>void vst1_s64_x3(int64_t *ptr, int64x1x3_t val)</code>	<code>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt3.1D},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_u64_x3(uint64_t *ptr, uint64x1x3_t val)</code>	val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn	ST1 {Vt.1D - Vt3.1D}, [Xn]		v7/A32/A64
<code>void vst1_p64_x3(poly64_t *ptr, poly64x1x3_t val)</code>	val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn	ST1 {Vt.1D - Vt3.1D}, [Xn]		A32/A64
<code>void vst1q_s64_x3(int64_t *ptr, int64x2x3_t val)</code>	val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST1 {Vt.2D - Vt3.2D}, [Xn]		v7/A32/A64
<code>void vst1q_u64_x3(uint64_t *ptr, uint64x2x3_t val)</code>	val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST1 {Vt.2D - Vt3.2D}, [Xn]		v7/A32/A64
<code>void vst1q_p64_x3(poly64_t *ptr, poly64x2x3_t val)</code>	val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST1 {Vt.2D - Vt3.2D}, [Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_f64_x3(float64_t *ptr, float64x1x3_t val)</code>	<code>val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn</code>	ST1 {Vt.1D - Vt3.1D},[Xn]		A64
<code>void vst1q_f64_x3(float64_t *ptr, float64x2x3_t val)</code>	<code>val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn</code>	ST1 {Vt.2D - Vt3.2D},[Xn]		A64
<code>void vst1_mf8_x3(mfloat8_t *ptr, mfloat8x8x3_t val)</code>	<code>val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt3.8B},[Xn]		A64
<code>void vst1q_mf8_x3(mfloat8_t *ptr, mfloat8x16x3_t val)</code>	<code>val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B - Vt3.16B},[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>void vst1_s8_x4(int8_t *ptr, int8x8x4_t val)</pre>	<pre>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</pre>	ST1 {Vt.8B - Vt4.8B},[Xn]		v7/A32/A64
<pre>void vst1q_s8_x4(int8_t *ptr, int8x16x4_t val)</pre>	<pre>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</pre>	ST1 {Vt.16B - Vt4.16B},[Xn]		v7/A32/A64
<pre>void vst1_s16_x4(int16_t *ptr, int16x4x4_t val)</pre>	<pre>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</pre>	ST1 {Vt.4H - Vt4.4H},[Xn]		v7/A32/A64
<pre>void vst1q_s16_x4(int16_t *ptr, int16x8x4_t val)</pre>	<pre>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</pre>	ST1 {Vt.8H - Vt4.8H},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_s32_x4(int32_t *ptr, int32x2x4_t val)</code>	val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn	ST1 {Vt.2S - Vt4.2S},[Xn]		v7/A32/A64
<code>void vst1q_s32_x4(int32_t *ptr, int32x4x4_t val)</code>	val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn	ST1 {Vt.4S - Vt4.4S},[Xn]		v7/A32/A64
<code>void vst1_u8_x4(uint8_t *ptr, uint8x8x4_t val)</code>	val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn	ST1 {Vt.8B - Vt4.8B},[Xn]		v7/A32/A64
<code>void vst1q_u8_x4(uint8_t *ptr, uint8x16x4_t val)</code>	val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn	ST1 {Vt.16B - Vt4.16B},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_u16_x4(uint16_t *ptr, uint16x4x4_t val)</code>	val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn	ST1 {Vt.4H - Vt4.4H},[Xn]		v7/A32/A64
<code>void vst1q_u16_x4(uint16_t *ptr, uint16x8x4_t val)</code>	val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn	ST1 {Vt.8H - Vt4.8H},[Xn]		v7/A32/A64
<code>void vst1_u32_x4(uint32_t *ptr, uint32x2x4_t val)</code>	val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn	ST1 {Vt.2S - Vt4.2S},[Xn]		v7/A32/A64
<code>void vst1q_u32_x4(uint32_t *ptr, uint32x4x4_t val)</code>	val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn	ST1 {Vt.4S - Vt4.4S},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_f16_x4(float16_t *ptr, float16x4x4_t val)</code>	<code>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt4.4H},[Xn]		v7/A32/A64
<code>void vst1q_f16_x4(float16_t *ptr, float16x8x4_t val)</code>	<code>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt4.8H},[Xn]		v7/A32/A64
<code>void vst1_f32_x4(float32_t *ptr, float32x2x4_t val)</code>	<code>val.val[3] -> Vt4.2S val.val[2] -> Vt3.2S val.val[1] -> Vt2.2S val.val[0] -> Vt.2S ptr -> Xn</code>	ST1 {Vt.2S - Vt4.2S},[Xn]		v7/A32/A64
<code>void vst1q_f32_x4(float32_t *ptr, float32x4x4_t val)</code>	<code>val.val[3] -> Vt4.4S val.val[2] -> Vt3.4S val.val[1] -> Vt2.4S val.val[0] -> Vt.4S ptr -> Xn</code>	ST1 {Vt.4S - Vt4.4S},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_p8_x4(poly8_t *ptr, poly8x8x4_t val)</code>	<code>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt4.8B},[Xn]		v7/A32/A64
<code>void vst1q_p8_x4(poly8_t *ptr, poly8x16x4_t val)</code>	<code>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B - Vt4.16B},[Xn]		v7/A32/A64
<code>void vst1_p16_x4(poly16_t *ptr, poly16x4x4_t val)</code>	<code>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt4.4H},[Xn]		v7/A32/A64
<code>void vst1q_p16_x4(poly16_t *ptr, poly16x8x4_t val)</code>	<code>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt4.8H},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_s64_x4(int64_t *ptr, int64x1x4_t val)</code>	val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn	ST1 {Vt.1D - Vt4.1D},[Xn]		v7/A32/A64
<code>void vst1_u64_x4(uint64_t *ptr, uint64x1x4_t val)</code>	val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn	ST1 {Vt.1D - Vt4.1D},[Xn]		v7/A32/A64
<code>void vst1_p64_x4(poly64_t *ptr, poly64x1x4_t val)</code>	val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn	ST1 {Vt.1D - Vt4.1D},[Xn]		A32/A64
<code>void vst1q_s64_x4(int64_t *ptr, int64x2x4_t val)</code>	val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST1 {Vt.2D - Vt4.2D},[Xn]		v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_u64_x4(uint64_t *ptr, uint64x2x4_t val)</code>	val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST1 {Vt.2D - Vt4.2D},[Xn]		v7/A32/A64
<code>void vst1q_p64_x4(poly64_t *ptr, poly64x2x4_t val)</code>	val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST1 {Vt.2D - Vt4.2D},[Xn]		A32/A64
<code>void vst1_f64_x4(float64_t *ptr, float64x1x4_t val)</code>	val.val[3] -> Vt4.1D val.val[2] -> Vt3.1D val.val[1] -> Vt2.1D val.val[0] -> Vt.1D ptr -> Xn	ST1 {Vt.1D - Vt4.1D},[Xn]		A64
<code>void vst1q_f64_x4(float64_t *ptr, float64x2x4_t val)</code>	val.val[3] -> Vt4.2D val.val[2] -> Vt3.2D val.val[1] -> Vt2.2D val.val[0] -> Vt.2D ptr -> Xn	ST1 {Vt.2D - Vt4.2D},[Xn]		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_mf8_x4(int8_t *ptr, int8x8x4_t val)</code>	<code>val.val[3] -> Vt4.8B val.val[2] -> Vt3.8B val.val[1] -> Vt2.8B val.val[0] -> Vt.8B ptr -> Xn</code>	ST1 {Vt.8B - Vt4.8B},[Xn]		A64
<code>void vst1q_mf8_x4(int8_t *ptr, int8x16x4_t val)</code>	<code>val.val[3] -> Vt4.16B val.val[2] -> Vt3.16B val.val[1] -> Vt2.16B val.val[0] -> Vt.16B ptr -> Xn</code>	ST1 {Vt.16B - Vt4.16B},[Xn]		A64

2.1.11.2 Store

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vstrq_p128(poly128_t *ptr, poly128_t val)</code>	<code>val -> Qt ptr -> Xn</code>	STR Qt,[Xn]		A32/A64

2.1.12 Table lookup

2.1.12.1 Table lookup

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vtbl1_s8(int8x8_t a, int8x8_t idx)</code>	Zeros(64):a -> Vn.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B},Vm.8B	Vd.8B -> result	v7/A32/A64
<code>uint8x8_t vtbl1_u8(uint8x8_t a, uint8x8_t idx)</code>	Zeros(64):a -> Vn.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B},Vm.8B	Vd.8B -> result	v7/A32/A64
<code>poly8x8_t vtbl1_p8(poly8x8_t a, uint8x8_t idx)</code>	Zeros(64):a -> Vn.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B},Vm.8B	Vd.8B -> result	v7/A32/A64
<code>mfloat8x8_t vtbl1_mf8(mfloat8x8_t a, uint8x8_t idx)</code>	Zeros(64):a -> Vn.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B},Vm.8B	Vd.8B -> result	A64
<code>int8x8_t vtbx1_s8(int8x8_t a, int8x8_t b, int8x8_t idx)</code>	Zeros(64):b -> Vn.16B a -> Vd.8B idx -> Vm.8B	MOVI Vtmp.8B,#8 CMHS Vtmp.8B,Vm.8B,Vtmp.8B TBL Vtmp1.8B,{Vn.16B},Vm.8B BIF Vd.8B,Vtmp1.8B,Vtmp.8B	Vd.8B -> result	v7/A32/A64
<code>uint8x8_t vtbx1_u8(uint8x8_t a, uint8x8_t b, uint8x8_t idx)</code>	Zeros(64):b -> Vn.16B a -> Vd.8B idx -> Vm.8B	MOVI Vtmp.8B,#8 CMHS Vtmp.8B,Vm.8B,Vtmp.8B TBL Vtmp1.8B,{Vn.16B},Vm.8B BIF Vd.8B,Vtmp1.8B,Vtmp.8B	Vd.8B -> result	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x8_t vtbx1_p8(poly8x8_t a, poly8x8_t b, uint8x8_t idx)</code>	Zeros(64):b -> Vn.16B a -> Vd.8B idx -> Vm.8B	MOVI Vtmp.8B,#8 CMHS Vtmp.8B,Vm.8B,Vtmp.8B TBL Vtmp1.8B,{Vn.16B},Vm.8B BIF Vd.8B,Vtmp1.8B, Vtmp.8B	Vd.8B -> result	v7/A32/A64
<code>mfloat8x8_t vtbx1_mf8(mfloat8x8_t a, mfloat8x8_t b, uint8x8_t idx)</code>	Zeros(64):b -> Vn.16B a -> Vd.8B idx -> Vm.8B	MOVI Vtmp.8B,#8 CMHS Vtmp.8B,Vm.8B,Vtmp.8B TBL Vtmp1.8B,{Vn.16B},Vm.8B BIF Vd.8B,Vtmp1.8B, Vtmp.8B	Vd.8B -> result	A64
<code>int8x8_t vtbl2_s8(int8x8x2_t a, int8x8_t idx)</code>	a.val[1]:a.val[0] -> Vn.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B},Vm.8B	Vd.8B -> result	v7/A32/A64
<code>uint8x8_t vtbl2_u8(uint8x8x2_t a, uint8x8_t idx)</code>	a.val[1]:a.val[0] -> Vn.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B},Vm.8B	Vd.8B -> result	v7/A32/A64
<code>poly8x8_t vtbl2_p8(poly8x8x2_t a, uint8x8_t idx)</code>	a.val[1]:a.val[0] -> Vn.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B},Vm.8B	Vd.8B -> result	v7/A32/A64
<code>mfloat8x8_t vtbl2_mf8(mfloat8x8x2_t a, uint8x8_t idx)</code>	a.val[1]:a.val[0] -> Vn.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B},Vm.8B	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vtbl3_s8(int8x8x3_t a, int8x8_t idx)</code>	<code>a.val[1]:a.val[0] -> Vn.16B Zeros(64):a.val[2] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x8_t vtbl3_u8(uint8x8x3_t a, uint8x8_t idx)</code>	<code>a.val[1]:a.val[0] -> Vn.16B Zeros(64):a.val[2] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x8_t vtbl3_p8(poly8x8x3_t a, uint8x8_t idx)</code>	<code>a.val[1]:a.val[0] -> Vn.16B Zeros(64):a.val[2] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>mfloat8x8_t vtbl3_mf8(mfloat8x8x3_t a, uint8x8_t idx)</code>	<code>a.val[1]:a.val[0] -> Vn.16B Zeros(64):a.val[2] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x8_t vtbl4_s8(int8x8x4_t a, int8x8_t idx)</code>	<code>a.val[1]:a.val[0] -> Vn.16B a.val[3]:a.val[2] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vtbl4_u8(uint8x8x4_t a, uint8x8_t idx)</code>	<code>a.val[1]:a.val[0] -> Vn.16B a.val[3]:a.val[2] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x8_t vtbl4_p8(poly8x8x4_t a, uint8x8_t idx)</code>	<code>a.val[1]:a.val[0] -> Vn.16B a.val[3]:a.val[2] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>mfloat8x8_t vtbl4_mf8(mfloat8x8x4_t a, uint8x8_t idx)</code>	<code>a.val[1]:a.val[0] -> Vn.16B a.val[3]:a.val[2] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x8_t vqtbl1_s8(int8x16_t t, uint8x8_t idx)</code>	<code>t -> Vn.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x16_t vqtbl1q_s8(int8x16_t t, uint8x16_t idx)</code>	<code>t -> Vn.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint8x8_t vqtbl1_u8(uint8x16_t t, uint8x8_t idx)</code>	<code>t -> Vn.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vqtbl1q_u8(uint8x16_t t, uint8x16_t idx)</code>	<code>t -> Vn.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>poly8x8_t vqtbl1_p8(poly8x16_t t, uint8x8_t idx)</code>	<code>t -> Vn.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>poly8x16_t vqtbl1q_p8(poly8x16_t t, uint8x16_t idx)</code>	<code>t -> Vn.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>mfloat8x8_t vqtbl1_mf8(mfloat8x16_t t, uint8x8_t idx)</code>	<code>t -> Vn.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vqtbl1q_mf8(mfloat8x16_t t, uint8x16_t idx)</code>	<code>t -> Vn.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int8x8_t vqtbl2_s8(int8x16x2_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B - Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vqtbl2q_s8(int8x16x2_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.16B</code>	TBL Vd.16B,{Vn.16B - Vn+1.16B},Vm.16B	Vd.16B -> result	A64
<code>uint8x8_t vqtbl2_u8(uint8x16x2_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.8B</code>	TBL Vd.8B,{Vn.16B - Vn+1.16B},Vm.8B	Vd.8B -> result	A64
<code>uint8x16_t vqtbl2q_u8(uint8x16x2_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.16B</code>	TBL Vd.16B,{Vn.16B - Vn+1.16B},Vm.16B	Vd.16B -> result	A64
<code>poly8x8_t vqtbl2_p8(poly8x16x2_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.8B</code>	TBL Vd.8B,{Vn.16B - Vn+1.16B},Vm.8B	Vd.8B -> result	A64
<code>poly8x16_t vqtbl2q_p8(poly8x16x2_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.16B</code>	TBL Vd.16B,{Vn.16B - Vn+1.16B},Vm.16B	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x8_t vqtbl2_mf8(</code> <code>mfloat8x16x2_t t,</code> <code>uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B</code> <code>t.val[1] -> Vn+1.16B</code> <code>idx -> Vm.8B</code>	TBL Vd.8B,{Vn.16B - Vn+1.16B},Vm.8B	Vd.8B -> result	A64
<code>mfloat8x16_t vqtbl2q_mf8(</code> <code>mfloat8x16x2_t t,</code> <code>uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B</code> <code>t.val[1] -> Vn+1.16B</code> <code>idx -> Vm.16B</code>	TBL Vd.16B,{Vn.16B - Vn+1.16B},Vm.16B	Vd.16B -> result	A64
<code>int8x8_t vqtbl3_s8(</code> <code>int8x16x3_t t,</code> <code>uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B</code> <code>t.val[1] -> Vn+1.16B</code> <code>t.val[2] -> Vn+2.16B</code> <code>idx -> Vm.8B</code>	TBL Vd.8B,{Vn.16B - Vn+2.16B},Vm.8B	Vd.8B -> result	A64
<code>int8x16_t vqtbl3q_s8(</code> <code>int8x16x3_t t,</code> <code>uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B</code> <code>t.val[1] -> Vn+1.16B</code> <code>t.val[2] -> Vn+2.16B</code> <code>idx -> Vm.16B</code>	TBL Vd.16B,{Vn.16B - Vn+2.16B},Vm.16B	Vd.16B -> result	A64
<code>uint8x8_t vqtbl3_u8(</code> <code>uint8x16x3_t t,</code> <code>uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B</code> <code>t.val[1] -> Vn+1.16B</code> <code>t.val[2] -> Vn+2.16B</code> <code>idx -> Vm.8B</code>	TBL Vd.8B,{Vn.16B - Vn+2.16B},Vm.8B	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vqtbl3q_u8(uint8x16x3_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B - Vn+2.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>poly8x8_t vqtbl3_p8(poly8x16x3_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B - Vn+2.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>poly8x16_t vqtbl3q_p8(poly8x16x3_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B - Vn+2.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>mfloat8x8_t vqtbl3_mf8(mfloat8x16x3_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B - Vn+2.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vqtbl3q_mf8(mfloat8x16x3_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B - Vn+2.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vqtbl4_s8(int8x16x4_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B - Vn+3.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x16_t vqtbl4q_s8(int8x16x4_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B - Vn+3.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint8x8_t vqtbl4_u8(uint8x16x4_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.8B</code>	<code>TBL Vd.8B,{Vn.16B - Vn+3.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vqtbl4q_u8(uint8x16x4_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.16B</code>	<code>TBL Vd.16B,{Vn.16B - Vn+3.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x8_t vqtbl4_p8(poly8x16x4_t t, uint8x8_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B - Vn+3.16B},Vm.8B	Vd.8B -> result	A64
poly8x16_t vqtbl4q_p8(poly8x16x4_t t, uint8x16_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.16B	TBL Vd.16B,{Vn.16B - Vn+3.16B},Vm.16B	Vd.16B -> result	A64
mfloating8x8_t vqtbl4_mf8(mfloating8x16x4_t t, uint8x8_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.8B	TBL Vd.8B,{Vn.16B - Vn+3.16B},Vm.8B	Vd.8B -> result	A64
mfloating8x16_t vqtbl4q_mf8(mfloating8x16x4_t t, uint8x16_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.16B	TBL Vd.16B,{Vn.16B - Vn+3.16B},Vm.16B	Vd.16B -> result	A64

