Add/subtract (extended register)<R>

Original text: Is a width specifier, encoded in "option", where x11->X, otherwise W.

Where:

<R> Is a width specifier, encoded in “option”:

option	<R>
00x	W
010	W
x11	X
10x	W
110	W

Add/subtract (extended register)<extend>

Original text: Is the extension to be applied to the second source operand, encoded in "option", where 000->UXTB, 001->UXTH, 010->LSL|UXTW, 011->UXTX, 100->SXTB, 101->SXTH, 110->SXTW, 111->SXTX. * If "Rd" or "Rn" is '11111' (WSP) and "option" is '010' then LSL is preferred, but may be omitted when "imm3" is '000'. In all other cases <extend> is required and must be UXTW when "option" is '010'.

Where:

<extend> Is the extension to be applied to the second source operand, encoded in “option”:

option	<extend>
000	UXTB
001	UXTH
010	LSL|UXTW
011	UXTX
100	SXTB
101	SXTH
110	SXTW
111	SXTX

If "Rd" or "Rn" is '11111' (WSP) and "option" is '010' then LSL is preferred, but may be omitted when "imm3" is '000'. In all other cases <extend> is required and must be UXTW when "option" is '010'.

Add/subtract (extended register)<extend>

Original text: Is the extension to be applied to the second source operand, encoded in "option", where 000->UXTB, 001->UXTH, 010->LSL|UXTW, 011->UXTX, 100->SXTB, 101->SXTH, 110->SXTW, 111-

>SXTX. * If "Rn" is '11111' (WSP) and "option" is '010' then LSL is preferred, but may be omitted when "imm3" is '000'. In all other cases <extend> is required and must be UXTW when "option" is '010'.

Where:

<extend> Is the extension to be applied to the second source operand, encoded in “option”:

option	<extend>
000	UXTB
001	UXTH
010	LSL|UXTW
011	UXTX
100	SXTB
101	SXTH
110	SXTW
111	SXTX

If "Rn" is '11111' (WSP) and "option" is '010' then LSL is preferred, but may be omitted when "imm3" is '000'. In all other cases <extend> is required and must be UXTW when "option" is '010'.

Add/subtract (extended register)<extend>

Original text: Is the extension to be applied to the second source operand, encoded in "option", where 000->UXTB, 001->UXTH, 010->UXTW, 011->LSL|UXTX, 100->SXTB, 101->SXTH, 110->SXTW, 111->SXTX. * If "Rd" or "Rn" is '11111' (SP) and "option" is '011' then LSL is preferred, but may be omitted when "imm3" is '000'. In all other cases <extend> is required and must be UXTX when "option" is '011'.

Where:

<extend> Is the extension to be applied to the second source operand, encoded in “option”:

option	<extend>
000	UXTB
001	UXTH
010	UXTW
011	LSL|UXTX
100	SXTB
101	SXTH
110	SXTW
111	SXTX

If "Rd" or "Rn" is '11111' (SP) and "option" is '011' then LSL is preferred, but may be omitted when "imm3" is '000'. In all other cases <extend> is required and must be UXTX when "option" is '011'.

Add/subtract (extended register)<extend>

Original text: Is the extension to be applied to the second source operand, encoded in "option", where 000->UXTB, 001->UXTH, 010->UXTW, 011->LSL|UXTX, 100->SXTB, 101->SXTH, 110->SXTW, 111->SXTX. * If "Rn" is '11111' (SP) and "option" is '011' then LSL is preferred, but may be omitted when "imm3" is '000'. In all other cases <extend> is required and must be UXTX when "option" is '011'.

Where:

<extend> Is the extension to be applied to the second source operand, encoded in “option”:

option	<extend>
000	UXTB
001	UXTH
010	UXTW
011	LSL|UXTX
100	SXTB
101	SXTH
110	SXTW
111	SXTX

If "Rn" is '11111' (SP) and "option" is '011' then LSL is preferred, but may be omitted when "imm3" is '000'. In all other cases <extend> is required and must be UXTX when "option" is '011'.

Add/subtract (immediate)<shift>

Original text: Is the optional left shift to apply to the immediate, defaulting to LSL #0 and encoded in "sh", where 0->LSL #0, 1->LSL #12.

Where:

<shift> Is the optional left shift to apply to the immediate, defaulting to LSL #0 and encoded in “sh”:

sh	<shift>
0	LSL #0
1	LSL #12

Add/subtract (shifted register)<shift>

Original text: Is the optional shift type to be applied to the second source operand, defaulting to LSL and encoded in "shift", where 00->LSL, 01->LSR, 10->ASR, 11->RESERVED.

Where:

<shift> Is the optional shift type to be applied to the second source operand, defaulting to LSL and encoded in “shift”:

shift	<shift>
00	LSL
01	LSR
10	ASR
11	RESERVED

Advanced SIMD across lanes<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->4H, 1->8H.

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	4H
1	8H

Advanced SIMD across lanes<T>

Original text: Is an arrangement specifier, encoded in "Q:sz", where 10->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “Q:sz”:

Q	sz	<T>
0	x	RESERVED
1	0	4S
1	1	RESERVED

Advanced SIMD across lanes<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H and 101->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	RESERVED
10	1	4S
11	x	RESERVED

Advanced SIMD across lanes<V>

Original text: Is the destination width specifier, encoded in "size", where 00->B, 01->H, 10->S and 11->RESERVED.

Where:

<V> Is the destination width specifier, encoded in “size”:

size	<V>
00	B
01	H
10	S
11	RESERVED

Advanced SIMD across lanes<V>

Original text: Is the destination width specifier, encoded in "size", where 00->H, 01->S, 10->D and 11->RESERVED.

Where:

<V> Is the destination width specifier, encoded in “size”:

size	<V>
00	H
01	S
10	D
11	RESERVED

Advanced SIMD across lanes<V>

Original text: Is the destination width specifier, encoded in "sz", where 0->S and 1->RESERVED.

Where:

<V> Is the destination width specifier, encoded in “sz”:

sz	<V>
0	S
1	RESERVED

Advanced SIMD copy<T>

Original text: Is an arrangement specifier, encoded in "imm5:Q", where xxxx10->8B, xxxx11->16B, xxx100->4H, xxx101->8H, xx1000->2S, xx1001->4S, x10000->RESERVED, x10001->2D, x0000x->RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “imm5:Q”:

imm5	Q	<T>
x0000	x	RESERVED
xxxx1	0	8B
xxxx1	1	16B
xxx10	0	4H
xxx10	1	8H
xx100	0	2S
xx100	1	4S
x1000	0	RESERVED
x1000	1	2D

Advanced SIMD copy<Ts>

Original text: Is an element size specifier, encoded in "imm5", where xxxx1->B, xxx10->H, xx100->S, x1000->D, x0000->RESERVED.

Where:

<Ts> Is an element size specifier, encoded in “imm5”:

imm5	<Ts>
x0000	RESERVED
xxxx1	B
xxx10	H
xx100	S
x1000	D

Advanced SIMD copy<Ts>

Original text: Is an element size specifier, encoded in "imm5", where xxxx1->B, xxx10->H, xx100->S, xx000->RESERVED.

Where:

<Ts> Is an element size specifier, encoded in “imm5”:

imm5	<Ts>
xx000	RESERVED
xxxx1	B
xxx10	H
xx100	S

Advanced SIMD copy<Ts>

Original text: Is an element size specifier, encoded in "imm5", where xxxx1->B, xxx10->H, xxx00->RESERVED.

Where:

<Ts> Is an element size specifier, encoded in “imm5”:

imm5	<Ts>
xxx00	RESERVED
xxxx1	B
xxx10	H

Advanced SIMD copy<Ts>

Original text: Is an element size specifier, encoded in "imm5", where xxxx1->RESERVED, xxx10->RESERVED, xx100-> RESERVED, x1000->D, x0000->RESERVED.