2.1.12.2 Extended table lookup

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vtbx2_s8(int8x8_t a, int8x8x2_t b, int8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B a -> Vd.8B idx -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>uint8x8_t vtbx2_u8(uint8x8_t a, uint8x8x2_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B a -> Vd.8B idx -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>poly8x8_t vtbx2_p8(poly8x8_t a, poly8x8x2_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B a -> Vd.8B idx -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>mfloat8x8_t vtbx2_mf8(mfloat8x8_t a, mfloat8x8x2_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B a -> Vd.8B idx -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x8_t vtbx3_s8(int8x8_t a, int8x8x3_t b, int8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B Zeros(64):b.val[2] -> Vn+1.16B a -> Vd.8B idx -> Vm.8B</code>	<code>MOVI Vtmp.8B,#24 CMHS Vtmp.8B,Vm.8B,Vtmp.8B TBL Vtmp1.8B,{Vn.16B,Vn+1.16B},Vm.8 BIF Vd.8B,Vtmp1.8B,Vtmp.8B</code>	<code>Vd.8B -> result</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x8_t vtbx3_u8(uint8x8_t a, uint8x8x3_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B</code> <code>Zeros(64):b.val[2] -> Vn+1.16B</code> <code>a -> Vd.8B</code> <code>idx -> Vm.8B</code>	<code>MOVI Vtmp.8B,#24</code> <code>CMHS Vtmp.8B,Vm.8B,Vtmp.8B</code> <code>TBL Vtmp1.8B,{Vn.16B,Vn+1.16B},Vm.8B</code> <code>BIF Vd.8B,Vtmp1.8B,Vtmp.8B</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>poly8x8_t vtbx3_p8(poly8x8_t a, poly8x8x3_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B</code> <code>Zeros(64):b.val[2] -> Vn+1.16B</code> <code>a -> Vd.8B</code> <code>idx -> Vm.8B</code>	<code>MOVI Vtmp.8B,#24</code> <code>CMHS Vtmp.8B,Vm.8B,Vtmp.8B</code> <code>TBL Vtmp1.8B,{Vn.16B,Vn+1.16B},Vm.8B</code> <code>BIF Vd.8B,Vtmp1.8B,Vtmp.8B</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>mfloat8x8_t vtbx3_mf8(mfloat8x8_t a, mfloat8x8x3_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B</code> <code>Zeros(64):b.val[2] -> Vn+1.16B</code> <code>a -> Vd.8B</code> <code>idx -> Vm.8B</code>	<code>MOVI Vtmp.8B,#24</code> <code>CMHS Vtmp.8B,Vm.8B,Vtmp.8B</code> <code>TBL Vtmp1.8B,{Vn.16B,Vn+1.16B},Vm.8B</code> <code>BIF Vd.8B,Vtmp1.8B,Vtmp.8B</code>	<code>Vd.8B -> result</code>	<code>A64</code>
<code>int8x8_t vtbx4_s8(int8x8_t a, int8x8x4_t b, int8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B</code> <code>b.val[3]:b.val[2] -> Vn+1.16B</code> <code>a -> Vd.8B</code> <code>c-> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>uint8x8_t vtbx4_u8(uint8x8_t a, uint8x8x4_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B</code> <code>b.val[3]:b.val[2] -> Vn+1.16B</code> <code>a -> Vd.8B</code> <code>c-> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly8x8_t vtbx4_p8(poly8x8_t a, poly8x8x4_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B b.val[3]:b.val[2] -> Vn+1.16B a -> Vd.8B c -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	<code>v7/A32/A64</code>
<code>mfloat8x8_t vtbx4_mf8(mfloat8x8_t a, mfloat8x8x4_t b, uint8x8_t idx)</code>	<code>b.val[1]:b.val[0] -> Vn.16B b.val[3]:b.val[2] -> Vn+1.16B a -> Vd.8B c -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B,Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	<code>A64</code>
<code>int8x8_t vqtbx1_s8(int8x8_t a, int8x16_t t, uint8x8_t idx)</code>	<code>a -> Vd.8B t -> Vn.16B idx -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	<code>A64</code>
<code>int8x16_t vqtbx1q_s8(int8x16_t a, int8x16_t t, uint8x16_t idx)</code>	<code>a -> Vd.16B t -> Vn.16B idx -> Vm.16B</code>	<code>TBX Vd.16B,{Vn.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	<code>A64</code>
<code>uint8x8_t vqtbx1_u8(uint8x8_t a, uint8x16_t t, uint8x8_t idx)</code>	<code>a -> Vd.8B t -> Vn.16B idx -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	<code>A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vqtbx1q_u8(uint8x16_t a, uint8x16_t t, uint8x16_t idx)</code>	<code>a -> Vd.16B t -> Vn.16B idx -> Vm.16B</code>	<code>TBX Vd.16B,{Vn.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>poly8x8_t vqtbx1_p8(poly8x8_t a, poly8x16_t t, uint8x8_t idx)</code>	<code>a -> Vd.8B t -> Vn.16B idx -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>poly8x16_t vqtbx1q_p8(poly8x16_t a, poly8x16_t t, uint8x16_t idx)</code>	<code>a -> Vd.16B t -> Vn.16B idx -> Vm.16B</code>	<code>TBX Vd.16B,{Vn.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>mffloat8x8_t vqtbx1_mf8(mffloat8x8_t a, mffloat8x16_t t, uint8x8_t idx)</code>	<code>a -> Vd.8B t -> Vn.16B idx -> Vm.8B</code>	<code>TBX Vd.8B,{Vn.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>mffloat8x16_t vqtbx1q_mf8(mffloat8x16_t a, mffloat8x16_t t, uint8x16_t idx)</code>	<code>a -> Vd.16B t -> Vn.16B idx -> Vm.16B</code>	<code>TBX Vd.16B,{Vn.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x8_t vqtbx2_s8(int8x8_t a, int8x16x2_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.8B a -> Vd.8B</code>	<code>TBX Vd.8B,{Vn.16B - Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>int8x16_t vqtbx2q_s8(int8x16_t a, int8x16x2_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.16B a -> Vd.16B</code>	<code>TBX Vd.16B,{Vn.16B - Vn+1.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint8x8_t vqtbx2_u8(uint8x8_t a, uint8x16x2_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.8B a -> Vd.8B</code>	<code>TBX Vd.8B,{Vn.16B - Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vqtbx2q_u8(uint8x16_t a, uint8x16x2_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.16B a -> Vd.16B</code>	<code>TBX Vd.16B,{Vn.16B - Vn+1.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>poly8x8_t vqtbx2_p8(poly8x8_t a, poly8x16x2_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.8B a -> Vd.8B</code>	<code>TBX Vd.8B,{Vn.16B - Vn+1.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x16_t vqtbx2q_p8(poly8x16_t a, poly8x16x2_t t, uint8x16_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.16B a -> Vd.16B	TBX Vd.16B,{Vn.16B - Vn+1.16B},Vm.16B	Vd.16B -> result	A64
mfloat8x8_t vqtbx2_mf8(mfloat8x8_t a, mfloat8x16x2_t t, uint8x8_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.8B a -> Vd.8B	TBX Vd.8B,{Vn.16B - Vn+1.16B},Vm.8B	Vd.8B -> result	A64
mfloat8x16_t vqtbx2q_mf8(mfloat8x16_t a, mfloat8x16x2_t t, uint8x16_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B idx -> Vm.16B a -> Vd.16B	TBX Vd.16B,{Vn.16B - Vn+1.16B},Vm.16B	Vd.16B -> result	A64
int8x8_t vqtbx3_s8(int8x8_t a, int8x16x3_t t, uint8x8_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.8B a -> Vd.8B	TBX Vd.8B,{Vn.16B - Vn+2.16B},Vm.8B	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vqtbx3q_s8(int8x16_t a, int8x16x3_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.16B a -> Vd.16B</code>	<code>TBX Vd.16B,{Vn.16B - Vn+2.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint8x8_t vqtbx3_u8(uint8x8_t a, uint8x16x3_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.8B a -> Vd.8B</code>	<code>TBX Vd.8B,{Vn.16B - Vn+2.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vqtbx3q_u8(uint8x16_t a, uint8x16x3_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.16B a -> Vd.16B</code>	<code>TBX Vd.16B,{Vn.16B - Vn+2.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>poly8x8_t vqtbx3_p8(poly8x8_t a, poly8x16x3_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.8B a -> Vd.8B</code>	<code>TBX Vd.8B,{Vn.16B - Vn+2.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x16_t vqtbx3q_p8(poly8x16_t a, poly8x16x3_t t, uint8x16_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.16B a -> Vd.16B	TBX Vd.16B,{Vn.16B - Vn+2.16B},Vm.16B	Vd.16B -> result	A64
mfloating8x8_t vqtbx3_mf8(mfloating8x8_t a, mfloating8x16x3_t t, uint8x8_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.8B a -> Vd.8B	TBX Vd.8B,{Vn.16B - Vn+2.16B},Vm.8B	Vd.8B -> result	A64
mfloating8x16_t vqtbx3q_mf8(mfloating8x16_t a, mfloating8x16x3_t t, uint8x16_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B idx -> Vm.16B a -> Vd.16B	TBX Vd.16B,{Vn.16B - Vn+2.16B},Vm.16B	Vd.16B -> result	A64
int8x8_t vqtbx4_s8(int8x8_t a, int8x16x4_t t, uint8x8_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.8B a -> Vd.8B	TBX Vd.8B,{Vn.16B - Vn+3.16B},Vm.8B	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int8x16_t vqtbx4q_s8(int8x16_t a, int8x16x4_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.16B a -> Vd.16B</code>	<code>TBX Vd.16B,{Vn.16B - Vn+3.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint8x8_t vqtbx4_u8(uint8x8_t a, uint8x16x4_t t, uint8x8_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.8B a -> Vd.8B</code>	<code>TBX Vd.8B,{Vn.16B - Vn+3.16B},Vm.8B</code>	<code>Vd.8B -> result</code>	A64
<code>uint8x16_t vqtbx4q_u8(uint8x16_t a, uint8x16x4_t t, uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.16B a -> Vd.16B</code>	<code>TBX Vd.16B,{Vn.16B - Vn+3.16B},Vm.16B</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x8_t vqtbx4_p8(poly8x8_t a, poly8x16x4_t t, uint8x8_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.8B a -> Vd.8B	TBX Vd.8B,{Vn.16B - Vn+3.16B},Vm.8B	Vd.8B -> result	A64
poly8x16_t vqtbx4q_p8(poly8x16_t a, poly8x16x4_t t, uint8x16_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.16B a -> Vd.16B	TBX Vd.16B,{Vn.16B - Vn+3.16B},Vm.16B	Vd.16B -> result	A64
mfloat8x8_t vqtbx4_mf8(mfloat8x8_t a, mfloat8x16x4_t t, uint8x8_t idx)	t.val[0] -> Vn.16B t.val[1] -> Vn+1.16B t.val[2] -> Vn+2.16B t.val[3] -> Vn+3.16B idx -> Vm.8B a -> Vd.8B	TBX Vd.8B,{Vn.16B - Vn+3.16B},Vm.8B	Vd.8B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x16_t vqtbx4q_mf8(</code> <code>mfloat8x16_t a,</code> <code>mfloat8x16x4_t t,</code> <code>uint8x16_t idx)</code>	<code>t.val[0] -> Vn.16B</code> <code>t.val[1] -> Vn+1.16B</code> <code>t.val[2] -> Vn+2.16B</code> <code>t.val[3] -> Vn+3.16B</code> <code>idx -> Vm.16B</code> <code>a -> Vd.16B</code>	<code>TBX Vd.16B, {Vn.16B - Vn+3.16B}, Vm.16B</code>	<code>Vd.16B -> result</code>	A64

2.1.12.3 Lookup table read with 2-bit indices

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vluti2_lane_u8(</code> <code>uint8x8_t vn,</code> <code>uint8x8_t vm,</code> <code>const int index)</code>	<code>vn -> Vn.16B</code> <code>vm -> Vm</code> <code>0 <= index <= 1</code>	<code>LUTI2 Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>uint8x16_t vluti2_laneq_u8(</code> <code>uint8x8_t vn,</code> <code>uint8x16_t vm,</code> <code>const int index)</code>	<code>vn -> Vn.16B</code> <code>vm -> Vm</code> <code>0 <= index <= 3</code>	<code>LUTI2 Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>uint8x16_t vluti2q_lane_u8(</code> <code>uint8x16_t vn,</code> <code>uint8x8_t vm,</code> <code>const int index)</code>	<code>vn -> Vn.16B</code> <code>vm -> Vm</code> <code>0 <= index <= 1</code>	<code>LUTI2 Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vluti2q_laneq_u8(uint8x16_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 3</code>	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64
<code>int8x16_t vluti2_lane_s8(int8x8_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 1</code>	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64
<code>int8x16_t vluti2_laneq_s8(int8x8_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 3</code>	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64
<code>int8x16_t vluti2q_lane_s8(int8x16_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 1</code>	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64
<code>int8x16_t vluti2q_laneq_s8(int8x16_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 3</code>	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x16_t vluti2_lane_p8(poly8x8_t vn, uint8x8_t vm, const int index)	vn -> Vn.16B vm -> Vm 0 <= index <= 1	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64
poly8x16_t vluti2_laneq_p8(poly8x8_t vn, uint8x16_t vm, const int index)	vn -> Vn.16B vm -> Vm 0 <= index <= 3	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64
poly8x16_t vluti2q_lane_p8(poly8x16_t vn, uint8x8_t vm, const int index)	vn -> Vn.16B vm -> Vm 0 <= index <= 1	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64
poly8x16_t vluti2q_laneq_p8(poly8x16_t vn, uint8x16_t vm, const int index)	vn -> Vn.16B vm -> Vm 0 <= index <= 3	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64
mfloat8x16_t vluti2_lane_mf8(mfloat8x8_t vn, uint8x8_t vm, const int index)	vn -> Vn.16B vm -> Vm 0 <= index <= 1	LUTI2 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x16_t vluti2_laneq_mf8(mfloat8x8_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>mfloat8x16_t vluti2q_lane_mf8(mfloat8x16_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 1</code>	<code>LUTI2 Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>mfloat8x16_t vluti2q_laneq_mf8(mfloat8x16_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vluti2_lane_u16(uint16x4_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>uint16x8_t vluti2_laneq_u16(uint16x4_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 7</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vluti2q_lane_u16(uint16x8_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>uint16x8_t vluti2q_laneq_u16(uint16x8_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 7</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>int16x8_t vluti2_lane_s16(int16x4_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>int16x8_t vluti2_laneq_s16(int16x4_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 7</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>int16x8_t vluti2q_lane_s16(int16x8_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vluti2q_laneq_s16(int16x8_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 7</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vluti2_lane_f16(float16x4_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vluti2_laneq_f16(float16x4_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 7</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vluti2q_lane_f16(float16x8_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vluti2q_laneq_f16(float16x8_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 7</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8_t vluti2_lane_bf16(bfloat16x4_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>bfloat16x8_t vluti2_laneq_bf16(bfloat16x4_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 7</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>bfloat16x8_t vluti2q_lane_bf16(bfloat16x8_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>bfloat16x8_t vluti2q_laneq_bf16(bfloat16x8_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 7</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>poly16x8_t vluti2_lane_p16(poly16x4_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI2 Vd.8H, {Vn.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly16x8_t vluti2_laneq_p16(poly16x4_t vn, uint8x16_t vm, const int index)	vn -> Vn.8H vm -> Vm 0 <= index <= 7	LUTI2 Vd.8H, {Vn.8H}, Vm[index]	Vd.8H -> result	A64
poly16x8_t vluti2q_lane_p16(poly16x8_t vn, uint8x8_t vm, const int index)	vn -> Vn.8H vm -> Vm 0 <= index <= 3	LUTI2 Vd.8H, {Vn.8H}, Vm[index]	Vd.8H -> result	A64
poly16x8_t vluti2q_laneq_p16(poly16x8_t vn, uint8x16_t vm, const int index)	vn -> Vn.8H vm -> Vm 0 <= index <= 7	LUTI2 Vd.8H, {Vn.8H}, Vm[index]	Vd.8H -> result	A64

2.1.12.4 Lookup table read with 4-bit indices

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint8x16_t vluti4q_lane_u8(uint8x16_t vn, uint8x8_t vm, const int index)	vn -> Vn.16B vm -> Vm 0 <= index <= 0	LUTI4 Vd.16B, {Vn.16B}, Vm[index]	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vluti4q_laneq_u8(uint8x16_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 1</code>	<code>LUTIA Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>int8x16_t vluti4q_lane_s8(int8x16_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 0</code>	<code>LUTIA Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>int8x16_t vluti4q_laneq_s8(int8x16_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 1</code>	<code>LUTIA Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>poly8x16_t vluti4q_lane_p8(poly8x16_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 0</code>	<code>LUTIA Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>poly8x16_t vluti4q_laneq_p8(poly8x16_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 1</code>	<code>LUTIA Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x16_t vluti4q_lane_mf8(mfloat8x16_t vn, uint8x8_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 0</code>	<code>LUTIA Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>mfloat8x16_t vluti4q_laneq_mf8(mfloat8x16_t vn, uint8x16_t vm, const int index)</code>	<code>vn -> Vn.16B vm -> Vm 0 <= index <= 1</code>	<code>LUTIA Vd.16B, {Vn.16B}, Vm[index]</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vluti4q_lane_u16_x2(uint16x8x2_t vn, uint8x8_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 1</code>	<code>LUTIA Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>uint16x8_t vluti4q_laneq_u16_x2(uint16x8x2_t vn, uint8x16_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTIA Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64
<code>int16x8_t vluti4q_lane_s16_x2(int16x8x2_t vn, uint8x8_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 1</code>	<code>LUTIA Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vluti4q_laneq_s16_x2(int16x8x2_t vn, uint8x16_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTIA Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]Vd.8H -> result</code>		A64
<code>float16x8_t vluti4q_lane_f16_x2(float16x8x2_t vn, uint8x8_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 1</code>	<code>LUTIA Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]Vd.8H -> result</code>		A64
<code>float16x8_t vluti4q_laneq_f16_x2(float16x8x2_t vn, uint8x16_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTIA Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]Vd.8H -> result</code>		A64
<code>bfloat16x8_t vluti4q_lane_bf16_x2(bfloat16x8x2_t vn, uint8x8_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 1</code>	<code>LUTIA Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]Vd.8H -> result</code>		A64
<code>bfloat16x8_t vluti4q_laneq_bf16_x2(bfloat16x8x2_t vn, uint8x16_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTIA Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]Vd.8H -> result</code>		A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly16x8_t vluti4q_lane_p16_x2(poly16x8x2_t vn, uint8x8_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 1</code>	<code>LUTI4 Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]#d.8H -> result</code>		A64
<code>poly16x8_t vluti4q_laneq_p16_x2(poly16x8x2_t vn, uint8x16_t vm, const int index)</code>	<code>vn.val[0] -> Vn1.8H vn.val[1] -> Vn2.8H vm -> Vm 0 <= index <= 3</code>	<code>LUTI4 Vd.8H, {Vn1.8H, Vn2.8H}, Vm[index]#d.8H -> result</code>		A64

2.2 Crypto

2.2.1 Cryptography

2.2.1.1 AES

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vaeseq_u8(uint8x16_t data, uint8x16_t key)</code>	<code>data -> Vd.16B key -> Vn.16B</code>	<code>AESE Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	A32/A64
<code>uint8x16_t vaesdq_u8(uint8x16_t data, uint8x16_t key)</code>	<code>data -> Vd.16B key -> Vn.16B</code>	<code>AESD Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vaesmcq_u8(uint8x16_t data)</code>	<code>data -> Vn.16B</code>	<code>AESMC Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	A32/A64
<code>uint8x16_t vaesimcq_u8(uint8x16_t data)</code>	<code>data -> Vn.16B</code>	<code>AESIMC Vd.16B,Vn.16B</code>	<code>Vd.16B -> result</code>	A32/A64

2.2.1.2 SHA1

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsha1cq_u32(uint32x4_t hash_abcd, uint32_t hash_e, uint32x4_t wk)</code>	<code>hash_abcd -> Qd hash_e -> Sn wk -> Vm.4S</code>	<code>SHA1C Qd,Sn,Vm.4S</code>	<code>Qd -> result</code>	A32/A64
<code>uint32x4_t vsha1pq_u32(uint32x4_t hash_abcd, uint32_t hash_e, uint32x4_t wk)</code>	<code>hash_abcd -> Qd hash_e -> Sn wk -> Vm.4S</code>	<code>SHA1P Qd,Sn,Vm.4S</code>	<code>Qd -> result</code>	A32/A64
<code>uint32x4_t vsha1mq_u32(uint32x4_t hash_abcd, uint32_t hash_e, uint32x4_t wk)</code>	<code>hash_abcd -> Qd hash_e -> Sn wk -> Vm.4S</code>	<code>SHA1M Qd,Sn,Vm.4S</code>	<code>Qd -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32_t vsha1h_u32(uint32_t hash_e)</code>	<code>hash_e -> Sn</code>	SHA1H <code>Sd, Sn</code>	<code>Sd -> result</code>	A32/A64
<code>uint32x4_t vsha1su0q_u32(uint32x4_t w0_3, uint32x4_t w4_7, uint32x4_t w8_11)</code>	<code>w0_3 -> Vd.4S w4_7 -> Vn.4S w8_11 -> Vm.4S</code>	SHA1SU0 <code>Vd.4S, Vn.4S, Vm.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>uint32x4_t vsha1su1q_u32(uint32x4_t tw0_3, uint32x4_t w12_15)</code>	<code>tw0_3 -> Vd.4S w12_15 -> Vn.4S</code>	SHA1SU1 <code>Vd.4S, Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64

2.2.1.3 SHA256

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsha256hq_u32(uint32x4_t hash_abcd, uint32x4_t hash_efgh, uint32x4_t wk)</code>	<code>hash_abcd -> Qd hash_efgh -> Qn wk -> Vm.4S</code>	SHA256H <code>Qd, Qn, Vm.4S</code>	<code>Qd -> result</code>	A32/A64
<code>uint32x4_t vsha256h2q_u32(uint32x4_t hash_efgh, uint32x4_t hash_abcd, uint32x4_t wk)</code>	<code>hash_efgh -> Qd hash_abcd -> Qn wk -> Vm.4S</code>	SHA256H2 <code>Qd, Qn, Vm.4S</code>	<code>Qd -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsha256su0q_u32(uint32x4_t w0_3, uint32x4_t w4_7)</code>	<code>w0_3 -> Vd.4S w4_7 -> Vn.4S</code>	<code>SHA256SU0 Vd.4S,Vn.4S</code>	<code>Vd.4S -> result</code>	A32/A64
<code>uint32x4_t vsha256su1q_u32(uint32x4_t tw0_3, uint32x4_t w8_11, uint32x4_t w12_15)</code>	<code>tw0_3 -> Vd.4S w8_11 -> Vn.4S w12_15 -> Vm.4S</code>	<code>SHA256SU1 Vd.4S,Vn.4S,Vm.4S</code>	<code>Vd.4S -> result</code>	A32/A64

2.2.2 Vector arithmetic

2.2.2.1 Polynomial

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>poly128_t vmull_p64(poly64_t a, poly64_t b)</code>	<code>a -> Vn.1D b -> Vm.1D</code>	<code>PMULL Vd.1Q,Vn.1D,Vm.1D</code>	<code>Vd.1Q -> result</code>	A32/A64
<code>poly128_t vmull_high_p64(poly64x2_t a, poly64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>PMULL2 Vd.1Q,Vn.2D,Vm.2D</code>	<code>Vd.1Q -> result</code>	A32/A64

2.2.2.1.1 Polynomial multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly8x8_t vadd_p8(poly8x8_t a, poly8x8_t b)	a -> Vn.8B b -> Vm.8B	EOR Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
poly16x4_t vadd_p16(poly16x4_t a, poly16x4_t b)	a -> Vn.8B b -> Vm.8B	EOR Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
poly64x1_t vadd_p64(poly64x1_t a, poly64x1_t b)	a -> Vn.8B b -> Vm.8B	EOR Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
poly8x16_t vaddq_p8(poly8x16_t a, poly8x16_t b)	a -> Vn.16B b -> Vm.16B	EOR Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
poly16x8_t vaddq_p16(poly16x8_t a, poly16x8_t b)	a -> Vn.16B b -> Vm.16B	EOR Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
poly64x2_t vaddq_p64(poly64x2_t a, poly64x2_t b)	a -> Vn.16B b -> Vm.16B	EOR Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64
poly128_t vaddq_p128(poly128_t a, poly128_t b)	a -> Vn.16B b -> Vm.16B	EOR Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64

2.2.2.1.2 Polynomial addition

2.3 CRC32

2.3.1 Cryptography

2.3.1.1 CRC32

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32_t __crc32b(uint32_t a, uint8_t b)</code>	<code>a -> Wn b -> Wm</code>	CRC32B Wd,Wn,Wm	Wd -> result	A32/A64
<code>uint32_t __crc32h(uint32_t a, uint16_t b)</code>	<code>a -> Wn b -> Wm</code>	CRC32H Wd,Wn,Wm	Wd -> result	A32/A64
<code>uint32_t __crc32w(uint32_t a, uint32_t b)</code>	<code>a -> Wn b -> Wm</code>	CRC32W Wd,Wn,Wm	Wd -> result	A32/A64
<code>uint32_t __crc32d(uint32_t a, uint64_t b)</code>	<code>a -> Wn b -> Xm</code>	CRC32X Wd,Wn,Xm	Wd -> result	A32/A64
<code>uint32_t __crc32cb(uint32_t a, uint8_t b)</code>	<code>a -> Wn b -> Wm</code>	CRC32CB Wd,Wn,Wm	Wd -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32_t __crc32ch(uint32_t a, uint16_t b)</code>	<code>a -> Wn b -> Wm</code>	<code>CRC32CH Wd,Wn,Wm</code>	<code>Wd -> result</code>	A32/A64
<code>uint32_t __crc32cw(uint32_t a, uint32_t b)</code>	<code>a -> Wn b -> Wm</code>	<code>CRC32CW Wd,Wn,Wm</code>	<code>Wd -> result</code>	A32/A64
<code>uint32_t __crc32cd(uint32_t a, uint64_t b)</code>	<code>a -> Wn b -> Xm</code>	<code>CRC32CX Wd,Wn,Xm</code>	<code>Wd -> result</code>	A32/A64

2.4 sqrdmlah intrinsics (From ARMv8.1-A)

2.4.1 Vector arithmetic

2.4.1.1 Multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vqrdmlah_s16(int16x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.4H</code>	<code>SQRDMLAH Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vqrdmlah_s32(int32x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.2S</code>	<code>SQRDMLAH Vd.2S, Vn.2S, Vm.2S</code>	<code>Vd.2S -> result</code>	A64
<code>int16x8_t vqrdmlahq_s16(int16x8_t a, int16x8_t b, int16x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.8H</code>	<code>SQRDMLAH Vd.8H, Vn.8H, Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>int32x4_t vqrdmlahq_s32(int32x4_t a, int32x4_t b, int32x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S</code>	<code>SQRDMLAH Vd.4S, Vn.4S, Vm.4S</code>	<code>Vd.4S -> result</code>	A64
<code>int16x4_t vqrdmlsh_s16(int16x4_t a, int16x4_t b, int16x4_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.4H</code>	<code>SQRDMLSH Vd.4H, Vn.4H, Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>int32x2_t vqrdmlsh_s32(int32x2_t a, int32x2_t b, int32x2_t c)</code>	<code>a -> Vd.2S b -> Vn.2S c -> Vm.2S</code>	<code>SQRDMLSH Vd.2S, Vn.2S, Vm.2S</code>	<code>Vd.2S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vqrdmlshq_s16(int16x8_t a, int16x8_t b, int16x8_t c)</code>	a -> Vd.8H b -> Vn.8H c -> Vm.8H	<code>SQRDMLSH Vd.8H,Vn.8H,Vm.8H</code>	Vd.8H -> result	A64
<code>int32x4_t vqrdmlshq_s32(int32x4_t a, int32x4_t b, int32x4_t c)</code>	a -> Vd.4S b -> Vn.4S c -> Vm.4S	<code>SQRDMLSH Vd.4S,Vn.4S,Vm.4S</code>	Vd.4S -> result	A64
<code>int16_t vqrdmlahh_s16(int16_t a, int16_t b, int16_t c)</code>	a -> Hd b -> Hn c -> Hm	<code>SQRDMLSH Hd,Hn,Hm</code>	Hd -> result	A64
<code>int32_t vqrdmlahs_s32(int32_t a, int32_t b, int32_t c)</code>	a -> Sd b -> Sn c -> Sm	<code>SQRDMLSH Sd,Sn,Sm</code>	Sd -> result	A64
<code>int16_t vqrdmlshh_s16(int16_t a, int16_t b, int16_t c)</code>	a -> Hd b -> Hn c -> Hm	<code>SQRDMLSH Hd,Hn,Hm</code>	Hd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t vqrdmlshs_s32(int32_t a, int32_t b, int32_t c)</code>	a -> Sd b -> Sn c -> Sm	SQRDMLSH Sd,Sn,Sm	Sd -> result	A64

2.4.1.1.1 Saturating multiply-accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vqrdmlah_lane_s16(int16x4_t a, int16x4_t b, int16x4_t v, const int lane)</code>	a -> Vd.4H b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	SQRDMLAH Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
<code>int16x8_t vqrdmlahq_lane_s16(int16x8_t a, int16x8_t b, int16x4_t v, const int lane)</code>	a -> Vd.8H b -> Vn.8H v -> Vm.4H 0 <= lane <= 3	SQRDMLAH Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64
<code>int16x4_t vqrdmlah_laneq_s16(int16x4_t a, int16x4_t b, int16x8_t v, const int lane)</code>	a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	SQRDMLAH Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vqrdmlahq_laneq_s16(int16x8_t a, int16x8_t b, int16x8_t v, const int lane)</code>	<code>a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQRDMLAH Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>int32x2_t vqrdmlah_lane_s32(int32x2_t a, int32x2_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQRDMLAH Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vqrdmlahq_lane_s32(int32x4_t a, int32x4_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQRDMLAH Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int32x2_t vqrdmlah_laneq_s32(int32x2_t a, int32x2_t b, int32x4_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQRDMLAH Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vqrdmlahq_laneq_s32(int32x4_t a, int32x4_t b, int32x4_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQRDMLAH Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x4_t vqrdmlsh_lane_s16(int16x4_t a, int16x4_t b, int16x4_t v, const int lane)</code>	<code>a -> Vd.4H b -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQRDMLSH Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vqrdmlshq_lane_s16(int16x8_t a, int16x8_t b, int16x4_t v, const int lane)</code>	<code>a -> Vd.8H b -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>SQRDMLSH Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>int16x4_t vqrdmlsh_laneq_s16(int16x4_t a, int16x4_t b, int16x8_t v, const int lane)</code>	<code>a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQRDMLSH Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>int16x8_t vqrdmlshq_laneq_s16(int16x8_t a, int16x8_t b, int16x8_t v, const int lane)</code>	<code>a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>SQRDMLSH Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>int32x2_t vqrdmlsh_lane_s32(int32x2_t a, int32x2_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQRDMLSH Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vqrdmlshq_lane_s32(int32x4_t a, int32x4_t b, int32x2_t v, const int lane)</code>	<code>a -> Vd.4S b -> Vn.4S v -> Vm.2S 0 <= lane <= 1</code>	<code>SQRDMLSH Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int32x2_t vqrdmlsh_laneq_s32(int32x2_t a, int32x2_t b, int32x4_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQRDMLSH Vd.2S,Vn.2S,Vm.S[lane]</code>	<code>Vd.2S -> result</code>	A64
<code>int32x4_t vqrdmlshq_laneq_s32(int32x4_t a, int32x4_t b, int32x4_t v, const int lane)</code>	<code>a -> Vd.2S b -> Vn.2S v -> Vm.4S 0 <= lane <= 3</code>	<code>SQRDMLSH Vd.4S,Vn.4S,Vm.S[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>int16_t vqrdmlahh_lane_s16(int16_t a, int16_t b, int16x4_t v, const int lane)</code>	<code>a -> Hd b -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>SQRDMLAH Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64
<code>int16_t vqrdmlahh_laneq_s16(int16_t a, int16_t b, int16x8_t v, const int lane)</code>	<code>a -> Hd b -> Hn v -> Vm.8H 0 <= lane <= 7</code>	<code>SQRDMLAH Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t vqrdmlahs_lane_s32(int32_t a, int32_t b, int32x2_t v, const int lane)</code>	a -> Sd b -> Sn v -> Vm.2S 0 <= lane <= 1	SQRDMLAH Sd,Sn,Vm.S[lane]	Sd -> result	A64
<code>int32_t vqrdmlahs_laneq_s32(int32_t a, int32_t b, int32x4_t v, const int lane)</code>	a -> Sd b -> Sn v -> Vm.4S 0 <= lane <= 3	SQRDMLAH Sd,Sn,Vm.S[lane]	Sd -> result	A64
<code>int16_t vqrdmlshh_lane_s16(int16_t a, int16_t b, int16x4_t v, const int lane)</code>	a -> Hd b -> Hn v -> Vm.4H 0 <= lane <= 3	SQRDMLSH Hd,Hn,Vm.H[lane]	Hd -> result	A64
<code>int16_t vqrdmlshh_laneq_s16(int16_t a, int16_t b, int16x8_t v, const int lane)</code>	a -> Hd b -> Hn v -> Vm.8H 0 <= lane <= 7	SQRDMLSH Hd,Hn,Vm.H[lane]	Hd -> result	A64
<code>int32_t vqrdmlshs_lane_s32(int32_t a, int32_t b, int32x2_t v, const int lane)</code>	a -> Sd b -> Sn v -> Vm.2S 0 <= lane <= 1	SQRDMLSH Sd,Sn,Vm.S[lane]	Sd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t vqrdmlshs_laneq_s32(int32_t a, int32_t b, int32x4_t v, const int lane)</code>	<code>a -> Sd b -> Sn v -> Vm.4S 0 <= lane <= 3</code>	<code>SQRDMLSH Sd,Sn,Vm.S[lane]</code>	<code>Sd -> result</code>	A64

2.4.1.1.2 Saturating multiply-accumulate by element

2.5 fp16 scalar intrinsics (available through <arm_fp16.h> from ARMv8.2-A)

2.5.1 Vector arithmetic

2.5.1.1 Absolute

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vabsh_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FABS Hd,Hn</code>	<code>Hd -> result</code>	A32/A64

2.5.1.1.1 Absolute value

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vabdh_f16(float16_t a, float16_t b)	a -> Hn b -> Hm	FABD (scalar) Hd,Hn,Hm	Hd -> result	A64

2.5.1.1.2 Absolute difference

2.5.1.2 Reciprocal

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vrecpeh_f16(float16_t a)	a -> Hn	FRECPE Hd,Hn	Hd -> result	A64
float16_t vrecpxh_f16(float16_t a)	a -> Hn	FRECPX Hd,Hn	Hd -> result	A64

2.5.1.2.1 Reciprocal estimate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vrsqrteh_f16(float16_t a)	a -> Hn	FRSQRTE Hd,Hn	Hd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vrsqrtsh_f16(float16_t a, float16_t b)	a -> Hn b -> Hm	FRSQRTS Hd,Hn,Hm	Hd -> result	A64

2.5.1.2.2 Reciprocal square-root estimate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vrecpsh_f16(float16_t a, float16_t b)	a -> Hn b -> Hm	FRECPS Hd,Hn,Hm	Hd -> result	A64

2.5.1.2.3 Reciprocal step

2.5.1.3 Rounding

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vrndh_f16(float16_t a)	a -> Hn	FRINTZ Hd,Hn	Hd -> result	A32/A64
float16_t vrndah_f16(float16_t a)	a -> Hn	FRINTA Hd,Hn	Hd -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vrndih_f16(float16_t a)	a -> Hn	FRINTI Hd,Hn	Hd -> result	A32/A64
float16_t vrndmh_f16(float16_t a)	a -> Hn	FRINTM Hd,Hn	Hd -> result	A32/A64
float16_t vrndnh_f16(float16_t a)	a -> Hn	FRINTN Hd,Hn	Hd -> result	A32/A64
float16_t vrndph_f16(float16_t a)	a -> Hn	FRINTP Hd,Hn	Hd -> result	A32/A64
float16_t vrndxh_f16(float16_t a)	a -> Hn	FRINTX Hd,Hn	Hd -> result	A32/A64

2.5.1.4 Square root

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vsqrth_f16(float16_t a)	a -> Hn	FSQRT Hd,Hn	Hd -> result	A32/A64

2.5.1.5 Add

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vaddh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FADD Hd,Hn,Hm</code>	<code>Hd -> result</code>	A32/A64

2.5.1.5.1 Addition**2.5.1.6 Division**

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vdivh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FDIV Hd,Hn,Hm</code>	<code>Hd -> result</code>	A32/A64

2.5.1.7 Maximum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vmaxh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FMAX Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vmaxnmh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FMAXNM Hd,Hn,Hm</code>	<code>Hd -> result</code>	A32/A64

2.5.1.8 Minimum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vminh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FMIN Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64
<code>float16_t vminnmh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FMINNM Hd,Hn,Hm</code>	<code>Hd -> result</code>	A32/A64

2.5.1.9 Multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vmulh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FMUL Hd,Hn,Hm</code>	<code>Hd -> result</code>	A32/A64

2.5.1.9.1 Multiplication

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vmulxh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FMULX Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

2.5.1.9.2 Multiply extended

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vfmah_f16(float16_t a, float16_t b, float16_t c)</code>	<code>b -> Hn c -> Hm a -> Ha</code>	<code>FMADD Hd,Hn,Hm,Ha</code>	<code>Hd -> result</code>	A32/A64
<code>float16_t vfmsb_f16(float16_t a, float16_t b, float16_t c)</code>	<code>b -> Hn c -> Hm a -> Ha</code>	<code>FMSUB Hd,Hn,Hm,Ha</code>	<code>Hd -> result</code>	A32/A64

2.5.1.9.3 Fused multiply-accumulate

2.5.1.10 Subtract

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vsubh_f16(float16_t a, float16_t b)	a -> Hn b -> Hm	FSUB Hd,Hn,Hm	Hd -> result	A32/A64

2.5.1.10.1 Subtraction

2.5.2 Compare

2.5.2.1 Bitwise equal to zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16_t vceqzh_f16(float16_t a)	a -> Hn	FCMEQ Hd,Hn,#0	Hd -> result	A64

2.5.2.2 Greater than or equal to zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16_t vcgezh_f16(float16_t a)	a -> Hn	FCMGE Hd,Hn,#0	Hd -> result	A64

2.5.2.3 Greater than zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16_t vcgtzh_f16(float16_t a)	a -> Hn	FCMGT Hd,Hn,#0	Hd -> result	A64

2.5.2.4 Less than or equal to zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16_t vclezh_f16(float16_t a)	a -> Hn	FCMLE Hd,Hn,#0	Hd -> result	A64

2.5.2.5 Less than zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16_t vcltzh_f16(float16_t a)	a -> Hn	FCMLT Hd,Hn,#0	Hd -> result	A64

2.5.2.6 Absolute greater than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vcageh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FACGE Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

2.5.2.7 Absolute greater than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vcagth_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FACGT Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

2.5.2.8 Absolute less than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vcaleh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FACGE Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

2.5.2.9 Absolute less than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vcalth_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FACGT Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

2.5.2.10 Equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vceqh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FCMEQ Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

2.5.2.11 Greater than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vcgeh_f16(float16_t a, float16_t b)</code>	<code>a -> Hn b -> Hm</code>	<code>FCMGE Hd,Hn,Hm</code>	<code>Hd -> result</code>	A64

2.5.2.12 Greater than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t</code> <code>vcgth_f16</code> (<code>float16_t</code> <code>a</code> , <code>float16_t</code> <code>b</code>)	<code>a</code> -> <code>Hn</code> <code>b</code> -> <code>Hm</code>	<code>FCMGT</code> <code>Hd</code> , <code>Hn</code> , <code>Hm</code>	<code>Hd</code> -> <code>result</code>	A64

2.5.2.13 Less than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t</code> <code>vcleh_f16</code> (<code>float16_t</code> <code>a</code> , <code>float16_t</code> <code>b</code>)	<code>a</code> -> <code>Hn</code> <code>b</code> -> <code>Hm</code>	<code>FCMGE</code> <code>Hd</code> , <code>Hn</code> , <code>Hm</code>	<code>Hd</code> -> <code>result</code>	A64

2.5.2.14 Less than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t</code> <code>vclth_f16</code> (<code>float16_t</code> <code>a</code> , <code>float16_t</code> <code>b</code>)	<code>a</code> -> <code>Hn</code> <code>b</code> -> <code>Hm</code>	<code>FCMGT</code> <code>Hd</code> , <code>Hn</code> , <code>Hm</code>	<code>Hd</code> -> <code>result</code>	A64

2.5.3 Data type conversion

2.5.3.1 Conversions

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vcvth_f16_s16(int16_t a)</code>	<code>a -> Hn</code>	<code>SCVTF Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>float16_t vcvth_f16_s32(int32_t a)</code>	<code>a -> Hn</code>	<code>SCVTF Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>float16_t vcvth_f16_s64(int64_t a)</code>	<code>a -> Hn</code>	<code>SCVTF Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>float16_t vcvth_f16_u16(uint16_t a)</code>	<code>a -> Hn</code>	<code>UCVTF Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>float16_t vcvth_f16_u32(uint32_t a)</code>	<code>a -> Hn</code>	<code>UCVTF Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>float16_t vcvth_f16_u64(uint64_t a)</code>	<code>a -> Hn</code>	<code>UCVTF Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int16_t vcvth_s16_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTZS Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int32_t vcvth_s32_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTZS Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>int64_t vcvth_s64_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTZS Hd,Hn</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16_t vcvth_u16_f16(float16_t a)	a -> Hn	FCVTZU Hd,Hn	Hd -> result	A64
uint32_t vcvth_u32_f16(float16_t a)	a -> Hn	FCVTZU Hd,Hn	Hd -> result	A32/A64
uint64_t vcvth_u64_f16(float16_t a)	a -> Hn	FCVTZU Hd,Hn	Hd -> result	A64
int16_t vcvtah_s16_f16(float16_t a)	a -> Hn	FCVTAS Hd,Hn	Hd -> result	A64
int32_t vcvtah_s32_f16(float16_t a)	a -> Hn	FCVTAS Hd,Hn	Hd -> result	A32/A64
int64_t vcvtah_s64_f16(float16_t a)	a -> Hn	FCVTAS Hd,Hn	Hd -> result	A64
uint16_t vcvtah_u16_f16(float16_t a)	a -> Hn	FCVTAU Hd,Hn	Hd -> result	A64
uint32_t vcvtah_u32_f16(float16_t a)	a -> Hn	FCVTAU Hd,Hn	Hd -> result	A32/A64
uint64_t vcvtah_u64_f16(float16_t a)	a -> Hn	FCVTAU Hd,Hn	Hd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16_t vcvtmh_s16_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTMS Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int32_t vcvtmh_s32_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTMS Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>int64_t vcvtmh_s64_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTMS Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>uint16_t vcvtmh_u16_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTMU Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>uint32_t vcvtmh_u32_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTMU Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>uint64_t vcvtmh_u64_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTMU Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int16_t vcvtnh_s16_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTNS Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int32_t vcvtnh_s32_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTNS Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>int64_t vcvtnh_s64_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTNS Hd,Hn</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16_t vcvtnh_u16_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTNU Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>uint32_t vcvtnh_u32_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTNU Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>uint64_t vcvtnh_u64_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTNU Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int16_t vcvtp_h_s16_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTPS Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>int32_t vcvtp_h_s32_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTPS Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>int64_t vcvtp_h_s64_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTPS Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>uint16_t vcvtp_u_u16_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTPU Hd,Hn</code>	<code>Hd -> result</code>	A64
<code>uint32_t vcvtp_u_u32_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTPU Hd,Hn</code>	<code>Hd -> result</code>	A32/A64
<code>uint64_t vcvtp_u_u64_f16(float16_t a)</code>	<code>a -> Hn</code>	<code>FCVTPU Hd,Hn</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vcvth_n_f16_s16(int16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	SCVTF Hd,Hn,#n	Hd -> result	A64
<code>float16_t vcvth_n_f16_s32(int32_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	SCVTF Hd,Hn,#n	Hd -> result	A32/A64
<code>float16_t vcvth_n_f16_s64(int64_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	SCVTF Hd,Hn,#n	Hd -> result	A64
<code>float16_t vcvth_n_f16_u16(uint16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	UCVTF Hd,Hn,#n	Hd -> result	A64
<code>float16_t vcvth_n_f16_u32(uint32_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	UCVTF Hd,Hn,#n	Hd -> result	A32/A64
<code>float16_t vcvth_n_f16_u64(uint64_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	UCVTF Hd,Hn,#n	Hd -> result	A64
<code>int16_t vcvth_n_s16_f16(float16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	FCVTZS Hd,Hn,#n	Hd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32_t vcvth_n_s32_f16(float16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	<code>FCVTZS Hd,Hn,#n</code>	<code>Hd -> result</code>	A32/A64
<code>int64_t vcvth_n_s64_f16(float16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	<code>FCVTZS Hd,Hn,#n</code>	<code>Hd -> result</code>	A64
<code>uint16_t vcvth_n_u16_f16(float16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	<code>FCVTZU Hd,Hn,#n</code>	<code>Hd -> result</code>	A64
<code>uint32_t vcvth_n_u32_f16(float16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	<code>FCVTZU Hd,Hn,#n</code>	<code>Hd -> result</code>	A32/A64
<code>uint64_t vcvth_n_u64_f16(float16_t a, const int n)</code>	<code>a -> Hn 1 <= n <= 16</code>	<code>FCVTZU Hd,Hn,#n</code>	<code>Hd -> result</code>	A64

2.5.4 Logical

2.5.4.1 Negate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vnegh_f16(float16_t a)	a -> Hn	FNEG Hd,Hn	Hd -> result	A32/A64

2.6 fp16 vector intrinsics (from ARMv8.2-A)

2.6.1 Vector arithmetic

2.6.1.1 Absolute

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vabs_f16(float16x4_t a)	a -> Vn.4H	FABS Vd.4H,Vn.4H	Vd.4H -> result	A32/A64
float16x8_t vabsq_f16(float16x8_t a)	a -> Vn.8H	FABS Vd.8H,Vn.8H	Vd.8H -> result	A32/A64