Where:

<Ts> Is an element size specifier, encoded in “imm5”:

imm5	<Ts>
x0000	RESERVED
xxxx1	RESERVED
xxx10	RESERVED
xx100	RESERVED
x1000	D

Advanced SIMD copy<index1>

Original text: Is the destination element index encoded in "imm5", where xxxx1->imm5<4:1>, xxx10->imm5<4:2>, xx100->imm5<4:3>, x1000->imm5<4>, x0000->RESERVED.

Where:

<index1> Is the destination element index encoded in “imm5”:

imm5	<index1>
x0000	RESERVED
xxxx1	imm5<4:1>
xxx10	imm5<4:2>
xx100	imm5<4:3>
x1000	imm5<4>

Advanced SIMD copy<index>

Original text: Is the element index encoded in "imm5", where xxxx1->imm5<4:1>, xxx10->imm5<4:2>, xx100->imm5<4:3>, x1000->imm5<4>, x0000->RESERVED.

Where:

<index> Is the element index encoded in “imm5”:

imm5	<index>
x0000	RESERVED
xxxx1	imm5<4:1>
xxx10	imm5<4:2>
xx100	imm5<4:3>
x1000	imm5<4>

Advanced SIMD copy<index>

Original text: Is the element index encoded in "imm5", where xxxx1->imm5<4:1>, xxx10->imm5<4:2>, xx100->imm5<4:3>, xx000->RESERVED.

Where:

<index> Is the element index encoded in “imm5”:

imm5	<index>
xx000	RESERVED
xxxx1	imm5<4:1>
xxx10	imm5<4:2>
xx100	imm5<4:3>

Advanced SIMD copy<index>

Original text: Is the element index encoded in "imm5", where xxxx1->imm5<4:1>, xxx10->imm5<4:2>, xxx00->RESERVED.

Where:

<index> Is the element index encoded in “imm5”:

imm5	<index>
xxx00	RESERVED
xxxx1	imm5<4:1>
xxx10	imm5<4:2>

Advanced SIMD copy<index2>

Original text: Is the source element index encoded in "imm5:imm4", using "imm5", where xxxx1->imm4<3:0>, xxx10->imm4<3:1>, xx100->imm4<3:2>, x1000->imm4<3>, x0000->RESERVED. Unspecified bits in "imm4" are ignored but should be set to zero by an assembler.

Where:

<index2> Is the source element index encoded in “imm5:imm4”, based on “imm5”:

imm5	<index2>
x0000	RESERVED
xxxx1	imm4<3:0>
xxx10	imm4<3:1>
xx100	imm4<3:2>
x1000	imm4<3>

Unspecified bits in "imm4" are ignored but should be set to zero by an assembler.

Advanced SIMD copy<R>

Original text: Is the width specifier for the general-purpose source register, encoded in "imm5", where xxxx1->W, xxx10->W, xx100->W, x1000->X, x0000->RESERVED.

Where:

<R> Is the width specifier for the general-purpose source register, encoded in “imm5”:

imm5	<R>
x0000	RESERVED
xxxx1	W
xxx10	W
xx100	W
x1000	X

Advanced SIMD copy<R>

Original text: Is the width specifier for the general-purpose source register, encoded in "imm5", where xxxx1->W, xxx10->W, xx100->W,

x1000->X, x0000->RESERVED. Unspecified bits in "imm5" are ignored but should be set to zero by an assembler.

Where:

<R> Is the width specifier for the general-purpose source register, encoded in “imm5”:

imm5	<R>
x0000	RESERVED
xxxx1	W
xxx10	W
xx100	W
x1000	X

Unspecified bits in "imm5" are ignored but should be set to zero by an assembler.

Advanced SIMD extract<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->8B and 1->16B

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	8B
1	16B

Advanced SIMD extract<index>

Original text: Is the lowest numbered byte element to be extracted, encoded in "Q:imm4", using "Q:imm4<3>", where 00->imm4<2:0>, 01->RESERVED, 1x->imm4.

Where:

<index> Is the lowest numbered byte element to be extracted, encoded in “Q:imm4”, based on “Q:imm4<3>”:

Q	imm4<3>	<index>
0	0	imm4<2:0>
0	1	RESERVED
1	x	imm4

Advanced SIMD load/store multiple structures<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S and 111->2D, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	RESERVED
11	1	2D

Advanced SIMD load/store multiple structures<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S, 110->1D and 111->2D.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	1D
11	1	2D

Advanced SIMD load/store multiple structures (post-indexed)<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S and 111->2D, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	RESERVED
11	1	2D

Advanced SIMD load/store multiple structures (post-indexed)<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S, 110->1D and 111->2D.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	1D
11	1	2D

Advanced SIMD load/store multiple structures (post-indexed)<imm>

Original text: Is the post-index immediate offset, encoded in "Q", where 0->#16, 1->#32.

Where:

<imm> Is the post-index immediate offset, encoded in “Q”:

Q	<imm>
0	#16
1	#32

Advanced SIMD load/store multiple structures (post-indexed)<imm>

Original text: Is the post-index immediate offset, encoded in "Q", where 0->#24, 1->#48.

Where:

<imm> Is the post-index immediate offset, encoded in “Q”:

Q	<imm>
0	#24
1	#48

Advanced SIMD load/store multiple structures (post-indexed)<imm>

Original text: Is the post-index immediate offset, encoded in "Q", where 0->#32, 1->#64.

Where:

<imm> Is the post-index immediate offset, encoded in “Q”:

Q	<imm>
0	#32
1	#64

Advanced SIMD load/store multiple structures (post-indexed)<imm>

Original text: Is the post-index immediate offset, encoded in "Q", where 0->#8, 1->#16.

Where:

<imm> Is the post-index immediate offset, encoded in “Q”:

Q	<imm>
0	#8
1	#16

Advanced SIMD load/store single structure<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S, 110->1D and 111->2D.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	1D
11	1	2D

Advanced SIMD load/store single structure (post-indexed)<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S, 110->1D and 111->2D.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	1D
11	1	2D

Advanced SIMD load/store single structure (post-indexed)<imm>

Original text: Is the post-index immediate offset, encoded in "size", where 00->#1, 01->#2, 10->#4, 11->#8.

Where:

<imm> Is the post-index immediate offset, encoded in “size”:

size	<imm>
00	#1
01	#2
10	#4
11	#8

Advanced SIMD load/store single structure (post-indexed)<imm>

Original text: Is the post-index immediate offset, encoded in "size", where 00->#2, 01->#4, 10->#8, 11->#16.

Where:

<imm> Is the post-index immediate offset, encoded in “size”:

size	<imm>
00	#2
01	#4
10	#8
11	#16

Advanced SIMD load/store single structure (post-indexed)<imm>

Original text: Is the post-index immediate offset, encoded in "size", where 00->#3, 01->#6, 10->#12, 11->#24.

Where:

<imm> Is the post-index immediate offset, encoded in “size”:

size	<imm>
00	#3
01	#6
10	#12
11	#24

Advanced SIMD load/store single structure (post-indexed)<imm>

Original text: Is the post-index immediate offset, encoded in "size", where 00->#4, 01->#8, 10->#16, 11->#32.

Where:

<imm> Is the post-index immediate offset, encoded in “size”:

size	<imm>
00	#4
01	#8
10	#16
11	#32

Advanced SIMD modified immediate<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->2S and 1->4S.

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	2S
1	4S

Advanced SIMD modified immediate<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->4H and 1->8H.

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	4H
1	8H

Advanced SIMD modified immediate<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->8B and 1->16B.

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	8B
1	16B

Advanced SIMD modified immediate<amount>

Original text: Is the shift amount encoded in "cmode<0>", where 0->8 and 1->16.

Where:

<amount> Is the shift amount encoded in “cmode<0>”:

cmode<0>	<amount>
0	8
1	16

Advanced SIMD modified immediate<amount>

Original text: Is the shift amount encoded in "cmode<1>", where 0->0 and 1->8, defaulting to 0 if LSL is omitted.