2.6.1.1.1 Absolute value

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vabd_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FABD Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A32/A64
float16x8_t vabdq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	FABD Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A32/A64

2.6.1.1.2 Absolute difference

2.6.1.2 Reciprocal

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vrecpe_f16(float16x4_t a)	a -> Vn.4H	FRECPE Vd.4H,Vn.4H	Vd.4H -> result	A32/A64
float16x8_t vrecpeq_f16(float16x8_t a)	a -> Vn.8H	FRECPE Vd.8H,Vn.8H	Vd.8H -> result	A32/A64

2.6.1.2.1 Reciprocal estimate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vrsqrte_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FRSQRTE Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vrsqrteq_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FRSQRTE Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vrsqrts_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FRSQRTS Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vrsqrtsq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FRSQRTS Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.1.2.2 Reciprocal square-root estimate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vrecps_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FRECPS Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vrecpsq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FRECPS Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.1.2.3 Reciprocal step

2.6.1.3 Rounding

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vrnd_f16(float16x4_t a)	a -> Vn.4H	FRINTZ Vd.4H,Vn.4H	Vd.4H -> result	A32/A64
float16x8_t vrndq_f16(float16x8_t a)	a -> Vn.8H	FRINTZ Vd.8H,Vn.8H	Vd.8H -> result	A32/A64
float16x4_t vrnda_f16(float16x4_t a)	a -> Vn.4H	FRINTA Vd.4H,Vn.4H	Vd.4H -> result	A32/A64
float16x8_t vrndaq_f16(float16x8_t a)	a -> Vn.8H	FRINTA Vd.8H,Vn.8H	Vd.8H -> result	A32/A64
float16x4_t vrndi_f16(float16x4_t a)	a -> Vn.4H	FRINTI Vd.4H,Vn.4H	Vd.4H -> result	A64
float16x8_t vrndiq_f16(float16x8_t a)	a -> Vn.8H	FRINTI Vd.8H,Vn.8H	Vd.8H -> result	A64
float16x4_t vrndm_f16(float16x4_t a)	a -> Vn.4H	FRINTM Vd.4H,Vn.4H	Vd.4H -> result	A32/A64
float16x8_t vrndmq_f16(float16x8_t a)	a -> Vn.8H	FRINTM Vd.8H,Vn.8H	Vd.8H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vrndn_f16(float16x4_t a)	a -> Vn.4H	FRINTN Vd.4H,Vn.4H	Vd.4H -> result	A32/A64
float16x8_t vrndnq_f16(float16x8_t a)	a -> Vn.8H	FRINTN Vd.8H,Vn.8H	Vd.8H -> result	A32/A64
float16x4_t vrndp_f16(float16x4_t a)	a -> Vn.4H	FRINTP Vd.4H,Vn.4H	Vd.4H -> result	A32/A64
float16x8_t vrndpq_f16(float16x8_t a)	a -> Vn.8H	FRINTP Vd.8H,Vn.8H	Vd.8H -> result	A32/A64
float16x4_t vrndx_f16(float16x4_t a)	a -> Vn.4H	FRINTX Vd.4H,Vn.4H	Vd.4H -> result	A32/A64
float16x8_t vrndxq_f16(float16x8_t a)	a -> Vn.8H	FRINTX Vd.8H,Vn.8H	Vd.8H -> result	A32/A64

2.6.1.4 Square root

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vsqrt_f16(float16x4_t a)	a -> Vn.4H	FSQRT Vd.4H,Vn.4H	Vd.4H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x8_t vsqrtq_f16(float16x8_t a)	a -> Vn.8H	FSQRT Vd.8H,Vn.8H	Vd.8H -> result	A64

2.6.1.5 Add

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vadd_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FADD Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A32/A64
float16x8_t vaddq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	FADD Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A32/A64

2.6.1.5.1 Addition

2.6.1.6 Division

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vdiv_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FDIV Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vdivq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FDIV Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64

2.6.1.7 Maximum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vmax_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FMAX Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vmaxq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FMAX Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vmaxnm_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FMAXNM Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x8_t vmaxnmq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	FMAXNM Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A32/A64
float16_t vmaxv_f16(float16x4_t a)	a -> Vn.4H	FMAXP Hd,Vn.4H	Hd -> result	A64
float16_t vmaxvq_f16(float16x8_t a)	a -> Vn.8H	FMAXP Hd,Vn.8H	Hd -> result	A64
float16_t vmaxnmv_f16(float16x4_t a)	a -> Vn.4H	FMAXNMP Hd,Vn.4H	Hd -> result	A64
float16_t vmaxnmvq_f16(float16x8_t a)	a -> Vn.8H	FMAXNMP Hd,Vn.8H	Hd -> result	A64

2.6.1.8 Minimum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vmin_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FMIN Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vminq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FMIN Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vminnm_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FMINNM Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vminnmq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FMINNM Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16_t vminv_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FMINP Hd,Vn.4H</code>	<code>Hd -> result</code>	A64
<code>float16_t vminvq_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FMINP Hd,Vn.8H</code>	<code>Hd -> result</code>	A64
<code>float16_t vminnmv_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FMINNMP Hd,Vn.4H</code>	<code>Hd -> result</code>	A64
<code>float16_t vminnmvq_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FMINNMP Hd,Vn.8H</code>	<code>Hd -> result</code>	A64

2.6.1.9 Multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vmul_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FMUL Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vmulq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FMUL Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vmul_lane_f16(float16x4_t a, float16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>FMUL Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vmulq_lane_f16(float16x8_t a, float16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>FMUL Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vmul_laneq_f16(float16x4_t a, float16x8_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>FMUL Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vmulq_laneq_f16(float16x8_t a, float16x8_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>FMUL Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vmul_n_f16(float16x4_t a, float16_t n)</code>	<code>n -> Vm.H[0] a -> Vn.4H</code>	<code>FMUL Vd.4H,Vn.4H,Vm.H[0]</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vmulq_n_f16(float16x8_t a, float16_t n)</code>	<code>n -> Vm.H[0] a -> Vn.8H</code>	<code>FMUL Vd.8H,Vn.8H,Vm.H[0]</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16_t vmulh_lane_f16(float16_t a, float16x4_t v, const int lane)</code>	<code>a -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>FMUL Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64
<code>float16_t vmulh_laneq_f16(float16_t a, float16x8_t v, const int lane)</code>	<code>a -> Hn v -> Vm.8H 0 <= lane <= 7</code>	<code>FMUL Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64

2.6.1.9.1 Multiplication

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vmulx_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FMULX Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vmulxq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FMULX Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>float16x4_t vmulx_lane_f16(float16x4_t a, float16x4_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>FMULX Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vmulxq_lane_f16(float16x8_t a, float16x4_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>FMULX Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>float16x4_t vmulx_laneq_f16(float16x4_t a, float16x8_t v, const int lane)</code>	<code>a -> Vn.4H v -> Vm.8H 0 <= lane <= 7</code>	<code>FMULX Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vmulxq_laneq_f16(float16x8_t a, float16x8_t v, const int lane)</code>	<code>a -> Vn.8H v -> Vm.8H 0 <= lane <= 7</code>	<code>FMULX Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>float16x4_t vmulx_n_f16(float16x4_t a, float16_t n)</code>	<code>n -> Vm.H[0] a -> Vn.4H</code>	<code>FMULX Vd.4H,Vn.4H,Vm.H[0]</code>	<code>Vd.4H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vmulxq_n_f16(float16x8_t a, float16_t n)</code>	<code>n -> Vm.H[0] a -> Vn.8H</code>	<code>FMULX Vd.8H,Vn.8H,Vm.H[0]</code>	<code>Vd.8H -> result</code>	A64
<code>float16_t vmulxh_lane_f16(float16_t a, float16x4_t v, const int lane)</code>	<code>a -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>FMULX Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64
<code>float16_t vmulxh_laneq_f16(float16_t a, float16x8_t v, const int lane)</code>	<code>a -> Hn v -> Vm.8H 0 <= lane <= 7</code>	<code>FMULX Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64

2.6.1.9.2 Multiply extended

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vfma_f16(float16x4_t a, float16x4_t b, float16x4_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.4H</code>	<code>FMLA Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vfmaq_f16(float16x8_t a, float16x8_t b, float16x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.8H</code>	<code>FMLA Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vfms_f16(float16x4_t a, float16x4_t b, float16x4_t c)</code>	<code>a -> Vd.4H b -> Vn.4H c -> Vm.4H</code>	<code>FMLS Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vfmsq_f16(float16x8_t a, float16x8_t b, float16x8_t c)</code>	<code>a -> Vd.8H b -> Vn.8H c -> Vm.8H</code>	<code>FMLS Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vfma_lane_f16(float16x4_t a, float16x4_t b, float16x4_t v, const int lane)</code>	<code>a -> Vd.4H b -> Vn.4H v -> Vm.4H 0 <= lane <= 3</code>	<code>FMLA Vd.4H,Vn.4H,Vm.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vfmaq_lane_f16(float16x8_t a, float16x8_t b, float16x4_t v, const int lane)</code>	<code>a -> Vd.8H b -> Vn.8H v -> Vm.4H 0 <= lane <= 3</code>	<code>FMLA Vd.8H,Vn.8H,Vm.H[lane]</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vfma_laneq_f16(float16x4_t a, float16x4_t b, float16x8_t v, const int lane)	a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	FMLA Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
float16x8_t vfmaq_laneq_f16(float16x8_t a, float16x8_t b, float16x8_t v, const int lane)	a -> Vd.8H b -> Vn.8H v -> Vm.8H 0 <= lane <= 7	FMLA Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64
float16x4_t vfma_n_f16(float16x4_t a, float16x4_t b, float16_t n)	n -> Vm.H[0] b -> Vn.4H a -> Vd.4H	FMLA Vd.4H,Vn.4H,Vm.H[0]	Vd.4H -> result	A64
float16x8_t vfmaq_n_f16(float16x8_t a, float16x8_t b, float16_t n)	n -> Vm.H[0] b -> Vn.8H a -> Vd.8H	FMLA Vd.8H,Vn.8H,Vm.H[0]	Vd.8H -> result	A64
float16_t vfma_lane_f16(float16_t a, float16_t b, float16x4_t v, const int lane)	a -> Hd b -> Hn v -> Vm.4H 0 <= lane <= 3	FMLA Hd,Hn,Vm.H[lane]	Hd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16_t vfмах_laneq_f16(float16_t a, float16_t b, float16x8_t v, const int lane)	a -> Hd b -> Hn v -> Vm.8H 0 <= lane <= 7	FMLA Hd,Hn,Vm.H[lane]	Hd -> result	A64
float16x4_t vfms_lane_f16(float16x4_t a, float16x4_t b, float16x4_t v, const int lane)	a -> Vd.4H b -> Vn.4H v -> Vm.4H 0 <= lane <= 3	FMLS Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
float16x8_t vfmsq_lane_f16(float16x8_t a, float16x8_t b, float16x4_t v, const int lane)	a -> Vd.8H b -> Vn.8H v -> Vm.4H 0 <= lane <= 3	FMLS Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64
float16x4_t vfms_laneq_f16(float16x4_t a, float16x4_t b, float16x8_t v, const int lane)	a -> Vd.4H b -> Vn.4H v -> Vm.8H 0 <= lane <= 7	FMLS Vd.4H,Vn.4H,Vm.H[lane]	Vd.4H -> result	A64
float16x8_t vfmsq_laneq_f16(float16x8_t a, float16x8_t b, float16x8_t v, const int lane)	a -> Vd.8H b -> Vn.8H v -> Vm.8H 0 <= lane <= 7	FMLS Vd.8H,Vn.8H,Vm.H[lane]	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vfms_n_f16(float16x4_t a, float16x4_t b, float16_t n)</code>	<code>n -> Vm.H[0] b -> Vn.4H a -> Vd.4H</code>	<code>FMLS Vd.4H,Vn.4H,Vm.H[0]</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vfmsq_n_f16(float16x8_t a, float16x8_t b, float16_t n)</code>	<code>n -> Vm.H[0] b -> Vn.8H a -> Vd.8H</code>	<code>FMLS Vd.8H,Vn.8H,Vm.H[0]</code>	<code>Vd.8H -> result</code>	A64
<code>float16_t vfms_h_lane_f16(float16_t a, float16_t b, float16x4_t v, const int lane)</code>	<code>a -> Hd b -> Hn v -> Vm.4H 0 <= lane <= 3</code>	<code>FMLS Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64
<code>float16_t vfms_h_laneq_f16(float16_t a, float16_t b, float16x8_t v, const int lane)</code>	<code>a -> Hd b -> Hn v -> Vm.8H 0 <= lane <= 7</code>	<code>FMLS Hd,Hn,Vm.H[lane]</code>	<code>Hd -> result</code>	A64

2.6.1.9.3 Fused multiply-accumulate

2.6.1.10 Pairwise arithmetic

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vpadd_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FADDP Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A32/A64
float16x8_t vpaddq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	FADDP Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64

2.6.1.10.1 Pairwise addition

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vpmask_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FMAXP Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A32/A64
float16x8_t vpmaskq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	FMAXP Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64
float16x4_t vpmasknm_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FMAXNMP Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x8_t vpmxnmq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	FMAXNMP Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64

2.6.1.10.2 Pairwise maximum

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vpmi_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FMINP Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A32/A64
float16x8_t vpmiq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	FMINP Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64
float16x4_t vpminnm_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FMINNMP Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
float16x8_t vpminnmq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	FMINNMP Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64

2.6.1.10.3 Pairwise minimum

2.6.1.11 Subtract

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vsub_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FSUB Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vsubq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FSUB Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.1.11.1 Subtraction**2.6.2 Compare****2.6.2.1 Bitwise equal to zero**

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vceqz_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCMEQ Vd.4H,Vn.4H,#0</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vceqzq_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCMEQ Vd.8H,Vn.8H,#0</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.2 Greater than or equal to zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x4_t vcgez_f16(float16x4_t a)	a -> Vn.4H	FCMGE Vd.4H,Vn.4H,#0	Vd.4H -> result	A32/A64
uint16x8_t vcgezq_f16(float16x8_t a)	a -> Vn.8H	FCMGE Vd.8H,Vn.8H,#0	Vd.8H -> result	A32/A64

2.6.2.3 Greater than zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x4_t vcgtz_f16(float16x4_t a)	a -> Vn.4H	FCMGT Vd.4H,Vn.4H,#0	Vd.4H -> result	A32/A64
uint16x8_t vcgtzq_f16(float16x8_t a)	a -> Vn.8H	FCMGT Vd.8H,Vn.8H,#0	Vd.8H -> result	A32/A64

2.6.2.4 Less than or equal to zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x4_t vclez_f16(float16x4_t a)	a -> Vn.4H	FCMLE Vd.4H,Vn.4H,#0	Vd.4H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x8_t vclezq_f16(float16x8_t a)	a -> Vn.8H	FCMLE Vd.8H,Vn.8H,#0	Vd.8H -> result	A32/A64

2.6.2.5 Less than zero

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x4_t vcltz_f16(float16x4_t a)	a -> Vn.4H	FCMLT Vd.4H,Vn.4H,#0	Vd.4H -> result	A32/A64
uint16x8_t vcltzq_f16(float16x8_t a)	a -> Vn.8H	FCMLT Vd.8H,Vn.8H,#0	Vd.8H -> result	A32/A64

2.6.2.6 Absolute greater than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
uint16x4_t vcage_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	FACGE Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vcageq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	<code>FACGE Vd.8H, Vn.8H, Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.7 Absolute greater than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcagt_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H</code> <code>b -> Vm.4H</code>	<code>FACGT Vd.4H, Vn.4H, Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcagtq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	<code>FACGT Vd.8H, Vn.8H, Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.8 Absolute less than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcale_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H</code> <code>b -> Vm.4H</code>	<code>FACGE Vd.4H, Vn.4H, Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vcaleq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FACGE Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.9 Absolute less than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcalt_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FACGT Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcaltq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FACGT Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.10 Equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vceq_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FCMEQ Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vceqq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FCMEQ Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.11 Greater than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcge_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FCMGE Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcgeq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FCMGE Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.12 Greater than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcgt_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FCMGT Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vcgtq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FCMGT Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.13 Less than or equal to

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcle_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FCMGE Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcleq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FCMGE Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.2.14 Less than

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vclt_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FCMGT Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vcltq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	<code>FCMGT Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.3 Data type conversion

2.6.3.1 Conversions

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vcvt_f16_s16(int16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>SCVTF Vd.4H,Vn.4H,#0</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vcvtq_f16_s16(int16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>SCVTF Vd.8H,Vn.8H,#0</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vcvt_f16_u16(uint16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>UCVTF Vd.4H,Vn.4H,#0</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vcvtq_f16_u16(uint16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>UCVTF Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>int16x4_t vcvt_s16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTZS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t vcvttq_s16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTZS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>uint16x4_t vcvtt_u16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTZS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcvttq_u16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTZS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>int16x4_t vcvta_s16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTAS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>int16x8_t vcvtaq_s16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTAS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>uint16x4_t vcvta_u16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTAU Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcvtaq_u16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTAU Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>int16x4_t vcvtm_s16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTMS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>int16x8_t vcvtmq_s16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTMS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcvtm_u16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTMU Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcvtmq_u16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTMU Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>int16x4_t vcvtn_s16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTNS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>int16x8_t vcvtnq_s16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTNS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>uint16x4_t vcvtn_u16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTNU Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcvtnq_u16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTNU Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>int16x4_t vcvtp_s16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTPS Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>int16x8_t vcvtpq_s16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTPS Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>uint16x4_t vcvtp_u16_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FCVTPU Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x8_t vcvtpq_u16_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FCVTPU Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vcvtn_f16_s16(int16x4_t a, const int n)</code>	<code>a -> Vn.4H</code> <code>1 <= n <= 16</code>	<code>SCVTF Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vcvtq_n_f16_s16(int16x8_t a, const int n)</code>	<code>a -> Vn.8H</code> <code>1 <= n <= 16</code>	<code>SCVTF Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float16x4_t vcvtn_f16_u16(uint16x4_t a, const int n)</code>	<code>a -> Vn.4H</code> <code>1 <= n <= 16</code>	<code>UCVTF Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vcvtq_n_f16_u16(uint16x8_t a, const int n)</code>	<code>a -> Vn.8H</code> <code>1 <= n <= 16</code>	<code>UCVTF Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	A32/A64
<code>int16x4_t vcvtn_s16_f16(float16x4_t a, const int n)</code>	<code>a -> Vn.4H</code> <code>1 <= n <= 16</code>	<code>FCVTZS Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	A32/A64
<code>int16x8_t vcvtq_n_s16_f16(float16x8_t a, const int n)</code>	<code>a -> Vn.8H</code> <code>1 <= n <= 16</code>	<code>FCVTZS Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint16x4_t vcvtn_u16_f16(float16x4_t a, const int n)</code>	<code>a -> Vn.4H</code> <code>1 <= n <= 16</code>	<code>FCVTZU Vd.4H,Vn.4H,#n</code>	<code>Vd.4H -> result</code>	A32/A64
<code>uint16x8_t vcvttq_n_u16_f16(float16x8_t a, const int n)</code>	<code>a -> Vn.8H</code> <code>1 <= n <= 16</code>	<code>FCVTZU Vd.8H,Vn.8H,#n</code>	<code>Vd.8H -> result</code>	A32/A64

2.6.4 Logical

2.6.4.1 Negate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vneg_f16(float16x4_t a)</code>	<code>a -> Vn.4H</code>	<code>FNEG Vd.4H,Vn.4H</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float16x8_t vnegq_f16(float16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>FNEG Vd.8H,Vn.8H</code>	<code>Vd.8H -> result</code>	A32/A64

2.7 Additional intrinsics added in ACLE 3.0 for data processing (Always available)

2.7.1 Bit manipulation

2.7.1.1 Bitwise select

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vbsl_f16(uint16x4_t a, float16x4_t b, float16x4_t c)	a -> Vd.8B b -> Vn.8B c -> Vm.8B	BSL Vd.8B,Vn.8B,Vm.8B	Vd.8B -> result	v7/A32/A64
float16x8_t vbslq_f16(uint16x8_t a, float16x8_t b, float16x8_t c)	a -> Vd.16B b -> Vn.16B c -> Vm.16B	BSL Vd.16B,Vn.16B,Vm.16B	Vd.16B -> result	v7/A32/A64

2.7.2 Vector manipulation

2.7.2.1 Zip elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4x2_t vzip_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	ZIP1 Vd1.4H,Vn.4H,Vm.4H ZIP2 Vd2.4H,Vn.4H,Vm.4H	Vd1.4H -> result.val[0] Vd2.4H -> result.val[1]	v7/A32/A64
float16x8x2_t vzipq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	ZIP1 Vd1.8H,Vn.8H,Vm.8H ZIP2 Vd2.8H,Vn.8H,Vm.8H	Vd1.8H -> result.val[0] Vd2.8H -> result.val[1]	v7/A32/A64
float16x4_t vzip1_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	ZIP1 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vzip1q_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	ZIP1 <code>Vd.8H, Vn.8H, Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>float16x4_t vzip2_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	ZIP2 <code>Vd.4H, Vn.4H, Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vzip2q_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	ZIP2 <code>Vd.8H, Vn.8H, Vm.8H</code>	<code>Vd.8H -> result</code>	A64

2.7.2.2 Unzip elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4x2_t vuzp_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	UZIP1 <code>Vd1.4H, Vn.4H, Vm.4H</code> UZIP2 <code>Vd2.4H, Vn.4H, Vm.4H</code>	<code>Vd1.4H -> result.val[0]</code> <code>Vd2.4H -> result.val[1]</code>	v7/A32/A64
<code>float16x8x2_t vuzpq_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	UZIP1 <code>Vd1.8H, Vn.8H, Vm.8H</code> UZIP2 <code>Vd2.8H, Vn.8H, Vm.8H</code>	<code>Vd1.8H -> result.val[0]</code> <code>Vd2.8H -> result.val[1]</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vuzp1_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP1 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vuzp1q_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UZP1 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64
<code>float16x4_t vuzp2_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>UZP2 Vd.4H,Vn.4H,Vm.4H</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vuzp2q_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>UZP2 Vd.8H,Vn.8H,Vm.8H</code>	<code>Vd.8H -> result</code>	A64