Where:

<amount> Is the shift amount encoded in “cmode<1>”:

cmode<1>	<amount>
0	0
1	8

defaulting to 0 if LSL is omitted.

Advanced SIMD modified immediate<amount>

Original text: Is the shift amount encoded in "cmode<2:1>", where 00->0, 01->8, 10->16 and 11->24, defaulting to 0 if LSL is omitted.

Where:

<amount> Is the shift amount encoded in “cmode<2:1>”:

cmode<2:1>	<amount>
00	0
01	8
10	16
11	24

defaulting to 0 if LSL is omitted.

Advanced SIMD permute<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S and 111->2D, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	RESERVED
11	1	2D

Advanced SIMD scalar copy<V>

Original text: Is the destination width specifier, encoded in "imm5", where xxxx1->B, xxx10->H, xx100->S, x1000->D, x0000->RESERVED.

Where:

<V> Is the destination width specifier, encoded in “imm5”:

imm5	<V>
x0000	RESERVED
xxxx1	B
xxx10	H
xx100	S
x1000	D

Advanced SIMD scalar copy<index>

Original text: Is the element index encoded in "imm5", where xxxx1->imm5<4:1>, xxx10->imm5<4:2>, xx100->imm5<4:3>, x1000->imm5<4>, x0000->RESERVED.

Where:

<index> Is the element index encoded in “imm5”:

imm5	<index>
x0000	RESERVED
xxxx1	imm5<4:1>
xxx10	imm5<4:2>
xx100	imm5<4:3>
x1000	imm5<4>

Advanced SIMD scalar copy<T>

Original text: Is the element width specifier, encoded in "imm5", where xxxx1->B, xxx10->H, xx100->S, x1000->D, x0000->RESERVED.

Where:

<T> Is the element width specifier, encoded in “imm5”:

imm5	<T>
x0000	RESERVED
xxxx1	B
xxx10	H
xx100	S
x1000	D

Advanced SIMD scalar pairwise<V>

Original text: Is the destination width specifier, encoded in "size", where 11->D, otherwise RESERVED.

Where:

<V> Is the destination width specifier, encoded in “size”:

size	<V>
0x	RESERVED
10	RESERVED
11	D

Advanced SIMD scalar pairwise<V>

Original text: Is the destination width specifier, encoded in "sz", where 0->H, otherwise RESERVED.

Where:

<V> Is the destination width specifier, encoded in “sz”:

sz	<V>
0	H
1	RESERVED

Advanced SIMD scalar pairwise<V>

Original text: Is the destination width specifier, encoded in "sz", where 0->S and 1->D.

Where:

<V> Is the destination width specifier, encoded in “sz”:

sz	<V>
0	S
1	D

Advanced SIMD scalar pairwise<T>

Original text: Is the source arrangement specifier, encoded in "size", where 11->2D, otherwise RESERVED.

Where:

<T> Is the source arrangement specifier, encoded in “size”:

size	<T>
0x	RESERVED
10	RESERVED
11	2D

Advanced SIMD scalar pairwise<T>

Original text: Is the source arrangement specifier, encoded in "sz", where 0->2H, otherwise RESERVED.

Where:

<T> Is the source arrangement specifier, encoded in “sz”:

sz	<T>
0	2H
1	RESERVED

Advanced SIMD scalar pairwise<T>

Original text: Is the source arrangement specifier, encoded in "sz", where 0->2S and 1->2D.

Where:

<T> Is the source arrangement specifier, encoded in “sz”:

sz	<T>
0	2S
1	2D

Advanced SIMD scalar shift by immediate<V>

Original text: Is a width specifier, encoded in "immh", where 0001->B, 001x->H, 01xx->S and 1xxx->D, otherwise RESERVED.

Where:

<V> Is a width specifier, encoded in “immh”:

immh	<V>
0000	RESERVED
0001	B
001x	H
01xx	S
1xxx	D

Advanced SIMD scalar shift by immediate<V>

Original text: Is a width specifier, encoded in "immh", where 001x->H, 01xx->S and 1xxx->D, otherwise RESERVED.

Where:

<V> Is a width specifier, encoded in “immh”:

immh	<V>
000x	RESERVED
001x	H
01xx	S
1xxx	D

Advanced SIMD scalar shift by immediate<V>

Original text: Is a width specifier, encoded in "immh", where 1xxx->D, otherwise RESERVED.

Where:

<V> Is a width specifier, encoded in “immh”:

immh	<V>
0xxx	RESERVED
1xxx	D

Advanced SIMD scalar shift by immediate<Vb>

Original text: Is the destination width specifier, encoded in "immh", where 0001->B, 001x->H and 01xx->S, otherwise RESERVED.

Where:

<Vb> Is the destination width specifier, encoded in “immh”:

immh	<Vb>
0000	RESERVED
0001	B
001x	H
01xx	S
1xxx	RESERVED

Advanced SIMD scalar shift by immediate<shift>

Original text: Is the left shift amount, in the range 0 to 63, encoded in "immh:immb", based on "immh", where 1xxx->(UInt(immh:immb)-64), otherwise RESERVED.

Where:

<shift> Is the left shift amount, in the range 0 to 63, encoded in “immh:immb”, based on “immh”:

immh	<shift>
0xxx	RESERVED
1xxx	(UInt(immh:immb)-64)

Advanced SIMD scalar shift by immediate<shift>

Original text: Is the left shift amount, in the range 0 to the operand width in bits minus 1, encoded in "immh:immb", based on "immh", where 0001->(UInt(immh:immb)-8), 001x->(UInt(immh:immb)-16), 01xx->(UInt(immh:immb)-32), 1xxx->(UInt(immh:immb)-64), otherwise RESERVED.

Where:

<shift> Is the left shift amount, in the range 0 to the operand width in bits minus 1, encoded in “immh:immb”, based on “immh”:

immh	<shift>
0000	RESERVED
0001	(UInt(immh:immb)-8)
001x	(UInt(immh:immb)-16)
01xx	(UInt(immh:immb)-32)
1xxx	(UInt(immh:immb)-64)

Advanced SIMD scalar shift by immediate<fbits>

Original text: Is the number of fractional bits, in the range 1 to the operand width, encoded in "immh:immb", using "immh", where 001x->(32-Uint(immh:immb)), 01xx->(64-UInt(immh:immb)), 1xxx->(128-UInt(immh:immb)), otherwise RESERVED.

Where:

<fbits> Is the number of fractional bits, in the range 1 to the operand width, encoded in “immh:immb”, based on “immh”:

immh	<fbits>
000x	RESERVED
001x	(32-Uint(immh:immb))
01xx	(64-UInt(immh:immb))
1xxx	(128-UInt(immh:immb))

Advanced SIMD scalar shift by immediate<shift>

Original text: Is the right shift amount, in the range 1 to 64, encoded in "immh:immb", based on "immh", where 1xxx->(128-UInt(immh:immb)), otherwise RESERVED.

Where:

<shift> Is the right shift amount, in the range 1 to 64, encoded in “immh:immb”, based on “immh”:

immh	<shift>
0xxx	RESERVED
1xxx	(128-UInt(immh:immb))

Advanced SIMD scalar shift by immediate<shift>

Original text: Is the right shift amount, in the range 1 to the destination operand width in bits, encoded in "immh:immb", using "immh" where 0001->(16-UInt(immh:immb)), 001x->(32-UInt(immh:immb)), 01xx->(64-UInt(immh:immb)), otherwise RESERVED.

Where:

<shift> Is the right shift amount, in the range 1 to the destination operand width in bits, encoded in “immh:immb”, based on “immh”:

immh	<shift>
0000	RESERVED
0001	(16-UInt(immh:immb))
001x	(32-UInt(immh:immb))
01xx	(64-UInt(immh:immb))
1xxx	RESERVED

Advanced SIMD scalar shift by immediate<Va>

Original text: Is the source width specifier, encoded in "immh", where 0001->H, 001x->S and 01xx->D, otherwise RESERVED.

Where:

<Va> Is the source width specifier, encoded in “immh”:

immh	<Va>
0000	RESERVED
0001	H
001x	S
01xx	D
1xxx	RESERVED

Advanced SIMD scalar three different<Va>

Original text: Is the destination width specifier, encoded in "size", where 01->S and 10->D, otherwise RESERVED.

Where:

<Va> Is the destination width specifier, encoded in “size”:

size	<Va>
00	RESERVED
01	S
10	D
11	RESERVED

Advanced SIMD scalar three different<Vb>

Original text: Is the source width specifier, encoded in "size", where 01->H and 10->S, otherwise RESERVED.

Where:

<Vb> Is the source width specifier, encoded in “size”:

size	<Vb>
00	RESERVED
01	H
10	S
11	RESERVED

Advanced SIMD scalar three same<V>

Original text: Is a width specifier, encoded in "size", where 00->B, 01->H, 10->S, 11->D.

Where:

<V> Is a width specifier, encoded in “size”:

size	<V>
00	B
01	H
10	S
11	D

Advanced SIMD scalar three same<V>

Original text: Is a width specifier, encoded in "size", where 01->H, 10->S, otherwise RESERVED.

Where:

<V> Is a width specifier, encoded in “size”:

size	<V>
00	RESERVED
01	H
10	S
11	RESERVED

Advanced SIMD scalar three same<V>

Original text: Is a width specifier, encoded in "size", where 11->D, otherwise RESERVED.

Where:

<V> Is a width specifier, encoded in “size”:

size	<V>
0x	RESERVED
10	RESERVED
11	D

Advanced SIMD scalar three same<V>

Original text: Is a width specifier, encoded in "sz", where 0->S, 1->D.

Where:

<V> Is a width specifier, encoded in “sz”:

sz	<V>
0	S
1	D

Advanced SIMD scalar three same extra<V>

Original text: Is a width specifier, encoded in "size", where 01->H, 10->S, otherwise RESERVED.

Where:

<V> Is a width specifier, encoded in “size”:

size	<V>
00	RESERVED
01	H
10	S
11	RESERVED

Advanced SIMD scalar two-register miscellaneous<V>

Original text: Is a width specifier, encoded in "size", where 00->B, 01->H, 10->S, 11->D.

Where:

<V> Is a width specifier, encoded in “size”:

size	<V>
00	B
01	H
10	S
11	D

Advanced SIMD scalar two-register miscellaneous<V>

Original text: Is a width specifier, encoded in "size", where 11->D, otherwise RESERVED.

Where:

<V> Is a width specifier, encoded in “size”:

size	<V>
0x	RESERVED
10	RESERVED
11	D

Advanced SIMD scalar two-register miscellaneous<V>

Original text: Is a width specifier, encoded in "sz", where 0->S and 1->D.

Where:

<V> Is a width specifier, encoded in “sz”:

sz	<V>
0	S
1	D

Advanced SIMD scalar two-register miscellaneous<Vb>

Original text: Is the destination width specifier, encoded in "size", where 00->B, 01->H, 10->S and 11->RESERVED.

Where:

<Vb> Is the destination width specifier, encoded in “size”:

size	<Vb>
00	B
01	H
10	S
11	RESERVED

Advanced SIMD scalar two-register miscellaneous<Vb>

Original text: Is the destination width specifier, encoded in "sz", where 0->RESERVED and 1->S.

Where:

<Vb> Is the destination width specifier, encoded in “sz”:

sz	<Vb>
0	RESERVED
1	S

Advanced SIMD scalar two-register miscellaneous<Va>

Original text: Is the source width specifier, encoded in "size", where 00->H, 01->S, 10->D and 11->RESERVED.

Where:

<Va> Is the source width specifier, encoded in “size”:

size	<Va>
00	H
01	S
10	D
11	RESERVED

Advanced SIMD scalar two-register miscellaneous<Va>

Original text: Is the source width specifier, encoded in "sz", where 0->RESERVED and 1->D.

Where:

<Va> Is the source width specifier, encoded in “sz”:

sz	<Va>
0	RESERVED
1	D

Advanced SIMD scalar x indexed element<V>

Original text: Is a width specifier, encoded in "size", where 01->H, 10->S, otherwise RESERVED.

Where:

<V> Is a width specifier, encoded in “size”:

size	<V>
00	RESERVED
01	H
10	S
11	RESERVED

Advanced SIMD scalar x indexed element<V>

Original text: Is a width specifier, encoded in "sz", where 0->S and 1->D.

Where:

<V> Is a width specifier, encoded in “sz”:

sz	<V>
0	S
1	D

Advanced SIMD scalar x indexed element<Ts>

Original text: Is an element size specifier, encoded in "size", where 01->H and 10->S, otherwise RESERVED.

Where:

<Ts> Is an element size specifier, encoded in “size”:

size	<Ts>
00	RESERVED
01	H
10	S
11	RESERVED

Advanced SIMD scalar x indexed element<Ts>

Original text: Is an element size specifier, encoded in "sz", where 0->S and 1->D.

Where:

<Ts> Is an element size specifier, encoded in “sz”:

sz	<Ts>
0	S
1	D

Advanced SIMD scalar x indexed element<Va>

Original text: Is the destination width specifier, encoded in "size", where 01->S and 10->D, otherwise RESERVED.

Where:

<Va> Is the destination width specifier, encoded in “size”:

size	<Va>
00	RESERVED
01	S
10	D
11	RESERVED

Advanced SIMD scalar x indexed element<index>

Original text: Is the element index, encoded in "size:L:H:M", using "size", where 01->H:L:M and 10->H:L, otherwise RESERVED.

Where:

<index> Is the element index, encoded in “size:L:H:M”, based on “size”:

size	<index>
00	RESERVED
01	H:L:M
10	H:L
11	RESERVED

Advanced SIMD scalar x indexed element<index>

Original text: Is the element index, encoded in "sz:L:H", using "sz:L", where 0x->H:L, 10->H, otherwise RESERVED.

Where:

<index> Is the element index, encoded in “sz:L:H”, based on “sz:L”:

sz	L	<index>
0	x	H:L
1	0	H
1	1	RESERVED

Advanced SIMD scalar x indexed element<Vm>

Original text: Is the name of the second SIMD&FP source register, encoded in the "size:M:Rm" fields, using "size" where 01->0:Rm and 10->M:Rm, otherwise RESERVED. Restricted to V0-V15 when element size <Ts> is H.

Where:

<Vm> Is the name of the second SIMD&FP source register, encoded in “size:M:Rm”, based on “size”:

size	<Vm>
00	RESERVED
01	0:Rm
10	M:Rm
11	RESERVED

Restricted to V0-V15 when element size <Ts> is H.

Advanced SIMD scalar x indexed element<Vb>

Original text: Is the source width specifier, encoded in "size", where 01->H and 10->S, otherwise RESERVED.

Where:

<Vb> Is the source width specifier, encoded in “size”:

size	<Vb>
00	RESERVED
01	H
10	S
11	RESERVED

Advanced SIMD shift by immediate<Ta>

Original text: Is an arrangement specifier, encoded in "immh", where 0000->SEE(asimdimm), 0001->8H, 001x->4S, 01xx->2D and 1xxx->RESERVED.

Where:

<Ta> Is an arrangement specifier, encoded in “immh”:

immh	<Ta>
0000	SEE(asimdimm)
0001	8H
001x	4S
01xx	2D
1xxx	RESERVED

Advanced SIMD shift by immediate<Tb>

Original text: Is an arrangement specifier, encoded in "immh:Q", where 0000x->SEE(asimdimm), 00010->8B, 00011->16B, 001x0->4H, 001x1->8H, 01xx0->2S, 01xx1->4S and 1xxxx->RESERVED.