2.7.2.3 Transpose elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4x2_t vtrn_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>TRN1 Vd1.4H,Vn.4H,Vm.4H TRN2 Vd2.4H,Vn.4H,Vm.4H</code>	<code>Vd1.4H -> result.val[0] Vd2.4H -> result.val[1]</code>	v7/A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x8x2_t vtrnq_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	TRN1 Vd1.8H,Vn.8H,Vm.8H TRN2 Vd2.8H,Vn.8H,Vm.8H	Vd1.8H -> result.val[0] Vd2.8H -> result.val[1]	v7/A32/A64
float16x4_t vtrn1_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	TRN1 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
float16x8_t vtrn1q_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	TRN1 Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64
float16x4_t vtrn2_f16(float16x4_t a, float16x4_t b)	a -> Vn.4H b -> Vm.4H	TRN2 Vd.4H,Vn.4H,Vm.4H	Vd.4H -> result	A64
float16x8_t vtrn2q_f16(float16x8_t a, float16x8_t b)	a -> Vn.8H b -> Vm.8H	TRN2 Vd.8H,Vn.8H,Vm.8H	Vd.8H -> result	A64

2.7.2.4 Set all lanes to the same value

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vdup_n_f16(float16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4H,rn</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>float16x8_t vdupq_n_f16(float16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8H,rn</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>float16x4_t vdup_lane_f16(float16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	v7/A32/A64
<code>float16x8_t vdupq_lane_f16(float16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	v7/A32/A64
<code>float16x4_t vdup_laneq_f16(float16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	A64
<code>float16x8_t vdupq_laneq_f16(float16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	A64
<code>float16_t vduph_lane_f16(float16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Hd,Vn.H[lane]</code>	<code>Hd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16_t vduph_laneq_f16(float16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Hd,Vn.H[lane]</code>	<code>Hd -> result</code>	A64

2.7.2.5 Extract vector from a pair of vectors

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vext_f16(float16x4_t a, float16x4_t b, const int n)</code>	<code>a -> Vn.8B b -> Vm.8B 0 <= n <= 3</code>	<code>EXT Vd.8B,Vn.8B,Vm.8B,#(n<<1)</code>	<code>Vd.8B -> result</code>	v7/A32/A64
<code>float16x8_t vextq_f16(float16x8_t a, float16x8_t b, const int n)</code>	<code>a -> Vn.16B b -> Vm.16B 0 <= n <= 7</code>	<code>EXT Vd.16B,Vn.16B,Vm.16B,#(n<<1)</code>	<code>Vd.16B -> result</code>	v7/A32/A64

2.7.2.6 Reverse elements

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vrev64_f16(float16x4_t vec)	vec -> Vn.4H	REV64 Vd.4H,Vn.4H	Vd.4H -> result	v7/A32/A64
float16x8_t vrev64q_f16(float16x8_t vec)	vec -> Vn.8H	REV64 Vd.8H,Vn.8H	Vd.8H -> result	v7/A32/A64

2.7.3 Move

2.7.3.1 Vector move

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vmov_n_f16(float16_t value)	value -> rn	DUP Vd.4H,rn	Vd.4H -> result	v7/A32/A64
float16x8_t vmovq_n_f16(float16_t value)	value -> rn	DUP Vd.8H,rn	Vd.8H -> result	v7/A32/A64

2.8 Dot Product intrinsics added for ARMv8.2-a and newer. Requires the +dotprod architecture extension.

2.8.1 Vector arithmetic

2.8.1.1 Dot product

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x2_t vdot_u32(uint32x2_t r, uint8x8_t a, uint8x8_t b)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.8B</code>	<code>UDOT Vd.2S,Vn.8B,Vm.8B</code>	<code>Vd.2S -> result</code>	A32/A64
<code>int32x2_t vdot_s32(int32x2_t r, int8x8_t a, int8x8_t b)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.8B</code>	<code>SDOT Vd.2S,Vn.8B,Vm.8B</code>	<code>Vd.2S -> result</code>	A32/A64
<code>uint32x4_t vdotq_u32(uint32x4_t r, uint8x16_t a, uint8x16_t b)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.16B</code>	<code>UDOT Vd.4S,Vn.16B,Vm.16B</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x4_t vdotq_s32(int32x4_t r, int8x16_t a, int8x16_t b)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.16B</code>	<code>SDOT Vd.4S,Vn.16B,Vm.16B</code>	<code>Vd.4S -> result</code>	A32/A64
<code>uint32x2_t vdot_lane_u32(uint32x2_t r, uint8x8_t a, uint8x8_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 1</code>	<code>UDOT Vd.2S,Vn.8B,Vm.4B[lane]</code>	<code>Vd.2S -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vdot_lane_s32(int32x2_t r, int8x8_t a, int8x8_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 1</code>	<code>SDOT Vd.2S,Vn.8B,Vm.4B[lane]</code>	<code>Vd.2S -> result</code>	A32/A64
<code>uint32x4_t vdotq_laneq_u32(uint32x4_t r, uint8x16_t a, uint8x16_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.4B 0 <= lane <= 3</code>	<code>UDOT Vd.4S,Vn.16B,Vm.4B[lane]</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x4_t vdotq_laneq_s32(int32x4_t r, int8x16_t a, int8x16_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.4B 0 <= lane <= 3</code>	<code>SDOT Vd.4S,Vn.16B,Vm.4B[lane]</code>	<code>Vd.4S -> result</code>	A32/A64
<code>uint32x2_t vdot_laneq_u32(uint32x2_t r, uint8x8_t a, uint8x16_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 3</code>	<code>UDOT Vd.2S,Vn.8B,Vm.4B[lane]</code>	<code>Vd.2S -> result</code>	A32/A64
<code>int32x2_t vdot_laneq_s32(int32x2_t r, int8x8_t a, int8x16_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 3</code>	<code>SDOT Vd.2S,Vn.8B,Vm.4B[lane]</code>	<code>Vd.2S -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vdotq_lane_u32(uint32x4_t r, uint8x16_t a, uint8x8_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.4B 0 <= lane <= 1</code>	<code>UDOT Vd.4S,Vn.16B,Vm.4B[lane]</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x4_t vdotq_lane_s32(int32x4_t r, int8x16_t a, int8x8_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.4B 0 <= lane <= 1</code>	<code>SDOT Vd.4S,Vn.16B,Vm.4B[lane]</code>	<code>Vd.4S -> result</code>	A32/A64

2.9 Armv8.4-a intrinsics.

2.9.1 Cryptography

2.9.1.1 SHA512

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vsha512hq_u64(uint64x2_t hash_ed, uint64x2_t hash_gf, uint64x2_t kwh_kwh2)</code>	<code>hash_ed -> Qd hash_gf -> Qn kwh_kwh2 -> Vm.2D</code>	<code>SHA512H Qd, Qn, Vm.2D</code>	<code>Qd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vsha512h2q_u64(uint64x2_t sum_ab, uint64x2_t hash_c_, uint64x2_t hash_ab)</code>	<code>sum_ab -> Qd hash_c_ -> Qn hash_ab -> Vm.2D</code>	SHA512H2 Qd, Qn, Vm.2D	Qd -> result	A64
<code>uint64x2_t vsha512su0q_u64(uint64x2_t w0_1, uint64x2_t w2_)</code>	<code>w0_1 -> Vd.2D w2_ -> Vn.2D</code>	SHA512SU0 Vd.2D, Vn.2D	Vd.2D -> result	A64
<code>uint64x2_t vsha512su1q_u64(uint64x2_t s01_s02, uint64x2_t w14_15, uint64x2_t w9_10)</code>	<code>s01_s02 -> Vd.2D w14_15 -> Vn.2D w9_10 -> Vm.2D</code>	SHA512SU1 Vd.2D, Vn.2D, Vm.2D	Vd.2D -> result	A64

2.9.1.2 SM3

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsm3ss1q_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vn.4S b -> Vm.4S c -> Va.4S</code>	SM3SS1 Vd.4S, Vn.4S, Vm.4S, Va.4S	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsm3tt1aq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c, __builtin_constant_p (imm2))</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S 0 <= imm2 <= 3</code>	SM3TT1A Vd.4S, Vn.4S, Vm.4S[imm2]	Vd.4S -> result	A64
<code>uint32x4_t vsm3tt1bq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c, __builtin_constant_p (imm2))</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S 0 <= imm2 <= 3</code>	SM3TT1B Vd.4S, Vn.4S, Vm.4S[imm2]	Vd.4S -> result	A64
<code>uint32x4_t vsm3tt2aq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c, __builtin_constant_p (imm2))</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S 0 <= imm2 <= 3</code>	SM3TT2A Vd.4S, Vn.4S, Vm.4S[imm2]	Vd.4S -> result	A64
<code>uint32x4_t vsm3tt2bq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c, __builtin_constant_p (imm2))</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S 0 <= imm2 <= 3</code>	SM3TT2B Vd.4S, Vn.4S, Vm.4S[imm2]	Vd.4S -> result	A64
<code>uint32x4_t vsm3partw1q_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S</code>	SM3PARTW1 Vd.4S, Vn.4S, Vm.4S	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsm3partw2q_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vd.4S b -> Vn.4S c -> Vm.4S</code>	<code>SM3PARTW2 Vd.4S, Vn.4S, Vm.4S</code>	<code>Vd.4S -> result</code>	A64

2.9.1.3 SM4

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vsm4eq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vd.4S b -> Vn.4S</code>	<code>SM4E Vd.4S, Vn.4S</code>	<code>Vd.4S -> result</code>	A64
<code>uint32x4_t vsm4ekeyq_u32(uint32x4_t a, uint32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>SM4EKEY Vd.4S, Vn.4S, Vm.4S</code>	<code>Vd.4S -> result</code>	A64

2.9.2 Logical

2.9.2.1 Exclusive OR

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t veor3q_u8(uint8x16_t a, uint8x16_t b, uint8x16_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	EOR3 Vd.16B, Vn.16B, Vm.16B, Va.16B	Vd.16B -> result	A64
<code>uint16x8_t veor3q_u16(uint16x8_t a, uint16x8_t b, uint16x8_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	EOR3 Vd.16B, Vn.16B, Vm.16B, Va.16B	Vd.16B -> result	A64
<code>uint32x4_t veor3q_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	EOR3 Vd.16B, Vn.16B, Vm.16B, Va.16B	Vd.16B -> result	A64
<code>uint64x2_t veor3q_u64(uint64x2_t a, uint64x2_t b, uint64x2_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	EOR3 Vd.16B, Vn.16B, Vm.16B, Va.16B	Vd.16B -> result	A64
<code>int8x16_t veor3q_s8(int8x16_t a, int8x16_t b, int8x16_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	EOR3 Vd.16B, Vn.16B, Vm.16B, Va.16B	Vd.16B -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int16x8_t veor3q_s16(int16x8_t a, int16x8_t b, int16x8_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>EOR3 Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int32x4_t veor3q_s32(int32x4_t a, int32x4_t b, int32x4_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>EOR3 Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int64x2_t veor3q_s64(int64x2_t a, int64x2_t b, int64x2_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>EOR3 Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64

2.9.2.2 Rotate and exclusive OR

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vrax1q_u64(uint64x2_t a, uint64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>RAX1 Vd.2D, Vn.2D, Vm.2D</code>	<code>Vd.2D -> result</code>	A64

2.9.2.3 Exclusive OR and rotate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vvarq_u64(uint64x2_t a, uint64x2_t b, __builtin_constant_p (imm6))</code>	<code>a -> Vn.2D b -> Vm.2D 0 <= imm6 <= 63</code>	<code>XAR Vd.2D, Vn.2D, Vm.2D, imm6</code>	<code>Vd.2D -> result</code>	A64

2.9.2.4 Bit clear and exclusive OR

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint8x16_t vbcaxq_u8(uint8x16_t a, uint8x16_t b, uint8x16_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>BCAX Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint16x8_t vbcaxq_u16(uint16x8_t a, uint16x8_t b, uint16x8_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>BCAX Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64
<code>uint32x4_t vbcaxq_u32(uint32x4_t a, uint32x4_t b, uint32x4_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>BCAX Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint64x2_t vbcaxq_u64(uint64x2_t a, uint64x2_t b, uint64x2_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>BCAX Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int8x16_t vbcaxq_s8(int8x16_t a, int8x16_t b, int8x16_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>BCAX Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int16x8_t vbcaxq_s16(int16x8_t a, int16x8_t b, int16x8_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>BCAX Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int32x4_t vbcaxq_s32(int32x4_t a, int32x4_t b, int32x4_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>BCAX Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64
<code>int64x2_t vbcaxq_s64(int64x2_t a, int64x2_t b, int64x2_t c)</code>	<code>a -> Vn.16B b -> Vm.16B c -> Va.16B</code>	<code>BCAX Vd.16B, Vn.16B, Vm.16B, Va.16B</code>	<code>Vd.16B -> result</code>	A64

2.10 FP16 Armv8.4-a

2.10.1 Vector arithmetic

2.10.1.1 Multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vfm1al_low_f16(float32x2_t r, float16x4_t a, float16x4_t b)</code>	<code>r -> Vd.2S a -> Vd.2H b -> Vd.2H</code>	<code>FMLAL Vd.2S, Vn.2H, Vm.2H</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float32x2_t vfm1sl_low_f16(float32x2_t r, float16x4_t a, float16x4_t b)</code>	<code>r -> Vd.2S a -> Vd.2H b -> Vd.2H</code>	<code>FMLSL Vd.2S, Vn.2H, Vm.2H</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float32x4_t vfm1alq_low_f16(float32x4_t r, float16x8_t a, float16x8_t b)</code>	<code>r -> Vd.4S a -> Vd.4H b -> Vd.4H</code>	<code>FMLAL Vd.4S, Vn.4H, Vm.4H</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float32x4_t vfm1slq_low_f16(float32x4_t r, float16x8_t a, float16x8_t b)</code>	<code>r -> Vd.4S a -> Vd.4H b -> Vd.4H</code>	<code>FMLSL Vd.4S, Vn.4H, Vm.4H</code>	<code>Vd.4S -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vfmlal_high_f16(float32x2_t r, float16x4_t a, float16x4_t b)</code>	<code>r -> Vd.2S a -> Vd.2H b -> Vd.2H</code>	<code>FMLAL2 Vd.2S, Vn.2H, Vm.2H</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float32x2_t vfmlsl_high_f16(float32x2_t r, float16x4_t a, float16x4_t b)</code>	<code>r -> Vd.2S a -> Vd.2H b -> Vd.2H</code>	<code>FMLSL2 Vd.2S, Vn.2H, Vm.2H</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float32x4_t vfmlalq_high_f16(float32x4_t r, float16x8_t a, float16x8_t b)</code>	<code>r -> Vd.4S a -> Vd.4H b -> Vd.4H</code>	<code>FMLAL2 Vd.4S, Vn.4H, Vm.4H</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float32x4_t vfmlslq_high_f16(float32x4_t r, float16x8_t a, float16x8_t b)</code>	<code>r -> Vd.4S a -> Vd.4H b -> Vd.4H</code>	<code>FMLSL2 Vd.4S, Vn.4H, Vm.4H</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float32x2_t vfmlal_lane_low_f16(float32x2_t r, float16x4_t a, float16x4_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vd.2H b -> Vm.H 0 <= lane <= 3</code>	<code>FMLAL Vd.2S, Vn.2H, Vm.H[lane]</code>	<code>Vd.2S -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vfm1al_laneq_low_f16(float32x2_t r, float16x4_t a, float16x8_t b, const int lane)	r -> Vd.2S a -> Vd.2H b -> Vm.H 0 <= lane <= 7	FMLAL Vd.2S, Vn.2H, Vm.H[lane]	Vd.2S -> result	A32/A64
float32x4_t vfm1alq_lane_low_f16(float32x4_t r, float16x8_t a, float16x4_t b, const int lane)	r -> Vd.4S a -> Vd.4H b -> Vm.H 0 <= lane <= 3	FMLAL Vd.4S, Vn.4H, Vm.H[lane]	Vd.4S -> result	A32/A64
float32x4_t vfm1alq_laneq_low_f16(float32x4_t r, float16x8_t a, float16x8_t b, const int lane)	r -> Vd.4S a -> Vd.4H b -> Vm.H 0 <= lane <= 7	FMLAL Vd.4S, Vn.4H, Vm.H[lane]	Vd.4S -> result	A32/A64
float32x2_t vfm1sl_lane_low_f16(float32x2_t r, float16x4_t a, float16x4_t b, const int lane)	r -> Vd.2S a -> Vd.2H b -> Vm.H 0 <= lane <= 3	FMLSL Vd.2S, Vn.2H, Vm.H[lane]	Vd.2S -> result	A32/A64
float32x2_t vfm1sl_laneq_low_f16(float32x2_t r, float16x4_t a, float16x8_t b, const int lane)	r -> Vd.2S a -> Vd.2H b -> Vm.H 0 <= lane <= 7	FMLSL Vd.2S, Vn.2H, Vm.H[lane]	Vd.2S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vfmlslq_lane_low_f16(float32x4_t r, float16x8_t a, float16x4_t b, const int lane)	r -> Vd.4S a -> Vd.4H b -> Vm.H 0 <= lane <= 3	FMLSL Vd.4S, Vn.4H, Vm.H[lane]	Vd.4S -> result	A32/A64
float32x4_t vfmlslq_laneq_low_f16(float32x4_t r, float16x8_t a, float16x8_t b, const int lane)	r -> Vd.4S a -> Vd.4H b -> Vm.H 0 <= lane <= 7	FMLSL Vd.4S, Vn.4H, Vm.H[lane]	Vd.4S -> result	A32/A64
float32x2_t vfmlal_lane_high_f16(float32x2_t r, float16x4_t a, float16x4_t b, const int lane)	r -> Vd.2S a -> Vd.2H b -> Vm.H 0 <= lane <= 3	FMLAL2 Vd.2S, Vn.2H, Vm.H[lane]	Vd.2S -> result	A32/A64
float32x2_t vfmlsl_lane_high_f16(float32x2_t r, float16x4_t a, float16x4_t b, const int lane)	r -> Vd.2S a -> Vd.2H b -> Vm.H 0 <= lane <= 3	FMLSL2 Vd.2S, Vn.2H, Vm.H[lane]	Vd.2S -> result	A32/A64
float32x4_t vfmlalq_lane_high_f16(float32x4_t r, float16x8_t a, float16x4_t b, const int lane)	r -> Vd.4S a -> Vd.4H b -> Vm.H 0 <= lane <= 3	FMLAL2 Vd.4S, Vn.4H, Vm.H[lane]	Vd.4S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vfmlslq_lane_high_f16(float32x4_t r, float16x8_t a, float16x4_t b, const int lane)	r -> Vd.4S a -> Vd.4H b -> Vm.H 0 <= lane <= 3	FMLSL2 Vd.4S, Vn.4H, Vm.H[lane]	Vd.4S -> result	A32/A64
float32x2_t vfmlal_laneq_high_f16(float32x2_t r, float16x4_t a, float16x8_t b, const int lane)	r -> Vd.2S a -> Vd.2H b -> Vm.H 0 <= lane <= 7	FMLAL2 Vd.2S, Vn.2H, Vm.H[lane]	Vd.2S -> result	A32/A64
float32x2_t vfmlsl_laneq_high_f16(float32x2_t r, float16x4_t a, float16x8_t b, const int lane)	r -> Vd.2S a -> Vd.2H b -> Vm.H 0 <= lane <= 7	FMLSL2 Vd.2S, Vn.2H, Vm.H[lane]	Vd.2S -> result	A32/A64
float32x4_t vfmlalq_laneq_high_f16(float32x4_t r, float16x8_t a, float16x8_t b, const int lane)	r -> Vd.4S a -> Vd.4H b -> Vm.H 0 <= lane <= 7	FMLAL2 Vd.4S, Vn.4H, Vm.H[lane]	Vd.4S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vfm1slq_laneq_high_f16(r -> Vd.4S float32x4_t r, float16x8_t a, float16x8_t b, const int lane)</code>	<code>a -> Vd.4H b -> Vm.H 0 <= lane <= 7</code>	<code>FMLSL2 Vd.4S, Vn.4H, Vm.H[lane]</code>	<code>Vd.4S -> result</code>	A32/A64

2.10.1.1.1 Fused multiply-accumulate

2.11 Complex operations from Armv8.3-a

2.11.1 Complex arithmetic

2.11.1.1 Complex addition

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vcadd_rot90_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FCADD Vd.4H, Vn.4H, Vm.4H, #90</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float32x2_t vcadd_rot90_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FCADD Vd.2S, Vn.2S, Vm.2S, #90</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float16x8_t vcaddq_rot90_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FCADD Vd.8H, Vn.8H, Vm.8H, #90</code>	<code>Vd.8H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vcaddq_rot90_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FCADD Vd.4S, Vn.4S, Vm.4S, #90</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float64x2_t vcaddq_rot90_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FCADD Vd.2D, Vn.2D, Vm.2D, #90</code>	<code>Vd.2D -> result</code>	A64
<code>float16x4_t vcadd_rot270_f16(float16x4_t a, float16x4_t b)</code>	<code>a -> Vn.4H b -> Vm.4H</code>	<code>FCADD Vd.4H, Vn.4H, Vm.4H, #270</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float32x2_t vcadd_rot270_f32(float32x2_t a, float32x2_t b)</code>	<code>a -> Vn.2S b -> Vm.2S</code>	<code>FCADD Vd.2S, Vn.2S, Vm.2S, #270</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float16x8_t vcaddq_rot270_f16(float16x8_t a, float16x8_t b)</code>	<code>a -> Vn.8H b -> Vm.8H</code>	<code>FCADD Vd.8H, Vn.8H, Vm.8H, #270</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float32x4_t vcaddq_rot270_f32(float32x4_t a, float32x4_t b)</code>	<code>a -> Vn.4S b -> Vm.4S</code>	<code>FCADD Vd.4S, Vn.4S, Vm.4S, #270</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float64x2_t vcaddq_rot270_f64(float64x2_t a, float64x2_t b)</code>	<code>a -> Vn.2D b -> Vm.2D</code>	<code>FCADD Vd.2D, Vn.2D, Vm.2D, #270</code>	<code>Vd.2D -> result</code>	A64