Where:

<Tb> Is an arrangement specifier, encoded in “immh:Q”:

immh	Q	<Tb>
0000	x	SEE(asimdimm)
0001	0	8B
0001	1	16B
001x	0	4H
001x	1	8H
01xx	0	2S
01xx	1	4S
1xxx	x	RESERVED

Advanced SIMD shift by immediate<T>

Original text: Is an arrangement specifier, encoded in "immh:Q", where 0000x->SEE(asimdimm), 00010->8B, 00011->16B, 001x0->4H, 001x1->8H, 01xx0->2S, 01xx1->4S, 1xxx0->RESERVED and 1xxx1->2D.

Where:

<T> Is an arrangement specifier, encoded in “immh:Q”:

immh	Q	<T>
0000	x	SEE(asimdimm)
0001	0	8B
0001	1	16B
001x	0	4H
001x	1	8H
01xx	0	2S
01xx	1	4S
1xxx	0	RESERVED
1xxx	1	2D

Advanced SIMD shift by immediate<T>

Original text: Is an arrangement specifier, encoded in "immh:Q", where 0000x->SEE(asimdimm), 0001x->RESERVED, 001x0->4H, 001x1->8H, 01xx0->2S, 01xx1->4S, 1xxx1->2D and 1xxx0->RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “immh:Q”:

immh	Q	<T>
0000	x	SEE(asimdimm)
0001	x	RESERVED
001x	0	4H
001x	1	8H
01xx	0	2S
01xx	1	4S
1xxx	0	RESERVED
1xxx	1	2D

Advanced SIMD shift by immediate<shift>

Original text: Is the left shift amount, in the range 0 to the element width in bits minus 1, encoded in "immh:immb", based on "immh", where 0000->SEE(asimdimm), 0001->(UInt(immh:immb)-8), 001x->(UInt(immh:immb)-16), 01xx->(UInt(immh:immb)-32) and 1xxx->(UInt(immh:immb)-64).

Where:

<shift> Is the left shift amount, in the range 0 to the element width in bits minus 1, encoded in “immh:immb”, based on “immh”:

immh	<shift>
0000	SEE(asimdimm)
0001	(UInt(immh:immb)-8)
001x	(UInt(immh:immb)-16)
01xx	(UInt(immh:immb)-32)
1xxx	(UInt(immh:immb)-64)

Advanced SIMD shift by immediate<shift>

Original text: Is the left shift amount, in the range 0 to the source element width in bits minus 1, encoded in "immh:immb", based on "immh" where 0000->SEE(asimdimm), 0001->(UInt(immh:immb)-8), 001x->(UInt(immh:immb)-16), 01xx->(UInt(immh:immb)-32) and 1xxx->RESERVED.

Where:

<shift> Is the left shift amount, in the range 0 to the source element width in bits minus 1, encoded in “immh:immb”, based on “immh”:

immh	<shift>
0000	SEE(asimdimm)
0001	(UInt(immh:immb)-8)
001x	(UInt(immh:immb)-16)
01xx	(UInt(immh:immb)-32)
1xxx	RESERVED

Advanced SIMD shift by immediate<fbits>

Original text: Is the number of fractional bits, in the range 1 to the element width, encoded in "immh:immb", using "immh", where 0000->SEE(asimdimm), 001x->(32-Uint(immh:immb)), 01xx->(64-UInt(immh:immb)), 1xxx->(128-UInt(immh:immb)), otherwise RESERVED.

Where:

<fbits> Is the number of fractional bits, in the range 1 to the element width, encoded in “immh:immb”, based on “immh”:

immh	<fbits>
0000	SEE(asimdimm)
0001	RESERVED
001x	(32-Uint(immh:immb))
01xx	(64-UInt(immh:immb))
1xxx	(128-UInt(immh:immb))

Advanced SIMD shift by immediate<shift>

Original text: Is the right shift amount, in the range 1 to the destination element width in bits, encoded in "immh:immb", based on "immh" where 0000->SEE(asimdimm), 0001->(16-UInt(immh:immb)), 001x->(32-UInt(immh:immb)), 01xx->(64-UInt(immh:immb)) and 1xxx->RESERVED.

Where:

<shift> Is the right shift amount, in the range 1 to the destination element width in bits, encoded in “immh:immb”, based on “immh”:

immh	<shift>
0000	SEE(asimdimm)
0001	(16-UInt(immh:immb))
001x	(32-UInt(immh:immb))
01xx	(64-UInt(immh:immb))
1xxx	RESERVED

Advanced SIMD shift by immediate<shift>

Original text: Is the right shift amount, in the range 1 to the element width in bits, encoded in "immh:immb", based on "immh", where 0000->SEE(asimdimm), 0001->(16-UInt(immh:immb)), 001x->(32-UInt(immh:immb)), 01xx->(64-UInt(immh:immb)) and 1xxx->(128-UInt(immh:immb)).

Where:

<shift> Is the right shift amount, in the range 1 to the element width in bits, encoded in “immh:immb”, based on “immh”:

immh	<shift>
0000	SEE(asimdimm)
0001	(16-UInt(immh:immb))
001x	(32-UInt(immh:immb))
01xx	(64-UInt(immh:immb))
1xxx	(128-UInt(immh:immb))

Advanced SIMD shift by immediate2

Original text: Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in "Q", where 0->[absent] and 1->[present].

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in “Q”:

Q	2
0	[absent]
1	[present]

Advanced SIMD table lookup<Ta>

Original text: Is an arrangement specifier, encoded in "Q", where 0->8B and 1->16B.

Where:

<Ta> — Is an arrangement specifier, encoded in “Q”:

Q	<Ta>
0	8B
1	16B

Advanced SIMD three different<Ta>

Original text: Is an arrangement specifier, encoded in "size", where 00->8H, 01->4S and 10->2D, otherwise RESERVED.

Where:

<Ta> — Is an arrangement specifier, encoded in “size”:

size	<Ta>
00	8H
01	4S
10	2D
11	RESERVED

Advanced SIMD three different<Ta>

Original text: Is an arrangement specifier, encoded in "size", where 00->8H, 11->1Q, otherwise RESERVED. * The '1Q' arrangement is only allocated in an implementation that includes the Cryptographic Extension, and is otherwise RESERVED.

Where:

<Ta> Is an arrangement specifier, encoded in “size”:

size	<Ta>
00	8H
01	RESERVED
10	RESERVED
11	1Q

The '1Q' arrangement is only allocated in an implementation that includes the Cryptographic Extension, and is otherwise RESERVED.

Advanced SIMD three different<Ta>

Original text: Is an arrangement specifier, encoded in "size", where 01->4S and 10->2D, otherwise RESERVED.

Where:

<Ta> Is an arrangement specifier, encoded in “size”:

size	<Ta>
00	RESERVED
01	4S
10	2D
11	RESERVED

Advanced SIMD three different<Tb>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S and 101->4S, otherwise RESERVED.

Where:

<Tb> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<Tb>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD three different<Tb>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 110->1D, 111->2D, otherwise RESERVED.

Where:

<Tb> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<Tb>
00	0	8B
00	1	16B
01	x	RESERVED
10	x	RESERVED
11	0	1D
11	1	2D

Advanced SIMD three different<Tb>

Original text: Is an arrangement specifier, encoded in "size:Q", where 010->4H, 011->8H, 100->2S and 101->4S, otherwise RESERVED.

Where:

<Tb> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<Tb>
00	x	RESERVED
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD three different2

Original text: Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in "Q", where 0->[absent] and 1->[present].

Where:

2 Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in “Q”:

Q	2
0	[absent]
1	[present]

Advanced SIMD three same<Tb>

Original text: Is an arrangement specifier, encoded in "Q", where 0->2H, 1->4H.

Where:

<Tb> Is an arrangement specifier, encoded in “Q”:

Q	<Tb>
0	2H
1	4H

Advanced SIMD three same<Ta>

Original text: Is an arrangement specifier, encoded in "Q", where 0->2S, 1->4S.

Where:

<Ta> Is an arrangement specifier, encoded in “Q”:

Q	<Ta>
0	2S
1	4S

Advanced SIMD three same<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->8B, 1->16B.

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	8B
1	16B

Advanced SIMD three same<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S, 111->2D, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	RESERVED
11	1	2D

Advanced SIMD three same<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD three same<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	x	RESERVED
1x	x	RESERVED

Advanced SIMD three same<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 010->4H, 011->8H, 100->2S, 101->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	x	RESERVED
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD three same<T>

Original text: Is an arrangement specifier, encoded in "sz:Q", where 00->2S, 01->4S, 11->2D, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “sz:Q”:

sz	Q	<T>
0	0	2S
0	1	4S
1	0	RESERVED
1	1	2D

Advanced SIMD three same (FP16)<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->4H, 1->8H.

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	4H
1	8H

Advanced SIMD three-register extension<Ta>

Original text: Is an arrangement specifier, encoded in "Q", where 0->2S and 1->4S.

Where:

<Ta> Is an arrangement specifier, encoded in “Q”:

Q	<Ta>
0	2S
1	4S

Advanced SIMD three-register extension<Tb>

Original text: Is an arrangement specifier, encoded in "Q", where 0->4H and 1->8H.

Where:

<Tb> Is an arrangement specifier, encoded in “Q”:

Q	<Tb>
0	4H
1	8H

Advanced SIMD three-register extension<Tb>

Original text: Is an arrangement specifier, encoded in "Q", where 0->8B and 1->16B.

Where:

<Tb> Is an arrangement specifier, encoded in “Q”:

Q	<Tb>
0	8B
1	16B

Advanced SIMD three-register extension<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 010->4H, 011->8H, 100->2S, 101->4S, 111->2D, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	x	RESERVED
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	RESERVED
11	1	2D

Advanced SIMD three-register extension<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 010->4H, 011->8H, 100->2S, 101->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	x	RESERVED
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD three-register extension<bt>

Original text: Is the bottom or top element specifier, encoded in "Q" where 0->B, 1->T.

Where:

<bt> Is the bottom or top element specifier, encoded in “Q”:

Q	<bt>
0	B
1	T

Advanced SIMD three-register extension<rotate>

Original text: Is the rotation, encoded in "rot", where 0->90 and 1->270.

Where:

<rotate> Is the rotation, encoded in “rot”:

rot	<rotate>
0	90
1	270

Advanced SIMD three-register extension<rotate>

Original text: Is the rotation, encoded in "rot", where 00->0, 01->90, 10->180 and 11->270.

Where:

<rotate> Is the rotation, encoded in “rot”:

rot	<rotate>
00	0
01	90
10	180
11	270

Advanced SIMD two-register miscellaneous<Ta>

Original text: Is an arrangement specifier, encoded in "Q", where 0->4H and 1->8H.

Where:

<Ta> Is an arrangement specifier, encoded in “Q”:

Q	<Ta>
0	4H
1	8H

Advanced SIMD two-register miscellaneous<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->8B and 1->16B.

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	8B
1	16B

Advanced SIMD two-register miscellaneous<Ta>

Original text: Is an arrangement specifier, encoded in "size", where 00->8H, 01->4S, 10->2D and 11->RESERVED.

Where:

<Ta> Is an arrangement specifier, encoded in “size”:

size	<Ta>
00	8H
01	4S
10	2D
11	RESERVED

Advanced SIMD two-register miscellaneous<Ta>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->4H, 001->8H, 010->2S, 011->4S, 100->1D and 101->2D, otherwise RESERVED.

Where:

<Ta> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<Ta>
00	0	4H
00	1	8H
01	0	2S
01	1	4S
10	0	1D
10	1	2D
11	x	RESERVED

Advanced SIMD two-register miscellaneous<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B and 001->16B, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	x	RESERVED
1x	x	RESERVED

Advanced SIMD two-register miscellaneous<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H and 011->8H, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
1x	x	RESERVED

Advanced SIMD two-register miscellaneous<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S and 101->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD two-register miscellaneous<Tb>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S and 101->4S, otherwise RESERVED.

Where:

<Tb> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<Tb>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD two-register miscellaneous<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S, 110->RESERVED and 111->2D.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	0	RESERVED
11	1	2D

Advanced SIMD two-register miscellaneous<Tb>

Original text: Is an arrangement specifier, encoded in "size:Q", where 000->8B, 001->16B, 010->4H, 011->8H, 100->2S, 101->4S, otherwise RESERVED.

Where:

<Tb> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<Tb>
00	0	8B
00	1	16B
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD two-register miscellaneous<Ta>

Original text: Is an arrangement specifier, encoded in "sz", where 0->4S and 1->2D.

Where:

<Ta> Is an arrangement specifier, encoded in “sz”:

sz	<Ta>
0	4S
1	2D

Advanced SIMD two-register miscellaneous<Ta>

Original text: Is an arrangement specifier, encoded in "sz", where 0->RESERVED and 1->2D.

Where:

<Ta> Is an arrangement specifier, encoded in “sz”:

sz	<Ta>
0	RESERVED
1	2D

Advanced SIMD two-register miscellaneous<T>

Original text: Is an arrangement specifier, encoded in "sz:Q", where 00->2S and 01->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “sz:Q”:

sz	Q	<T>
0	0	2S
0	1	4S
1	x	RESERVED

Advanced SIMD two-register miscellaneous<T>

Original text: Is an arrangement specifier, encoded in "sz:Q", where 00->2S, 01->4S, 10->RESERVED and 11->2D.

Where:

<T> Is an arrangement specifier, encoded in “sz:Q”:

sz	Q	<T>
0	0	2S
0	1	4S
1	0	RESERVED
1	1	2D

Advanced SIMD two-register miscellaneous<Tb>

Original text: Is an arrangement specifier, encoded in "sz:Q", where 00->4H, 01->8H, 10->2S and 11->4S.

Where:

<Tb> Is an arrangement specifier, encoded in “sz:Q”:

sz	Q	<Tb>
0	0	4H
0	1	8H
1	0	2S
1	1	4S

Advanced SIMD two-register miscellaneous<Tb>

Original text: Is an arrangement specifier, encoded in "sz:Q", where 10->2S and 11->4S, otherwise RESERVED.

Where:

<Tb> Is an arrangement specifier, encoded in “sz:Q”:

sz	Q	<Tb>
0	x	RESERVED
1	0	2S
1	1	4S

Advanced SIMD two-register miscellaneous<shift>

Original text: Is the left shift amount, which must be equal to the source element width in bits, encoded in "size", where 00->8, 01->16, 10->32 and 11->RESERVED.

Where:

<shift> Is the left shift amount, which must be equal to the source element width in bits, encoded in “size”:

size	<shift>
00	8
01	16
10	32
11	RESERVED

Advanced SIMD two-register miscellaneous2

Original text: Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in "Q", where 0->[absent] and 1->[present].

Where:

2 Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in “Q”:

Q	2
0	[absent]
1	[present]

Advanced SIMD two-register miscellaneous (FP16)<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->4H, 1->8H.

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	4H
1	8H

Advanced SIMD vector x indexed element<Tb>

Original text: Is an arrangement specifier, encoded in "Q", where 0->2H and 1->4H.

Where:

<Tb> Is an arrangement specifier, encoded in “Q”:

Q	<Tb>
0	2H
1	4H

Advanced SIMD vector x indexed element<Ta>

Original text: Is an arrangement specifier, encoded in "Q", where 0->2S and 1->4S.

Where:

<Ta> Is an arrangement specifier, encoded in “Q”:

Q	<Ta>
0	2S
1	4S

Advanced SIMD vector x indexed element<Tb>

Original text: Is an arrangement specifier, encoded in "Q", where 0->4H and 1->8H.

Where:

<Tb> Is an arrangement specifier, encoded in “Q”:

Q	<Tb>
0	4H
1	8H

Advanced SIMD vector x indexed element<T>

Original text: Is an arrangement specifier, encoded in "Q", where 0->4H, 1->8H

Where:

<T> Is an arrangement specifier, encoded in “Q”:

Q	<T>
0	4H
1	8H

Advanced SIMD vector x indexed element<Tb>

Original text: Is an arrangement specifier, encoded in "Q", where 0->8B and 1->16B.