2.11.1.2 Complex multiply-accumulate

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vcmla_f16(float16x4_t r, float16x4_t a, float16x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4H b -> Vm.4H</code>	<code>FCMLA Vd.4H, Vn.4H, Vm.4H, #0</code>	<code>Vd.4H -> result</code>	A32/A64
<code>float32x2_t vcmla_f32(float32x2_t r, float32x2_t a, float32x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2S b -> Vm.2S</code>	<code>FCMLA Vd.2S, Vn.2S, Vm.2S, #0</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float16x8_t vcmlaq_f16(float16x8_t r, float16x8_t a, float16x8_t b)</code>	<code>r -> Vd.8H a -> Vn.8H b -> Vm.8H</code>	<code>FCMLA Vd.8H, Vn.8H, Vm.8H, #0</code>	<code>Vd.8H -> result</code>	A32/A64
<code>float32x4_t vcmlaq_f32(float32x4_t r, float32x4_t a, float32x4_t b)</code>	<code>r -> Vd.4S a -> Vn.4S b -> Vm.4S</code>	<code>FCMLA Vd.4S, Vn.4S, Vm.4S, #0</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float64x2_t vcmlaq_f64(float64x2_t r, float64x2_t a, float64x2_t b)</code>	<code>r -> Vd.2D a -> Vn.2D b -> Vm.2D</code>	<code>FCMLA Vd.2D, Vn.2D, Vm.2D, #0</code>	<code>Vd.2D -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vcmla_rot90_f16(float16x4_t r, float16x4_t a, float16x4_t b)	r -> Vd.4H a -> Vn.4H b -> Vm.4H	FCMLA Vd.4H, Vn.4H, Vm.4H, #90	Vd.4H -> result	A32/A64
float32x2_t vcmla_rot90_f32(float32x2_t r, float32x2_t a, float32x2_t b)	r -> Vd.2S a -> Vn.2S b -> Vm.2S	FCMLA Vd.2S, Vn.2S, Vm.2S, #90	Vd.2S -> result	A32/A64
float16x8_t vcmlaq_rot90_f16(float16x8_t r, float16x8_t a, float16x8_t b)	r -> Vd.8H a -> Vn.8H b -> Vm.8H	FCMLA Vd.8H, Vn.8H, Vm.8H, #90	Vd.8H -> result	A32/A64
float32x4_t vcmlaq_rot90_f32(float32x4_t r, float32x4_t a, float32x4_t b)	r -> Vd.4S a -> Vn.4S b -> Vm.4S	FCMLA Vd.4S, Vn.4S, Vm.4S, #90	Vd.4S -> result	A32/A64
float64x2_t vcmlaq_rot90_f64(float64x2_t r, float64x2_t a, float64x2_t b)	r -> Vd.2D a -> Vn.2D b -> Vm.2D	FCMLA Vd.2D, Vn.2D, Vm.2D, #90	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x4_t vcmla_rot180_f16(float16x4_t r, float16x4_t a, float16x4_t b)</code>	<code>r -> Vd.4H a -> Vn.4H b -> Vm.4H</code>	FCMLA Vd.4H, Vn.4H, Vm.4H, #180	Vd.4H -> result	A32/A64
<code>float32x2_t vcmla_rot180_f32(float32x2_t r, float32x2_t a, float32x2_t b)</code>	<code>r -> Vd.2S a -> Vn.2S b -> Vm.2S</code>	FCMLA Vd.2S, Vn.2S, Vm.2S, #180	Vd.2S -> result	A32/A64
<code>float16x8_t vcmlaq_rot180_f16(float16x8_t r, float16x8_t a, float16x8_t b)</code>	<code>r -> Vd.8H a -> Vn.8H b -> Vm.8H</code>	FCMLA Vd.8H, Vn.8H, Vm.8H, #180	Vd.8H -> result	A32/A64
<code>float32x4_t vcmlaq_rot180_f32(float32x4_t r, float32x4_t a, float32x4_t b)</code>	<code>r -> Vd.4S a -> Vn.4S b -> Vm.4S</code>	FCMLA Vd.4S, Vn.4S, Vm.4S, #180	Vd.4S -> result	A32/A64
<code>float64x2_t vcmlaq_rot180_f64(float64x2_t r, float64x2_t a, float64x2_t b)</code>	<code>r -> Vd.2D a -> Vn.2D b -> Vm.2D</code>	FCMLA Vd.2D, Vn.2D, Vm.2D, #180	Vd.2D -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vcmla_rot270_f16(float16x4_t r, float16x4_t a, float16x4_t b)	r -> Vd.4H a -> Vn.4H b -> Vm.4H	FCMLA Vd.4H, Vn.4H, Vm.4H, #270	Vd.4H -> result	A32/A64
float32x2_t vcmla_rot270_f32(float32x2_t r, float32x2_t a, float32x2_t b)	r -> Vd.2S a -> Vn.2S b -> Vm.2S	FCMLA Vd.2S, Vn.2S, Vm.2S, #270	Vd.2S -> result	A32/A64
float16x8_t vcmlaq_rot270_f16(float16x8_t r, float16x8_t a, float16x8_t b)	r -> Vd.8H a -> Vn.8H b -> Vm.8H	FCMLA Vd.8H, Vn.8H, Vm.8H, #270	Vd.8H -> result	A32/A64
float32x4_t vcmlaq_rot270_f32(float32x4_t r, float32x4_t a, float32x4_t b)	r -> Vd.4S a -> Vn.4S b -> Vm.4S	FCMLA Vd.4S, Vn.4S, Vm.4S, #270	Vd.4S -> result	A32/A64
float64x2_t vcmlaq_rot270_f64(float64x2_t r, float64x2_t a, float64x2_t b)	r -> Vd.2D a -> Vn.2D b -> Vm.2D	FCMLA Vd.2D, Vn.2D, Vm.2D, #270	Vd.2D -> result	A64

2.11.1.3 Complex multiply-accumulate by scalar

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vcmla_lane_f16(float16x4_t r, float16x4_t a, float16x4_t b, const int lane)	r -> Vd.4H a -> Vn.4H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.4H, Vn.4H, Vm.H[lane], #0	Vd.4H -> result	A32/A64
float32x2_t vcmla_lane_f32(float32x2_t r, float32x2_t a, float32x2_t b, const int lane)	r -> Vd.2S a -> Vn.2S b -> Vm.S 0 <= lane <= 0	FCMLA Vd.2S, Vn.2S, Vm.2S, #0	Vd.2S -> result	A32/A64
float16x4_t vcmla_laneq_f16(float16x4_t r, float16x4_t a, float16x8_t b, const int lane)	r -> Vd.4H a -> Vn.4H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.4H, Vn.4H, Vm.H[lane], #0	Vd.4H -> result	A32/A64
float16x4_t vcmla_laneq_f16(float16x4_t r, float16x4_t a, float16x8_t b, const int lane)	r -> Vd.4H a -> Vn.4H b -> Vm.H 2 <= lane <= 3	DUP Dm, Vm.D[1] FCMLA Vd.4H, Vn.4H, Vm.H[lane % 2], #0	Vd.4H -> result	A32/A64
float32x2_t vcmla_laneq_f32(float32x2_t r, float32x2_t a, float32x4_t b, const int lane)	r -> Vd.2S a -> Vn.2S b -> Vm.S 0 <= lane <= 1	DUP Dm, Vm.D[1] FCMLA Vd.2S, Vn.2S, Vm.2S, #0	Vd.2S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x8_t vcmlaq_lane_f16(float16x8_t r, float16x8_t a, float16x4_t b, const int lane)	r -> Vd.8H a -> Vn.8H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.8H, Vn.8H, Vm.H[lane], #0	Vd.8H -> result	A32/A64
float32x4_t vcmlaq_lane_f32(float32x4_t r, float32x4_t a, float32x2_t b, const int lane)	r -> Vd.4S a -> Vn.4S b -> Vm.S 0 <= lane <= 0	FCMLA Vd.4S, Vn.4S, Vm.S[lane], #0	Vd.4S -> result	A32/A64
float16x8_t vcmlaq_laneq_f16(float16x8_t r, float16x8_t a, float16x8_t b, const int lane)	r -> Vd.8H a -> Vn.8H b -> Vm.H 0 <= lane <= 3	FCMLA Vd.8H, Vn.8H, Vm.H[lane], #0	Vd.8H -> result	A32/A64
float32x4_t vcmlaq_laneq_f32(float32x4_t r, float32x4_t a, float32x4_t b, const int lane)	r -> Vd.4S a -> Vn.4S b -> Vm.S 0 <= lane <= 1	FCMLA Vd.4S, Vn.4S, Vm.S[lane], #0	Vd.4S -> result	A32/A64
float16x4_t vcmla_rot90_lane_f16(float16x4_t r, float16x4_t a, float16x4_t b, const int lane)	r -> Vd.4H a -> Vn.4H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.4H, Vn.4H, Vm.H[lane], #90	Vd.4H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vcmla_rot90_lane_f32(float32x2_t r, float32x2_t a, float32x2_t b, const int lane)	r -> Vd.2S a -> Vn.2S b -> Vm.S 0 <= lane <= 0	FCMLA Vd.2S, Vn.2S, Vm.2S, #90	Vd.2S -> result	A32/A64
float16x4_t vcmla_rot90_laneq_f16(float16x4_t r, float16x4_t a, float16x8_t b, const int lane)	r -> Vd.4H a -> Vn.4H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.4H, Vn.4H, Vm.H[lane], #90	Vd.4H -> result	A32/A64
float16x4_t vcmla_rot90_laneq_f16(float16x4_t r, float16x4_t a, float16x8_t b, const int lane)	r -> Vd.4H a -> Vn.4H b -> Vm.H 2 <= lane <= 3	DUP Dm, Vm.D[1] FCMLA Vd.4H, Vn.4H, Vm.H[lane % 2], #90	Vd.4H -> result	A32/A64
float32x2_t vcmla_rot90_laneq_f32(float32x2_t r, float32x2_t a, float32x4_t b, const int lane)	r -> Vd.2S a -> Vn.2S b -> Vm.S 0 <= lane <= 1	DUP Dm, Vm.D[1] FCMLA Vd.2S, Vn.2S, Vm.2S, #90	Vd.2S -> result	A32/A64
float16x8_t vcmlaq_rot90_lane_f16(float16x8_t r, float16x8_t a, float16x4_t b, const int lane)	r -> Vd.8H a -> Vn.8H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.8H, Vn.8H, Vm.H[lane], #90	Vd.8H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vcmlaq_rot90_lane_f32(r -> Vd.4S float32x4_t r, float32x4_t a, float32x2_t b, const int lane)	a -> Vn.4S b -> Vm.S 0 <= lane <= 0	FCMLA Vd.4S, Vn.4S, Vm.S[lane], #90	Vd.4S -> result	A32/A64
float16x8_t vcmlaq_rot90_laneq_f16(r -> Vd.8H float16x8_t r, float16x8_t a, float16x8_t b, const int lane)	a -> Vn.8H b -> Vm.H 0 <= lane <= 3	FCMLA Vd.8H, Vn.8H, Vm.H[lane], #90	Vd.8H -> result	A32/A64
float32x4_t vcmlaq_rot90_laneq_f32(r -> Vd.4S float32x4_t r, float32x4_t a, float32x4_t b, const int lane)	a -> Vn.4S b -> Vm.S 0 <= lane <= 1	FCMLA Vd.4S, Vn.4S, Vm.S[lane], #90	Vd.4S -> result	A32/A64
float16x4_t vcmla_rot180_lane_f16(r -> Vd.4H float16x4_t r, float16x4_t a, float16x4_t b, const int lane)	a -> Vn.4H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.4H, Vn.4H, Vm.H[lane], #180	Vd.4H -> result	A32/A64
float32x2_t vcmla_rot180_lane_f32(r -> Vd.2S float32x2_t r, float32x2_t a, float32x2_t b, const int lane)	a -> Vn.2S b -> Vm.S 0 <= lane <= 0	FCMLA Vd.2S, Vn.2S, Vm.2S, #180	Vd.2S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vcmla_rot180_laneq_f16(r -> Vd.4H float16x4_t r, float16x4_t a, float16x8_t b, const int lane)	a -> Vn.4H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.4H, Vn.4H, Vm.H[lane], #180	Vd.4H -> result	A32/A64
float16x4_t vcmla_rot180_laneq_f16(r -> Vd.4H float16x4_t r, float16x4_t a, float16x8_t b, const int lane)	a -> Vn.4H b -> Vm.H 2 <= lane <= 3	DUP Dm, Vm.D[1] FCMLA Vd.4H, Vn.4H, Vm.H[lane % 2], #180	Vd.4H -> result	A32/A64
float32x2_t vcmla_rot180_laneq_f32(r -> Vd.2S float32x2_t r, float32x2_t a, float32x4_t b, const int lane)	a -> Vn.2S b -> Vm.S 0 <= lane <= 1	DUP Dm, Vm.D[1] FCMLA Vd.2S, Vn.2S, Vm.2S, #180	Vd.2S -> result	A32/A64
float16x8_t vcmlaq_rot180_lane_f16(r -> Vd.8H float16x8_t r, float16x8_t a, float16x4_t b, const int lane)	a -> Vn.8H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.8H, Vn.8H, Vm.H[lane], #180	Vd.8H -> result	A32/A64
float32x4_t vcmlaq_rot180_lane_f32(r -> Vd.4S float32x4_t r, float32x4_t a, float32x2_t b, const int lane)	a -> Vn.4S b -> Vm.S 0 <= lane <= 0	FCMLA Vd.4S, Vn.4S, Vm.S[lane], #180	Vd.4S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<pre>float16x8_t vcmlaq_rot180_laneq_f16(float16x8_t r, float16x8_t a, float16x8_t b, const int lane)</pre>	<pre>r -> Vd.8H a -> Vn.8H b -> Vm.H 0 <= lane <= 3</pre>	FCMLA Vd.8H, Vn.8H, Vm.H[lane], #180	Vd.8H -> result	A32/A64
<pre>float32x4_t vcmlaq_rot180_laneq_f32(float32x4_t r, float32x4_t a, float32x4_t b, const int lane)</pre>	<pre>r -> Vd.4S a -> Vn.4S b -> Vm.S 0 <= lane <= 1</pre>	FCMLA Vd.4S, Vn.4S, Vm.S[lane], #180	Vd.4S -> result	A32/A64
<pre>float16x4_t vcmla_rot270_lane_f16(float16x4_t r, float16x4_t a, float16x4_t b, const int lane)</pre>	<pre>r -> Vd.4H a -> Vn.4H b -> Vm.H 0 <= lane <= 1</pre>	FCMLA Vd.4H, Vn.4H, Vm.H[lane], #270	Vd.4H -> result	A32/A64
<pre>float32x2_t vcmla_rot270_lane_f32(float32x2_t r, float32x2_t a, float32x2_t b, const int lane)</pre>	<pre>r -> Vd.2S a -> Vn.2S b -> Vm.S 0 <= lane <= 0</pre>	FCMLA Vd.2S, Vn.2S, Vm.2S, #270	Vd.2S -> result	A32/A64
<pre>float16x4_t vcmla_rot270_laneq_f16(float16x4_t r, float16x4_t a, float16x8_t b, const int lane)</pre>	<pre>r -> Vd.4H a -> Vn.4H b -> Vm.H 0 <= lane <= 1</pre>	FCMLA Vd.4H, Vn.4H, Vm.H[lane], #270	Vd.4H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vcmla_rot270_laneq_f16(r -> Vd.4H float16x4_t r, float16x4_t a, float16x8_t b, const int lane)	a -> Vn.4H b -> Vm.H 2 <= lane <= 3	DUP Dm, Vm.D[1] FCMLA Vd.4H, Vn.4H, Vm.H[lane % 2], #270	Vd.4H -> result	A32/A64
float32x2_t vcmla_rot270_laneq_f32(r -> Vd.2S float32x2_t r, float32x2_t a, float32x4_t b, const int lane)	a -> Vn.2S b -> Vm.S 0 <= lane <= 1	DUP Dm, Vm.D[1] FCMLA Vd.2S, Vn.2S, Vm.2S, #270	Vd.2S -> result	A32/A64
float16x8_t vcmlaq_rot270_lane_f16(r -> Vd.8H float16x8_t r, float16x8_t a, float16x4_t b, const int lane)	a -> Vn.8H b -> Vm.H 0 <= lane <= 1	FCMLA Vd.8H, Vn.8H, Vm.H[lane], #270	Vd.8H -> result	A32/A64
float32x4_t vcmlaq_rot270_lane_f32(r -> Vd.4S float32x4_t r, float32x4_t a, float32x2_t b, const int lane)	a -> Vn.4S b -> Vm.S 0 <= lane <= 0	FCMLA Vd.4S, Vn.4S, Vm.S[lane], #270	Vd.4S -> result	A32/A64
float16x8_t vcmlaq_rot270_laneq_f16(r -> Vd.8H float16x8_t r, float16x8_t a, float16x8_t b, const int lane)	a -> Vn.8H b -> Vm.H 0 <= lane <= 3	FCMLA Vd.8H, Vn.8H, Vm.H[lane], #270	Vd.8H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vcmlaq_rot270_laneq_f32(float32x4_t r, float32x4_t a, float32x4_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.4S b -> Vm.S 0 <= lane <= 1</code>	<code>FCMLA Vd.4S, Vn.4S, Vm.S[lane], #270</code>	<code>Vd.4S -> result</code>	A32/A64

2.12 Floating-point rounding intrinsics from Armv8.5-A

2.12.1 Vector arithmetic

2.12.1.1 Rounding

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vrnd32z_f32(float32x2_t a)</code>	<code>a -> Vn</code>	<code>FRINT32Z Vd.2S,Vn.2S</code>	<code>Vd -> result</code>	A64
<code>float32x4_t vrnd32zq_f32(float32x4_t a)</code>	<code>a -> Vn</code>	<code>FRINT32Z Vd.4S,Vn.4S</code>	<code>Vd -> result</code>	A64
<code>float64x1_t vrnd32z_f64(float64x1_t a)</code>	<code>a -> Dn</code>	<code>FRINT32Z Dd,Dn</code>	<code>Dd -> result</code>	A64
<code>float64x2_t vrnd32zq_f64(float64x2_t a)</code>	<code>a -> Vn</code>	<code>FRINT32Z Vd.2D,Vn.2D</code>	<code>Vd -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vrnd64z_f32(float32x2_t a)	a -> Vn	FRINT64Z Vd.2S,Vn.2S	Vd -> result	A64
float32x4_t vrnd64zq_f32(float32x4_t a)	a -> Vn	FRINT64Z Vd.4S,Vn.4S	Vd -> result	A64
float64x1_t vrnd64z_f64(float64x1_t a)	a -> Dn	FRINT64Z Dd,Dn	Dd -> result	A64
float64x2_t vrnd64zq_f64(float64x2_t a)	a -> Vn	FRINT64Z Vd.2D,Vn.2D	Vd -> result	A64
float32x2_t vrnd32x_f32(float32x2_t a)	a -> Vn	FRINT32X Vd.2S,Vn.2S	Vd -> result	A64
float32x4_t vrnd32xq_f32(float32x4_t a)	a -> Vn	FRINT32X Vd.4S,Vn.4S	Vd -> result	A64
float64x1_t vrnd32x_f64(float64x1_t a)	a -> Dn	FRINT32X Dd,Dn	Dd -> result	A64
float64x2_t vrnd32xq_f64(float64x2_t a)	a -> Vn	FRINT32X Vd.2D,Vn.2D	Vd -> result	A64
float32x2_t vrnd64x_f32(float32x2_t a)	a -> Vn	FRINT64X Vd.2S,Vn.2S	Vd -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vrnd64xq_f32(float32x4_t a)	a -> Vn	FRINT64X Vd.4S,Vn.4S	Vd -> result	A64
float64x1_t vrnd64x_f64(float64x1_t a)	a -> Dn	FRINT64X Dd,Dn	Dd -> result	A64
float64x2_t vrnd64xq_f64(float64x2_t a)	a -> Vn	FRINT64X Vd.2D,Vn.2D	Vd -> result	A64

2.13 Matrix multiplication intrinsics from Armv8.6-A

2.13.1 Vector arithmetic

2.13.1.1 Matrix multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
int32x4_t vmmlaq_s32(int32x4_t r, int8x16_t a, int8x16_t b)	r -> Vd.4S a -> Vn.16B b -> Vm.16B	SMMLA Vd.4S,Vn.16B,Vm.16B	Vd.4S -> result	A32/A64
uint32x4_t vmmlaq_u32(uint32x4_t r, uint8x16_t a, uint8x16_t b)	r -> Vd.4S a -> Vn.16B b -> Vm.16B	UMMLA Vd.4S,Vn.16B,Vm.16B	Vd.4S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vusmmlaq_s32(int32x4_t r, uint8x16_t a, int8x16_t b)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.16B</code>	USMMLA Vd.4S,Vn.16B,Vm.16B	Vd.4S -> result	A32/A64