Where:

<Tb> Is an arrangement specifier, encoded in “Q”:

Q	<Tb>
0	8B
1	16B

Advanced SIMD vector x indexed element<T>

Original text: Is an arrangement specifier, encoded in "Q:sz", where 00->2S, 10->4S, 11->2D, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “Q:sz”:

Q	sz	<T>
0	0	2S
0	1	RESERVED
1	0	4S
1	1	2D

Advanced SIMD vector x indexed element<Ta>

Original text: Is an arrangement specifier, encoded in "size", where 01->4S and 10->2D, otherwise RESERVED.

Where:

<Ta> Is an arrangement specifier, encoded in “size”:

size	<Ta>
00	RESERVED
01	4S
10	2D
11	RESERVED

Advanced SIMD vector x indexed element<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 010->4H, 011->8H and 101->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	x	RESERVED
01	0	4H
01	1	8H
10	0	RESERVED
10	1	4S
11	x	RESERVED

Advanced SIMD vector x indexed element<T>

Original text: Is an arrangement specifier, encoded in "size:Q", where 010->4H, 011->8H, 100->2S and 101->4S, otherwise RESERVED.

Where:

<T> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	x	RESERVED
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD vector x indexed element<Tb>

Original text: Is an arrangement specifier, encoded in "size:Q", where 010->4H, 011->8H, 100->2S and 101->4S, otherwise RESERVED.

Where:

<Tb> Is an arrangement specifier, encoded in “size:Q”:

size	Q	<Tb>
00	x	RESERVED
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

Advanced SIMD vector x indexed element<Ts>

Original text: Is an element size specifier, encoded in "size", where 01->H and 10->S, otherwise RESERVED.

Where:

<Ts> Is an element size specifier, encoded in “size”:

size	<Ts>
00	RESERVED
01	H
10	S
11	RESERVED

Advanced SIMD vector x indexed element<Ts>

Original text: Is an element size specifier, encoded in "sz", where 0->S and 1->D.

Where:

<Ts> Is an element size specifier, encoded in “sz”:

sz	<Ts>
0	S
1	D

Advanced SIMD vector x indexed element<bt>

Original text: Is the bottom or top element specifier, encoded in "Q" where 0->B, 1->T.

Where:

<bt> Is the bottom or top element specifier, encoded in “Q”:

Q	<bt>
0	B
1	T

Advanced SIMD vector x indexed element<index>

Original text: Is the element index, encoded in "size:H:L", using "size", where 01->H:L and 10->H, otherwise RESERVED.

Where:

<index> Is the element index, encoded in “size:H:L”, based on “size”:

size	<index>
00	RESERVED
01	H:L
10	H
11	RESERVED

Advanced SIMD vector x indexed element<index>

Original text: Is the element index, encoded in "size:L:H:M", using "size", where 01->H:L:M and 10->H:L, otherwise RESERVED.

Where:

<index> Is the element index, encoded in “size:L:H:M”, based on “size”:

size	<index>
00	RESERVED
01	H:L:M
10	H:L
11	RESERVED

Advanced SIMD vector x indexed element<index>

Original text: Is the element index, encoded in "sz:L:H", using "sz:L", where 0x->H:L, 10->H, otherwise RESERVED.

Where:

<index> Is the element index, encoded in “sz:L:H”, based on “sz:L”:

sz	L	<index>
0	x	H:L
1	0	H
1	1	RESERVED

Advanced SIMD vector x indexed element<Vm>

Original text: Is the name of the second SIMD&FP source register, encoded in the "size:M:Rm" fields, using "size" where 01->0:Rm and 10->M:Rm, otherwise RESERVED. Restricted to V0-V15 when element size <Ts> is H.

Where:

<Vm> Is the name of the second SIMD&FP source register, encoded in “size:M:Rm”, based on “size”:

size	<Vm>
00	RESERVED
01	0:Rm
10	M:Rm
11	RESERVED

Restricted to V0-V15 when element size <Ts> is H.

Advanced SIMD vector x indexed element<rotate>

Original text: Is the rotation, encoded in "rot", where 00->0, 01->90, 10->180 and 11->270.

Where:

<rotate> Is the rotation, encoded in “rot”:

rot	<rotate>
00	0
01	90
10	180
11	270

Advanced SIMD vector x indexed element2

Original text: Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in "Q", where 0->[absent] and 1->[present].

Where:

2 Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in “Q”:

Q	2
0	[absent]
1	[present]

Hints<targets>

Original text: Is the type of indirection, encoded in "op2<2:1>", where 00->(omitted), 01->c, 10->j, 11->jc.

Where:

<targets> Is the type of indirection, encoded in “op2<2:1>”:

op2<2:1>	<targets>
00	(omitted)
01	c
10	j
11	jc

Load/store register (register offset)<extend>

Original text: Is the index extend specifier, encoded in "option", where 010->UXTW, 110->SXTW, 111->SXTX.

Where:

<extend> Is the index extend specifier, encoded in “option”:

option	<extend>
010	UXTW
110	SXTW
111	SXTX

Load/store register (register offset)<extend>

Original text: Is the index extend/shift specifier, defaulting to LSL, and which must be omitted for the LSL option when <amount> is omitted. Encoded in "option", where 010->UXTW, 011->LSL, 110->SXTW, 111->SXTX.

Where:

<extend> Is the index extend/shift specifier, defaulting to LSL, and which must be omitted for the LSL option when <amount> is omitted. encoded in “option”:

option	<extend>
010	UXTW
011	LSL
110	SXTW
111	SXTX

Load/store register (register offset)<amount>

Original text: Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in "S", where 0->#0, 1->#1.

Where:

<amount> Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in “S”:

S	<amount>
0	#0
1	#1

Load/store register (register offset)<amount>

Original text: Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in "S", where 0->#0, 1->#2.

Where:

<amount> Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in “S”:

S	<amount>
0	#0
1	#2

Load/store register (register offset)<amount>

Original text: Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in "S", where 0->#0, 1->#3.

Where:

<amount> Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in “S”:

S	<amount>
0	#0
1	#3

Load/store register (register offset)<amount>

Original text: Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in "S", where 0->#0, 1->#4.

Where:

<amount> Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in “S”:

S	<amount>
0	#0
1	#4

Logical (shifted register)<shift>

Original text: Is the optional shift to be applied to the final source, defaulting to LSL and encoded in "shift", where 00->LSL, 01->LSR, 10->ASR, 11->ROR.

Where:

<shift> Is the optional shift to be applied to the final source, defaulting to LSL and encoded in “shift”:

shift	<shift>
00	LSL
01	LSR
10	ASR
11	ROR

PSTATE<pstatefield>

Original text: Is a PSTATE field name. For the MSR instruction, this is encoded in "op1:op2:CRm", where 00000xxxxx->SEE(pstate), 000010xxxx->SEE(pstate), 000011xxxx->UAO (FEAT_UAO), 000100xxxx->PAN (FEAT_PAN), 000101xxxx->SPSel, 001000000x->ALLINT (FEAT_NMI), 011001xxxx->SSBS (FEAT_SSBS), 011010xxxx->DIT (FEAT_DIT), 011011001x->SVCRSM (FEAT_SME), 011011010x->SVCRZA (FEAT_SME), 011011011x->SVCRSMZA (FEAT_SME), 011100xxxx->TCO (FEAT_MTE), 011110xxxx->DAIFSet, 011111xxxx->DAIFClr, 001000001x->PM (FEAT_EBEP), otherwise RESERVED. For the SMSTART and SMSTOP aliases, this is encoded in "CRm<2:1>", where 0b01 specifies SVCRSM, 0b10 specifies SVCRZA, and 0b11 specifies SVCRSMZA.