2.13.1.2 Dot product

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vusdot_s32(int32x2_t r, uint8x8_t a, int8x8_t b)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.8B</code>	USDOT Vd.2S,Vn.8B,Vm.8B	Vd.2S -> result	A32/A64
<code>int32x2_t vusdot_lane_s32(int32x2_t r, uint8x8_t a, int8x8_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 1</code>	USDOT Vd.2S,Vn.8B,Vm.4B[lane]	Vd.2S -> result	A32/A64
<code>int32x2_t vsudot_lane_s32(int32x2_t r, int8x8_t a, uint8x8_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 1</code>	SUDOT Vd.2S,Vn.8B,Vm.4B[lane]	Vd.2S -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vusdot_laneq_s32(int32x2_t r, uint8x8_t a, int8x16_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 3</code>	<code>USDOT Vd.2S,Vn.8B,Vm.4B[lane]</code>	<code>Vd.2S -> result</code>	A32/A64
<code>int32x2_t vsudot_laneq_s32(int32x2_t r, int8x8_t a, uint8x16_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 3</code>	<code>SUDOT Vd.2S,Vn.8B,Vm.4B[lane]</code>	<code>Vd.2S -> result</code>	A32/A64
<code>int32x4_t vusdotq_s32(int32x4_t r, uint8x16_t a, int8x16_t b)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.16B</code>	<code>USDOT Vd.4S,Vn.16B,Vm.16B</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x4_t vusdotq_lane_s32(int32x4_t r, uint8x16_t a, int8x8_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.4B 0 <= lane <= 1</code>	<code>USDOT Vd.4S,Vn.16B,Vm.4B[lane]</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x4_t vsudotq_lane_s32(int32x4_t r, int8x16_t a, uint8x8_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.8B b -> Vm.4B 0 <= lane <= 1</code>	<code>SUDOT Vd.4S,Vn.16B,Vm.4B[lane]</code>	<code>Vd.4S -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x4_t vusdotq_laneq_s32(int32x4_t r, uint8x16_t a, int8x16_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.16B b -> Vm.4B 0 <= lane <= 3</code>	<code>USDOT Vd.4S,Vn.16B,Vm.4B[lane]</code>	<code>Vd.4S -> result</code>	A32/A64
<code>int32x4_t vsudotq_laneq_s32(int32x4_t r, int8x16_t a, uint8x16_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.8B b -> Vm.4B 0 <= lane <= 3</code>	<code>SUDOT Vd.4S,Vn.16B,Vm.4B[lane]</code>	<code>Vd.4S -> result</code>	A32/A64

2.14 Bfloat16 intrinsics Requires the +bf16 architecture extension.

2.14.1 Vector manipulation

2.14.1.1 Create vector

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4_t vcreate_bf16(uint64_t a)</code>	<code>a -> Xn</code>	<code>INS Vd.D[0],Xn</code>	<code>Vd.4H -> result</code>	A32/A64

2.14.1.2 Set all lanes to the same value

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4_t vdup_n_bf16(bfloat16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.4H,rn</code>	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x8_t vdupq_n_bf16(bfloat16_t value)</code>	<code>value -> rn</code>	<code>DUP Vd.8H,rn</code>	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x4_t vdup_lane_bf16(bfloat16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x8_t vdupq_lane_bf16(bfloat16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x4_t vdup_laneq_bf16(bfloat16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.4H,Vn.H[lane]</code>	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x8_t vdupq_laneq_bf16(bfloat16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Vd.8H,Vn.H[lane]</code>	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16_t vduph_lane_bf16(bfloat16x4_t vec, const int lane)</code>	<code>vec -> Vn.4H 0 <= lane <= 3</code>	<code>DUP Hd,Vn.H[lane]</code>	<code>Hd -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16_t vduph_laneq_bf16(bfloat16x8_t vec, const int lane)</code>	<code>vec -> Vn.8H 0 <= lane <= 7</code>	<code>DUP Hd,Vn.H[lane]</code>	<code>Hd -> result</code>	A32/A64

2.14.1.3 Combine vectors

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8_t vcombine_bf16(bfloat16x4_t low, bfloat16x4_t high)</code>	<code>low -> Vn.4H high -> Vm.4H</code>	<code>DUP Vd.1D,Vn.D[0] INS Vd.D[1],Vm.D[0]</code>	<code>Vd.8H -> result</code>	A32/A64

2.14.1.4 Split vectors

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4_t vget_high_bf16(bfloat16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>DUP Vd.1D,Vn.D[1]</code>	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vget_low_bf16(bfloat16x8_t a)</code>	<code>a -> Vn.8H</code>	<code>DUP Vd.1D,Vn.D[0]</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16_t vget_lane_bf16(bfloat16x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ $v \rightarrow Vn.4H$	DUP Hd,Vn.H[lane]	Hd -> result	A32/A64
<code>bfloat16_t vgetq_lane_bf16(bfloat16x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ $v \rightarrow Vn.8H$	DUP Hd,Vn.H[lane]	Hd -> result	A32/A64

2.14.1.5 Set vector lane

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4_t vset_lane_bf16(bfloat16_t a, bfloat16x4_t v, const int lane)</code>	$0 \leq \text{lane} \leq 3$ $a \rightarrow VnH$ $v \rightarrow Vd.4H$	INS Vd.H[lane],Vn.H[0]	Vd.4H -> result	A32/A64
<code>bfloat16x8_t vsetq_lane_bf16(bfloat16_t a, bfloat16x8_t v, const int lane)</code>	$0 \leq \text{lane} \leq 7$ $a \rightarrow VnH$ $v \rightarrow Vd.8H$	INS Vd.H[lane],Vn.H[0]	Vd.8H -> result	A32/A64

2.14.1.6 Copy vector lane

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4_t vcopy_lane_bf16(bfloat16x4_t a, const int lane1, bfloat16x4_t b, const int lane2)</code>	<code>a -> Vd.4H 0 <= lane1 <= 3 b -> Vn.4H 0 <= lane2 <= 3</code>	<code>INS Vd.H[lane1],Vn.H[lane2]</code>	<code>Vd.4H -> result</code>	A64
<code>bfloat16x8_t vcopyq_lane_bf16(bfloat16x8_t a, const int lane1, bfloat16x4_t b, const int lane2)</code>	<code>a -> Vd.8H 0 <= lane1 <= 7 b -> Vn.4H 0 <= lane2 <= 3</code>	<code>INS Vd.H[lane1],Vn.H[lane2]</code>	<code>Vd.8H -> result</code>	A64
<code>bfloat16x4_t vcopy_laneq_bf16(bfloat16x4_t a, const int lane1, bfloat16x8_t b, const int lane2)</code>	<code>a -> Vd.4H 0 <= lane1 <= 3 b -> Vn.8H 0 <= lane2 <= 7</code>	<code>INS Vd.H[lane1],Vn.H[lane2]</code>	<code>Vd.4H -> result</code>	A64
<code>bfloat16x8_t vcopyq_laneq_bf16(bfloat16x8_t a, const int lane1, bfloat16x8_t b, const int lane2)</code>	<code>a -> Vd.8H 0 <= lane1 <= 7 b -> Vn.8H 0 <= lane2 <= 7</code>	<code>INS Vd.H[lane1],Vn.H[lane2]</code>	<code>Vd.8H -> result</code>	A64

2.14.2 Load

2.14.2.1 Stride

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4_t vld1_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H},[Xn]</code>	<code>Vt.4H -> result</code>	A32/A64
<code>bfloat16x8_t vld1q_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H},[Xn]</code>	<code>Vt.8H -> result</code>	A32/A64
<code>bfloat16x4_t vld1_lane_bf16(bfloat16_t const *ptr, bfloat16x4_t src, const int lane)</code>	<code>ptr -> Xn</code> <code>src -> Vt.4H</code> <code>0 <= lane <= 3</code>	<code>LD1 {Vt.H}[lane],[Xn]</code>	<code>Vt.4H -> result</code>	A32/A64
<code>bfloat16x8_t vld1q_lane_bf16(bfloat16_t const *ptr, bfloat16x8_t src, const int lane)</code>	<code>ptr -> Xn</code> <code>src -> Vt.8H</code> <code>0 <= lane <= 7</code>	<code>LD1 {Vt.H}[lane],[Xn]</code>	<code>Vt.8H -> result</code>	A32/A64
<code>bfloat16x4_t vld1_dup_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.4H},[Xn]</code>	<code>Vt.4H -> result</code>	A32/A64
<code>bfloat16x8_t vld1q_dup_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1R {Vt.8H},[Xn]</code>	<code>Vt.8H -> result</code>	A32/A64
<code>bfloat16x4x2_t vld2_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.4H - Vt.2.4H},[Xn]</code>	<code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8x2_t vld2q_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2 {Vt.8H - Vt2.8H}, [Xn]</code>	Vt2.8H -> result.val[2 32/A64] Vt.8H -> result.val[0]	
<code>bfloat16x4x3_t vld3_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2 32/A64] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	
<code>bfloat16x8x3_t vld3q_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3 {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2 32/A64] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	
<code>bfloat16x4x4_t vld4_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[2 32/A64] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	
<code>bfloat16x8x4_t vld4q_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4 {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[2 32/A64] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4x2_t vld2_dup_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.4H - Vt2.4H}, [Xn]</code>	<code>Vt2.4H -> result.val[A32/A64]</code> <code>Vt.4H -> result.val[0]</code>	
<code>bfloat16x8x2_t vld2q_dup_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD2R {Vt.8H - Vt2.8H}, [Xn]</code>	<code>Vt2.8H -> result.val[A32/A64]</code> <code>Vt.8H -> result.val[0]</code>	
<code>bfloat16x4x3_t vld3_dup_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.4H - Vt3.4H}, [Xn]</code>	<code>Vt3.4H -> result.val[A32/A64]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	
<code>bfloat16x8x3_t vld3q_dup_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD3R {Vt.8H - Vt3.8H}, [Xn]</code>	<code>Vt3.8H -> result.val[A32/A64]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	
<code>bfloat16x4x4_t vld4_dup_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.4H - Vt4.4H}, [Xn]</code>	<code>Vt4.4H -> result.val[A32/A64]</code> <code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8x4_t vld4q_dup_bf16(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD4R {Vt.8H - Vt4.8H}, [Xn]</code>	<code>Vt4.8H -> result.val[3]</code> <code>Vt3.8H -> result.val[2]</code> <code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>bfloat16x4x2_t vld2_lane_bf16(bfloat16_t const *ptr, bfloat16x4x2_t src, const int lane)</code>	<code>ptr -> Xn</code> <code>src.val[1] -> Vt2.4H</code> <code>src.val[0] -> Vt.4H</code> <code>0 <= lane <= 3</code>	<code>LD2 {Vt.h - Vt2.h}[lane], [Xn]</code>	<code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64
<code>bfloat16x8x2_t vld2q_lane_bf16(bfloat16_t const *ptr, bfloat16x8x2_t src, const int lane)</code>	<code>ptr -> Xn</code> <code>src.val[1] -> Vt2.8H</code> <code>src.val[0] -> Vt.8H</code> <code>0 <= lane <= 7</code>	<code>LD2 {Vt.h - Vt2.h}[lane], [Xn]</code>	<code>Vt2.8H -> result.val[1]</code> <code>Vt.8H -> result.val[0]</code>	A32/A64
<code>bfloat16x4x3_t vld3_lane_bf16(bfloat16_t const *ptr, bfloat16x4x3_t src, const int lane)</code>	<code>ptr -> Xn</code> <code>src.val[2] -> Vt3.4H</code> <code>src.val[1] -> Vt2.4H</code> <code>src.val[0] -> Vt.4H</code> <code>0 <= lane <= 3</code>	<code>LD3 {Vt.h - Vt3.h}[lane], [Xn]</code>	<code>Vt3.4H -> result.val[2]</code> <code>Vt2.4H -> result.val[1]</code> <code>Vt.4H -> result.val[0]</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
bf float16x8x3_t vld3q_lane_bf16(bf float 16_t const *ptr, bf float 16x8x3_t src, const int lane)	ptr -> Xn src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7	LD3 {Vt.h - Vt3.h}[lane], [Xn]	Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
bf float16x4x4_t vld4_lane_bf16(bf float 16_t const *ptr, bf float 16x4x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.4H src.val[2] -> Vt3.4H src.val[1] -> Vt2.4H src.val[0] -> Vt.4H 0 <= lane <= 3	LD4 {Vt.h - Vt4.h}[lane], [Xn]	Vt4.4H -> result.val[3] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64
bf float16x8x4_t vld4q_lane_bf16(bf float 16_t const *ptr, bf float 16x8x4_t src, const int lane)	ptr -> Xn src.val[3] -> Vt4.8H src.val[2] -> Vt3.8H src.val[1] -> Vt2.8H src.val[0] -> Vt.8H 0 <= lane <= 7	LD4 {Vt.h - Vt4.h}[lane], [Xn]	Vt4.8H -> result.val[3] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	A32/A64
bf float16x4x2_t vld1_bf16_x2(bf float 16_t const *ptr)	ptr -> Xn	LD1 {Vt.4H - Vt2.4H}, [Xn]	Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8x2_t vld1q_bf16_x2(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt2.8H}, [Xn]</code>	Vt2.8H -> result.val[A32/A64] Vt.8H -> result.val[0]	
<code>bfloat16x4x3_t vld1_bf16_x3(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt3.4H}, [Xn]</code>	Vt3.4H -> result.val[2]/A32/A64 Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	
<code>bfloat16x8x3_t vld1q_bf16_x3(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt3.8H}, [Xn]</code>	Vt3.8H -> result.val[2]/A32/A64 Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	
<code>bfloat16x4x4_t vld1_bf16_x4(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.4H - Vt4.4H}, [Xn]</code>	Vt4.4H -> result.val[A32/A64] Vt3.4H -> result.val[2] Vt2.4H -> result.val[1] Vt.4H -> result.val[0]	
<code>bfloat16x8x4_t vld1q_bf16_x4(bfloat16_t const *ptr)</code>	<code>ptr -> Xn</code>	<code>LD1 {Vt.8H - Vt4.8H}, [Xn]</code>	Vt4.8H -> result.val[A32/A64] Vt3.8H -> result.val[2] Vt2.8H -> result.val[1] Vt.8H -> result.val[0]	

2.14.3 Store

2.14.3.1 Stride

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1_bf16(bfloat16_t *ptr, bfloat16x4_t val)</code>	<code>val -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H}, [Xn]		A32/A64
<code>void vst1q_bf16(bfloat16_t *ptr, bfloat16x8_t val)</code>	<code>val -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H}, [Xn]		A32/A64
<code>void vst1_lane_bf16(bfloat16_t *ptr, bfloat16x4_t val, const int lane)</code>	<code>val -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST1 {Vt.h}[lane], [Xn]		A32/A64
<code>void vst1q_lane_bf16(bfloat16_t *ptr, bfloat16x8_t val, const int lane)</code>	<code>val -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST1 {Vt.h}[lane], [Xn]		A32/A64
<code>void vst2_bf16(bfloat16_t *ptr, bfloat16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST2 {Vt.4H - Vt2.4H}, [Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst2q_bf16(bfloat16_t *ptr, bfloat16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST2 {Vt.8H - Vt2.8H}, [Xn]		A32/A64
<code>void vst3_bf16(bfloat16_t *ptr, bfloat16x4x3_t val)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST3 {Vt.4H - Vt3.4H}, [Xn]		A32/A64
<code>void vst3q_bf16(bfloat16_t *ptr, bfloat16x8x3_t val)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST3 {Vt.8H - Vt3.8H}, [Xn]		A32/A64
<code>void vst4_bf16(bfloat16_t *ptr, bfloat16x4x4_t val)</code>	<code>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST4 {Vt.4H - Vt4.4H}, [Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst4q_bf16(bfloat16_t *ptr, bfloat16x8x4_t val)</code>	<code>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST4 {Vt.8H - Vt4.8H},[Xn]		A32/A64
<code>void vst2_lane_bf16(bfloat16_t *ptr, bfloat16x4x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST2 {Vt.h - Vt2.h}[lane],[Xn]		A32/A64
<code>void vst2q_lane_bf16(bfloat16_t *ptr, bfloat16x8x2_t val, const int lane)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST2 {Vt.h - Vt2.h}[lane],[Xn]		A32/A64
<code>void vst3_lane_bf16(bfloat16_t *ptr, bfloat16x4x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST3 {Vt.h - Vt3.h}[lane],[Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst3q_lane_bf16(bfloat16_t *ptr, bfloat16x8x3_t val, const int lane)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST3 {Vt.h - Vt3.h}[lane],[Xn]		A32/A64
<code>void vst4_lane_bf16(bfloat16_t *ptr, bfloat16x4x4_t val, const int lane)</code>	<code>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn 0 <= lane <= 3</code>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		A32/A64
<code>void vst4q_lane_bf16(bfloat16_t *ptr, bfloat16x8x4_t val, const int lane)</code>	<code>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn 0 <= lane <= 7</code>	ST4 {Vt.h - Vt4.h}[lane],[Xn]		A32/A64
<code>void vst1_bf16_x2(bfloat16_t *ptr, bfloat16x4x2_t val)</code>	<code>val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt2.4H},[Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_bf16_x2(bfloat16_t *ptr, bfloat16x8x2_t val)</code>	<code>val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt2.8H},[Xn]		A32/A64
<code>void vst1_bf16_x3(bfloat16_t *ptr, bfloat16x4x3_t val)</code>	<code>val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt3.4H},[Xn]		A32/A64
<code>void vst1q_bf16_x3(bfloat16_t *ptr, bfloat16x8x3_t val)</code>	<code>val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	ST1 {Vt.8H - Vt3.8H},[Xn]		A32/A64
<code>void vst1_bf16_x4(bfloat16_t *ptr, bfloat16x4x4_t val)</code>	<code>val.val[3] -> Vt4.4H val.val[2] -> Vt3.4H val.val[1] -> Vt2.4H val.val[0] -> Vt.4H ptr -> Xn</code>	ST1 {Vt.4H - Vt4.4H},[Xn]		A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>void vst1q_bf16_x4(bfloat16_t *ptr, bfloat16x8x4_t val)</code>	<code>val.val[3] -> Vt4.8H val.val[2] -> Vt3.8H val.val[1] -> Vt2.8H val.val[0] -> Vt.8H ptr -> Xn</code>	<code>ST1 {Vt.8H - Vt4.8H}, [Xn]</code>		A32/A64

2.14.4 Data type conversion

2.14.4.1 Reinterpret casts

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4_t vreinterpret_bf16_s8(int8x8_t a)</code>	<code>a -> Vd.8B</code>	<code>NOP</code>	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_s16(int16x4_t a)</code>	<code>a -> Vd.4H</code>	<code>NOP</code>	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_s32(int32x2_t a)</code>	<code>a -> Vd.2S</code>	<code>NOP</code>	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_f32(float32x2_t a)</code>	<code>a -> Vd.2S</code>	<code>NOP</code>	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x4_t vreinterpret_bf16_u8(uint8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_u16(uint16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_u32(uint32x2_t a)</code>	<code>a -> Vd.2S</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_p8(poly8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_p16(poly16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_mf8(mfloat8x8_t a)</code>	<code>a -> Vd.8B</code>	NOP	<code>Vd.4H -> result</code>	A64
<code>bfloat16x4_t vreinterpret_bf16_u64(uint64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_s64(int64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_s8(int8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8_t vreinterpretq_bf16_s16(int16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_s32(int32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_f32(float32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_u8(uint8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_u16(uint16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_u32(uint32x4_t a)</code>	<code>a -> Vd.4S</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_p8(poly8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_p16(poly16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_mf8(mfloat8x16_t a)</code>	<code>a -> Vd.16B</code>	NOP	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8_t vreinterpretq_bf16_u64(uint64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_s64(int64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x4_t vreinterpret_bf16_f64(float64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A64
<code>bfloat16x8_t vreinterpretq_bf16_f64(float64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A64
<code>bfloat16x4_t vreinterpret_bf16_p64(poly64x1_t a)</code>	<code>a -> Vd.1D</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_p64(poly64x2_t a)</code>	<code>a -> Vd.2D</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>bfloat16x8_t vreinterpretq_bf16_p128(poly128_t a)</code>	<code>a -> Vd.1Q</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>int8x8_t vreinterpret_s8_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	A32/A64
<code>int16x4_t vreinterpret_s16_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int32x2_t vreinterpret_s32_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	A32/A64
<code>float32x2_t vreinterpret_f32_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	A32/A64
<code>uint8x8_t vreinterpret_u8_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	A32/A64
<code>uint16x4_t vreinterpret_u16_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>uint32x2_t vreinterpret_u32_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.2S -> result</code>	A32/A64
<code>poly8x8_t vreinterpret_p8_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	A32/A64
<code>poly16x4_t vreinterpret_p16_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.4H -> result</code>	A32/A64
<code>mfloat8x8_t vreinterpret_mf8_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.8B -> result</code>	A64
<code>uint64x1_t vreinterpret_u64_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>int64x1_t vreinterpret_s64_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>float64x1_t vreinterpret_f64_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A64
<code>poly64x1_t vreinterpret_p64_bf16(bfloat16x4_t a)</code>	<code>a -> Vd.4H</code>	NOP	<code>Vd.1D -> result</code>	A32/A64
<code>int8x16_t vreinterpretq_s8_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>int16x8_t vreinterpretq_s16_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>int32x4_t vreinterpretq_s32_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	A32/A64
<code>float32x4_t vreinterpretq_f32_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	A32/A64
<code>uint8x16_t vreinterpretq_u8_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>uint16x8_t vreinterpretq_u16_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>uint32x4_t vreinterpretq_u32_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.4S -> result</code>	A32/A64
<code>poly8x16_t vreinterpretq_p8_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	A32/A64
<code>poly16x8_t vreinterpretq_p16_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.8H -> result</code>	A32/A64
<code>mfloat8x16_t vreinterpretq_mf8_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.16B -> result</code>	A64
<code>uint64x2_t vreinterpretq_u64_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	A32/A64
<code>int64x2_t vreinterpretq_s64_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	A32/A64
<code>float64x2_t vreinterpretq_f64_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	A64
<code>poly64x2_t vreinterpretq_p64_bf16(bfloat16x8_t a)</code>	<code>a -> Vd.8H</code>	NOP	<code>Vd.2D -> result</code>	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
poly128_t vreinterpretq_p128_bf16(bfloat16x8_t a)	a -> Vd.8H	NOP	Vd.1Q -> result	A32/A64

2.14.4.2 Conversions

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vcvtf32_bf16(bfloat16x4_t a)	a -> Vn.8H	SHLL Vd.4S,Vn.8H, #16	Vd.4S -> result	A32/A64
float32x4_t vcvttq_low_f32_bf16(bfloat16x8_t a)	a -> Vn.8H	SHLL Vd.4S,Vn.8H, #16	Vd.4S -> result	A32/A64
float32x4_t vcvttq_high_f32_bf16(bfloat16x8_t a)	a -> Vn.8H	SHLL2 Vd.4S,Vn.8H, #16	Vd.4S -> result	A32/A64
bfloat16x4_t vcvtfb16_f32(float32x4_t a)	a -> Vn.4S	BFCVTN Vd.4H,Vn.4S	Vd.4H -> result	A32/A64
bfloat16x8_t vcvttq_low_bf16_f32(float32x4_t a)	a -> Vn.4S	BFCVTN Vd.4H,Vn.4S	Vd.8H -> result	A32/A64
bfloat16x8_t vcvttq_high_bf16_f32(bfloat16x8_t inactive, float32x4_t a)	inactive -> Vd.8H a -> Vn.4S	BFCVTN2 Vd.8H,Vn.4S	Vd.8H -> result	A32/A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16_t vcvth_bf16_f32(float32_t a)</code>	<code>a -> Sn</code>	<code>BFCVT Hd,Sn</code>	<code>Hd -> result</code>	A32/A64
<code>float32_t vcvtah_f32_bf16(bfloat16_t a)</code>	<code>a -> Hn</code>	<code>SHL Dd,Dn, #16</code>	<code>Sd -> result</code>	A32/A64

2.14.5 Vector arithmetic

2.14.5.1 Dot product

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x2_t vbfdot_f32(float32x2_t r, bfloat16x4_t a, bfloat16x4_t b)</code>	<code>r -> Vd.2S</code> <code>a -> Vn.4H</code> <code>b -> Vm.4H</code>	<code>BFDOT Vd.2S,Vn.4H,Vm.4H</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float32x4_t vbfdotq_f32(float32x4_t r, bfloat16x8_t a, bfloat16x8_t b)</code>	<code>r -> Vd.4S</code> <code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	<code>BFDOT Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float32x2_t vbfdot_lane_f32(float32x2_t r, bfloat16x4_t a, bfloat16x4_t b, const int lane)</code>	<code>r -> Vd.2S</code> <code>a -> Vn.4H</code> <code>b -> Vm.4H</code> <code>0 <= lane <= 1</code>	<code>BFDOT Vd.2S,Vn.4H,Vm.2H[lane]</code>	<code>Vd.2S -> result</code>	A32/A64

Copyright: see section [Copyright](#).

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vbfdotq_laneq_f32(float32x4_t r, bfloat16x8_t a, bfloat16x8_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.8H b -> Vm.8H 0 <= lane <= 3</code>	<code>BFDOT Vd.4S,Vn.8H,Vm.2H[lane]</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float32x2_t vbfdot_laneq_f32(float32x2_t r, bfloat16x4_t a, bfloat16x8_t b, const int lane)</code>	<code>r -> Vd.2S a -> Vn.4H b -> Vm.8H 0 <= lane <= 3</code>	<code>BFDOT Vd.2S,Vn.4H,Vm.2H[lane]</code>	<code>Vd.2S -> result</code>	A32/A64
<code>float32x4_t vbfdotq_lane_f32(float32x4_t r, bfloat16x8_t a, bfloat16x4_t b, const int lane)</code>	<code>r -> Vd.4S a -> Vn.8H b -> Vm.4H 0 <= lane <= 1</code>	<code>BFDOT Vd.4S,Vn.8H,Vm.2H[lane]</code>	<code>Vd.4S -> result</code>	A32/A64

2.14.5.2 Matrix multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vbfmmlaq_f32(float32x4_t r, bfloat16x8_t a, bfloat16x8_t b)</code>	<code>r -> Vd.4S</code> <code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	<code>BFMMLA Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A32/A64

2.14.5.3 Multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vbfmlalbq_f32(float32x4_t r, bfloat16x8_t a, bfloat16x8_t b)</code>	<code>r -> Vd.4S</code> <code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	<code>BFMLALB Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A32/A64
<code>float32x4_t vbfmlaltq_f32(float32x4_t r, bfloat16x8_t a, bfloat16x8_t b)</code>	<code>r -> Vd.4S</code> <code>a -> Vn.8H</code> <code>b -> Vm.8H</code>	<code>BFMLALT Vd.4S,Vn.8H,Vm.8H</code>	<code>Vd.4S -> result</code>	A32/A64

2.14.5.3.1 Multiply-accumulate

2.14.6 Scalar arithmetic

2.14.6.1 Vector multiply-accumulate by scalar

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vbfmlalbq_lane_f32(float32x4_t r, bfloat16x8_t a, bfloat16x4_t b, const int lane)	r -> Vd.4S a -> Vn.8H b -> Vm.4H 0 <= lane <= 3	BFMLALB Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A32/A64
float32x4_t vbfmlalbq_laneq_f32(float32x4_t r, bfloat16x8_t a, bfloat16x8_t b, const int lane)	r -> Vd.4S a -> Vn.8H b -> Vm.8H 0 <= lane <= 7	BFMLALB Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A32/A64
float32x4_t vbfmlaltq_lane_f32(float32x4_t r, bfloat16x8_t a, bfloat16x4_t b, const int lane)	r -> Vd.4S a -> Vn.8H b -> Vm.4H 0 <= lane <= 3	BFMLALT Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A32/A64
float32x4_t vbfmlaltq_laneq_f32(float32x4_t r, bfloat16x8_t a, bfloat16x8_t b, const int lane)	r -> Vd.4S a -> Vn.8H b -> Vm.8H 0 <= lane <= 7	BFMLALT Vd.4S,Vn.8H,Vm.H[lane]	Vd.4S -> result	A32/A64

2.15 Modal 8-bit floating-point intrinsics

2.15.1 Data type conversion

2.15.1.1 Conversions

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8_t vcvtt1_bf16_mf8_fpm(mfloat8x8_t vn, fpm_t fpm)</code>	<code>vn -> Vn.8B</code>	<code>BF1CVTL Vd.8H,Vn.8B</code>	<code>Vd.8H -> result</code>	A64
<code>bfloat16x8_t vcvtt1_low_bf16_mf8_fpm(mfloat8x16_t vn, fpm_t fpm)</code>	<code>vn -> Vn.8B</code>	<code>BF1CVTL Vd.8H,Vn.8B</code>	<code>Vd.8H -> result</code>	A64
<code>bfloat16x8_t vcvtt2_bf16_mf8_fpm(mfloat8x8_t vn, fpm_t fpm)</code>	<code>vn -> Vn.8B</code>	<code>BF2CVTL Vd.8H,Vn.8B</code>	<code>Vd.8H -> result</code>	A64
<code>bfloat16x8_t vcvtt2_low_bf16_mf8_fpm(mfloat8x16_t vn, fpm_t fpm)</code>	<code>vn -> Vn.8B</code>	<code>BF2CVTL Vd.8H,Vn.8B</code>	<code>Vd.8H -> result</code>	A64
<code>bfloat16x8_t vcvtt1_high_bf16_mf8_fpm(mfloat8x16_t vn, fpm_t fpm)</code>	<code>vn -> Vn.16B</code>	<code>BF1CVTL2 Vd.8H,Vn.16B</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>bfloat16x8_t vcvtt2_high_bf16_mf8_fpm(vn -> Vn.16B mfloat8x16_t vn, fpm_t fpm)</code>		<code>BF2CVTL2 Vd.8H,Vn.16B</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vcvtt1_f16_mf8_fpm(vn -> Vn.8B mfloat8x8_t vn, fpm_t fpm)</code>		<code>F1CVTL Vd.8H,Vn.8B</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vcvtt1_low_f16_mf8_fpm(vn -> Vn.8B mfloat8x16_t vn, fpm_t fpm)</code>		<code>F1CVTL Vd.8H,Vn.8B</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vcvtt2_f16_mf8_fpm(vn -> Vn.8B mfloat8x8_t vn, fpm_t fpm)</code>		<code>F2CVTL Vd.8H,Vn.8B</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vcvtt2_low_f16_mf8_fpm(vn -> Vn.8B mfloat8x16_t vn, fpm_t fpm)</code>		<code>F2CVTL Vd.8H,Vn.8B</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vcvtt1_high_f16_mf8_fpm(vn -> Vn.16B mfloat8x16_t vn, fpm_t fpm)</code>		<code>F1CVTL2 Vd.8H,Vn.16B</code>	<code>Vd.8H -> result</code>	A64
<code>float16x8_t vcvtt2_high_f16_mf8_fpm(vn -> Vn.16B mfloat8x16_t vn, fpm_t fpm)</code>		<code>F2CVTL2 Vd.8H,Vn.16B</code>	<code>Vd.8H -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>mfloat8x8_t vcvtfm8f32fpm(float32x4_t vn, float32x4_t vm, fpm_t fpm)</code>	<code>vn -> Vn.4S</code> <code>vm -> Vm.4S</code>	<code>FCVTN Vd.8B, Vn.4S, Vm.4S</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vcvthighmf8f32fpm(mfloat8x8_t vd, float32x4_t vn, float32x4_t vm, fpm_t fpm)</code>	<code>vn -> Vn.4S</code> <code>vm -> Vm.4S</code>	<code>FCVTN2 Vd.16B, Vn.4S, Vm.4S</code>	<code>Vd.16B -> result</code>	A64
<code>mfloat8x8_t vcvtfm8f16fpm(float16x4_t vn, float16x4_t vm, fpm_t fpm)</code>	<code>vn -> Vn.4H</code> <code>vm -> Vm.4H</code>	<code>FCVTN Vd.8B, Vn.4H, Vm.4H</code>	<code>Vd.8B -> result</code>	A64
<code>mfloat8x16_t vcvttqmf8f16fpm(float16x8_t vn, float16x8_t vm, fpm_t fpm)</code>	<code>vn -> Vn.8H</code> <code>vm -> Vm.8H</code>	<code>FCVTN Vd.16B, Vn.8H, Vm.8H</code>	<code>Vd.16B -> result</code>	A64

2.15.2 Vector arithmetic

2.15.2.1 Exponent

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vscale_f16(float16x4_t vn, int16x4_t vm)	vn -> Vn.4H vm -> Vm.4H	FSCALE Vd.4H, Vn.4H, Vm.4H	Vd.4H -> result	A64
float16x8_t vscaleq_f16(float16x8_t vn, int16x8_t vm)	vn -> Vn.8H vm -> Vm.8H	FSCALE Vd.8H, Vn.8H, Vm.8H	Vd.8H -> result	A64
float32x2_t vscale_f32(float32x2_t vn, int32x2_t vm)	vn -> Vn.2S vm -> Vm.2S	FSCALE Vd.2S, Vn.2S, Vm.2S	Vd.2S -> result	A64
float32x4_t vscaleq_f32(float32x4_t vn, int32x4_t vm)	vn -> Vn.4S vm -> Vm.4S	FSCALE Vd.4S, Vn.4S, Vm.4S	Vd.4S -> result	A64
float64x2_t vscaleq_f64(float64x2_t vn, int64x2_t vm)	vn -> Vn.2D vm -> Vm.2D	FSCALE Vd.2D, Vn.2D, Vm.2D	Vd.2D -> result	A64

2.15.2.2 Dot product

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x2_t vdot_f32_mf8_fpm(float32x2_t vd, mfloat8x8_t vn, mfloat8x8_t vm, fpm_t fpm)	vd -> Vd.2S vn -> Vn.8B vm -> Vm.8B	FDOT Vd.2S, Vn.8B, Vm.8B	Vd.2S -> result	A64
float32x4_t vdotq_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, fpm_t fpm)	vd -> Vd.4S vn -> Vn.16B vm -> Vm.16B	FDOT Vd.4S, Vn.16B, Vm.16B	Vd.4S -> result	A64
float32x2_t vdot_lane_f32_mf8_fpm(float32x2_t vd, mfloat8x8_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)	vd -> Vd.2S vn -> Vn.8B vm -> Vm.4B 0 <= lane <= 1	FDOT Vd.2S, Vn.8B, Vm.4B[lane]	Vd.2S -> result	A64
float32x2_t vdot_laneq_f32_mf8_fpm(float32x2_t vd, mfloat8x8_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)	vd -> Vd.2S vn -> Vn.16B vm -> Vm.4B 0 <= lane <= 3	FDOT Vd.2S, Vn.8B, Vm.4B[lane]	Vd.2S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vdotq_lane_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)	vd -> Vd.4S vn -> Vn.8B vm -> Vm.4B 0 <= lane <= 1	FDOT Vd.4S, Vn.8B, Vm.4B[lane]	Vd.4S -> result	A64
float32x4_t vdotq_laneq_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)	vd -> Vd.4S vn -> Vn.16 vm -> Vm.4B 0 <= lane <= 3	FDOT Vd.4S, Vn.8B, Vm.4B[lane]	Vd.4SB -> result	A64
float16x4_t vdot_f16_mf8_fpm(float16x4_t vd, mfloat8x8_t vn, mfloat8x8_t vm, fpm_t fpm)	vd -> Vd.4H vn -> Vn.8B vm -> Vm.8B	FDOT Vd.4H, Vn.8B, Vm.8B	Vd.4H -> result	A64
float16x8_t vdotq_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x16_t vm, fpm_t fpm)	vd -> Vd.8H vn -> Vn.16B vm -> Vm.16B	FDOT Vd.8H, Vn.16B, Vm.16B	Vd.8H -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x4_t vdot_lane_f16_mf8_fpm(float16x4_t vd, mfloat8x8_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)	vd -> Vd.4H vn -> Vn.8B vm -> Vm.2B 0 <= lane <= 3	FDOT Vd.4H, Vn.8B, Vm.2B[lane]	Vd.4H -> result	A64
float16x4_t vdot_laneq_f16_mf8_fpm(float16x4_t vd, mfloat8x8_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)	vd -> Vd.4H vn -> Vn.8B vm -> Vm.2B 0 <= lane <= 7	FDOT Vd.4H, Vn.8B, Vm.2B[lane]	Vd.4H -> result	A64
float16x8_t vdotq_lane_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)	vd -> Vd.8H vn -> Vn.16B vm -> Vm.2B 0 <= lane <= 3	FDOT Vd.8H, Vn.16B, Vm.2B[lane]	Vd.8H -> result	A64
float16x8_t vdotq_laneq_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)	vd -> Vd.8H vn -> Vn.16B vm -> Vm.2B 0 <= lane <= 7	FDOT Vd.8H, Vn.16B, Vm.2B[lane]	Vd.8H -> result	A64

2.15.2.3 Multiply

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float16x8_t vmlalbq_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x16_t vm, fpm_t fpm)</code>	<code>vd -> Vd.8H vn -> Vn.16B vm -> Vm.16B</code>	<code>FMLALB Vd.8H, Vn.16B, Vm.16B</code>	<code>Vd.8H -> result</code>	<code>A64</code>
<code>float16x8_t vmlaltq_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x16_t vm, fpm_t fpm)</code>	<code>vd -> Vd.8H vn -> Vn.16B vm -> Vm.16B</code>	<code>FMLALT Vd.8H, Vn.16B, Vm.16B</code>	<code>Vd.8H -> result</code>	<code>A64</code>
<code>float16x8_t vmlalbq_lane_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)</code>	<code>vd -> Vd.8H vn -> Vn.16B vm -> Vm.B 0 <= lane <= 7</code>	<code>FMLALB Vd.8H, Vn.16B, Vm.B[lane]</code>	<code>Vd.8H -> result</code>	<code>A64</code>
<code>float16x8_t vmlalbq_laneq_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)</code>	<code>vd -> Vd.8H vn -> Vn.16B vm -> Vm.B 0 <= lane <= 15</code>	<code>FMLALB Vd.8H, Vn.16B, Vm.B[lane]</code>	<code>Vd.8H -> result</code>	<code>A64</code>

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float16x8_t vmlaltq_lane_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)	vd -> Vd.8H vn -> Vn.16B vm -> Vm.B 0 <= lane <= 7	FMLALT Vd.8H, Vn.16B, Vm.B[lane]	Vd.8H -> result	A64
float16x8_t vmlaltq_laneq_f16_mf8_fpm(float16x8_t vd, mfloat8x16_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)	vd -> Vd.8H vn -> Vn.16B vm -> Vm.B 0 <= lane <= 15	FMLALT Vd.8H, Vn.16B, Vm.B[lane]	Vd.8H -> result	A64
float32x4_t vmlallbbq_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, fpm_t fpm)	vd -> Vd.4S vn -> Vn.16B vm -> Vm.16B	FMLALLBB Vd.4S, Vn.16B, Vm.16B	Vd.4S -> result	A64
float32x4_t vmlallbtq_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, fpm_t fpm)	vd -> Vd.4S vn -> Vn.16B vm -> Vm.16B	FMLALLBT Vd.4S, Vn.16B, Vm.16B	Vd.4S -> result	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vmlalltbq_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, fpm_t fpm)</code>	<code>vd -> Vd.4S vn -> Vn.16B vm -> Vm.16B</code>	<code>FMLALLTB Vd.4S, Vn.16B, Vm.16B</code>	<code>Vd.4S -> result</code>	A64
<code>float32x4_t vmlallttq_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, fpm_t fpm)</code>	<code>vd -> Vd.4S vn -> Vn.16B vm -> Vm.16B</code>	<code>FMLALLTT Vd.4S, Vn.16B, Vm.16B</code>	<code>Vd.4S -> result</code>	A64
<code>float32x4_t vmlallbbq_lane_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)</code>	<code>vd -> Vd.4S vm -> Vn.16B vm -> Vm.B 0 <= lane <= 7</code>	<code>FMLALLBB Vd.4S, Vn.16B, Vm.B[lane]</code>	<code>Vd.4S -> result</code>	A64
<code>float32x4_t vmlallbbq_laneq_f32_mf8_fpm(float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)</code>	<code>vd -> Vd.4S vm -> Vn.16B vm -> Vm.B 0 <= lane <= 15</code>	<code>FMLALLBB Vd.4S, Vn.16B, Vm.B[lane]</code>	<code>Vd.4S -> result</code>	A64

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
float32x4_t vmlallbtq_lane_f32_mf8_fpm4d -> Vd.4S float32x4_t vd, mfloat8x16_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)	vm -> Vn.16B vm -> Vm.B 0 <= lane <= 7	FMLALLBB Vd.4S, Vn.16B, Vm.B[lane] Vd.4S -> result	A64	
float32x4_t vmlallbtq_laneq_f32_mf8_fpm4d -> Vd.4S float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)	vm -> Vn.16B vm -> Vm.B 0 <= lane <= 15	FMLALLBB Vd.4S, Vn.16B, Vm.B[lane] Vd.4S -> result	A64	
float32x4_t vmlalltbq_lane_f32_mf8_fpm4d -> Vd.4S float32x4_t vd, mfloat8x16_t vn, mfloat8x8_t vm, const int lane, fpm_t fpm)	vm -> Vn.16B vm -> Vm.B 0 <= lane <= 7	FMLALLBB Vd.4S, Vn.16B, Vm.B[lane] Vd.4S -> result	A64	
float32x4_t vmlalltbq_laneq_f32_mf8_fpm4d -> Vd.4S float32x4_t vd, mfloat8x16_t vn, mfloat8x16_t vm, const int lane, fpm_t fpm)	vm -> Vn.16B vm -> Vm.B 0 <= lane <= 15	FMLALLBB Vd.4S, Vn.16B, Vm.B[lane] Vd.4S -> result	A64	

Intrinsic	Argument preparation	AArch64 Instruction	Result	Supported architectures
<code>float32x4_t vmlallttq_lane_f32_mf8_fpm4d -> Vd.4S</code> <code>float32x4_t vd,</code> <code>mfloat8x16_t vn,</code> <code>mfloat8x8_t vm,</code> <code>const int lane,</code> <code>fpm_t fpm)</code>	<code>vm -> Vn.16B</code> <code>vm -> Vm.B</code> <code>0 <= lane <= 7</code>	FMLALLBB Vd.4S, Vn.16B, Vm.B[lane] Vd.4S -> result	A64	
<code>float32x4_t vmlallttq_laneq_f32_mf8_fpm4d -> Vd.4S</code> <code>float32x4_t vd,</code> <code>mfloat8x16_t vn,</code> <code>mfloat8x16_t vm,</code> <code>const int lane,</code> <code>fpm_t fpm)</code>	<code>vm -> Vn.16B</code> <code>vm -> Vm.B</code> <code>0 <= lane <= 15</code>	FMLALLBB Vd.4S, Vn.16B, Vm.B[lane] Vd.4S -> result	A64	

2.15.2.3.1 Multiply-accumulate and widen