Where:

<pstatefield> Is a PSTATE field name. For the MSR instruction, this is encoded in “op1:op2:CRm”:

op1	op2	CRm	<pstatefield>	Architectural Feature
000	00x	xxxx	SEE(pstate)	
000	010	xxxx	SEE(pstate)	
000	011	xxxx	UAO	
000	100	xxxx	PAN	
000	101	xxxx	SPSel	
000	11x	xxxx	RESERVED	
001	000	000x	ALLINT	
001	000	001x	PM	
001	000	01xx	RESERVED	
001	000	1xxx	RESERVED	
001	001	xxxx	RESERVED	
001	01x	xxxx	RESERVED	
001	1xx	xxxx	RESERVED	
010	xxx	xxxx	RESERVED	
011	000	xxxx	RESERVED	
011	001	xxxx	SSBS	
011	010	xxxx	DIT	
011	011	000x	RESERVED	
011	011	001x	SVCRSM	
011	011	010x	SVCRZA	
011	011	011x	SVCRSMZA	
011	011	1xxx	RESERVED	
011	100	xxxx	TCO	
011	101	xxxx	RESERVED	
011	110	xxxx	DAIFSet	
011	111	xxxx	DAIFClr	
1xx	xxx	xxxx	RESERVED	

For the SMSTART and SMSTOP aliases, this is encoded in "CRm<2:1>", where 0b01 specifies SVCRSM, 0b10 specifies SVCRZA, and 0b11 specifies SVCRSMZA.

PSTATE<option>

Original text: Is an optional mode, encoded in "CRm<2:1>", where 00->RESERVED, 01->SM, 10->ZA, 11->[no specifier].

Where:

<option> Is an optional mode, encoded in “CRm<2:1>”:

CRm<2:1>	<option>
00	RESERVED
01	SM
10	ZA
11	[no specifier]

System instructions<brb_op>

Original text: Is a BRB instruction name, as listed for the BRB system instruction group, encoded in "op2", where 100->IALL (FEAT_BRBE), 101->INJ (FEAT_BRBE).

Where:

<brb_op> Is a BRB instruction name, as listed for the BRB system instruction group, encoded in “op2”:

op2	<brb_op>	Architectural Feature
100	IALL	
101	INJ	

System instructions<dc_op>

Original text: Is a DC instruction name, as listed for the DC system instruction group, encoded in "op1:CRm:op2", where 0110100001->ZVA, 0000110001->IVAC, 0000110010->ISW, 0111010001->CVAC, 0111100001->CVAP (FEAT_DPB), 0111101001->CVADP (FEAT_DPB2), 0001010010->CSW, 0111011001->CVAU, 0111110001->CIVAC, 0001110010->CISW, 0000110011->IGVAC (FEAT_MTE2), 0000110100->IGSW (FEAT_MTE2), 0001010100->CGSW (FEAT_MTE2), 0001110100->CIGSW (FEAT_MTE2), 0111010011->CGVAC (FEAT_MTE), 0111100011->CGVAP (FEAT_MTE), 0111101011->CGVADP (FEAT_MTE), 0111110011->CIGVAC (FEAT_MTE), 0110100011->GVA (FEAT_MTE), 0000110101->IGDVAC (FEAT_MTE2), 0000110110->IGDSW (FEAT_MTE2), 0001010110->CGDSW (FEAT_MTE2), 0001110110->CIGDSW (FEAT_MTE2), 0111010101->CGDVAC (FEAT_MTE), 0111100101->CGDVAP (FEAT_MTE), 0111101101->CGDVADP (FEAT_MTE), 0111110101->CIGDVAC (FEAT_MTE), 0110100100->GZVA (FEAT_MTE), 1101110101->CIGDPAPA (FEAT_RME), 1101110001->CIPAPA (FEAT_RME), 1001110000->CIPAE (FEAT_MEC), 1001110111->CIGDPAE (FEAT_MEC).

Where:

<dc_op> Is a DC instruction name, as listed for the DC system instruction group, encoded in “op1:CRm:op2”:

op1	CRm	op2	<dc_op>	Architectural Feature
000	0110	001	IVAC	
000	0110	010	ISW	
000	0110	011	IGVAC	
000	0110	100	IGSW	
000	0110	101	IGDVAC	
000	0110	110	IGDSW	
000	1010	010	CSW	
000	1010	100	CGSW	
000	1010	110	CGDSW	
000	1110	010	CISW	
000	1110	100	CIGSW	
000	1110	110	CIGDSW	
011	0100	001	ZVA	
011	0100	011	GVA	
011	0100	100	GZVA	
011	1010	001	CVAC	
011	1010	011	CGVAC	
011	1010	101	CGDVAC	
011	1011	001	CVAU	
011	1100	001	CVAP	
011	1100	011	CGVAP	
011	1100	101	CGDVAP	
011	1101	001	CVADP	
011	1101	011	CGVADP	
011	1101	101	CGDVADP	
011	1110	001	CIVAC	
011	1110	011	CIGVAC	
011	1110	101	CIGDVAC	
100	1110	000	CIPAE	
100	1110	111	CIGDPAE	
110	1110	001	CIPAPA	
110	1110	101	CIGDPAPA	

System instructions<tlbi_op>

Original text: Is a TLBI instruction name, as listed for the TLBI system instruction group, encoded in the "op1:CRn:CRm:op2", where 10010000000001->IPAS2E1IS, 10010000000101->IPAS2LE1IS, 00010000011000->VMALLE1IS, 10010000011000->ALLE2IS, 11010000011000->ALLE3IS, 00010000011001->VAE1IS, 10010000011001->VAE2IS, 11010000011001->VAE3IS, 00010000011010->ASIDE1IS, 00010000011011->VAAE1IS, 10010000011100->ALLE1IS, 00010000011101->VALE1IS, 10010000011101->VALE2IS, 11010000011101->VALE3IS, 10010000011110->VMALLS12E1IS, 00010000011111->VAALE1IS, 10010000100001->IPAS2E1, 10010000100101->IPAS2LE1,

00010000111000->VMALLE1, 10010000111000->ALLE2, 11010000111000->ALLE3, 00010000111001->VAE1, 10010000111001->VAE2, 11010000111001->VAE3, 00010000111010->ASIDE1, 00010000111011->VAAE1, 10010000111100->ALLE1, 00010000111101->VALE1, 10010000111101->VALE2, 11010000111101->VALE3, 10010000111110->VMALLS12E1, 00010000111111->VAALE1, 00010000001000->VMALLE1OS (FEAT_TLBIOS), 00010000001001->VAE1OS (FEAT_TLBIOS), 00010000001010->ASIDE1OS (FEAT_TLBIOS), 00010000001011->VAAE1OS (FEAT_TLBIOS), 00010000001101->VALE1OS (FEAT_TLBIOS), 00010000001111->VAALE1OS (FEAT_TLBIOS), 10010000100000->IPAS2E1OS (FEAT_TLBIOS), 10010000100100->IPAS2LE1OS (FEAT_TLBIOS), 10010000001001->VAE2OS (FEAT_TLBIOS), 10010000001101->VALE2OS (FEAT_TLBIOS), 10010000001110->VMALLS12E1OS (FEAT_TLBIOS), 11010000001001->VAE3OS (FEAT_TLBIOS), 11010000001101->VALE3OS (FEAT_TLBIOS), 10010000001000->ALLE2OS (FEAT_TLBIOS), 10010000001100->ALLE1OS (FEAT_TLBIOS), 11010000001000->ALLE3OS (FEAT_TLBIOS), 00010000110001->RVAE1 (FEAT_TLBIRANGE), 00010000110011->RVAAE1 (FEAT_TLBIRANGE), 00010000110101->RVALE1 (FEAT_TLBIRANGE), 00010000110111->RVAALE1 (FEAT_TLBIRANGE), 00010000010001->RVAE1IS (FEAT_TLBIRANGE), 00010000010011->RVAAE1IS (FEAT_TLBIRANGE), 00010000010101->RVALE1IS (FEAT_TLBIRANGE), 00010000010111->RVAALE1IS (FEAT_TLBIRANGE), 00010000101001->RVAE1OS (FEAT_TLBIRANGE), 00010000101011->RVAAE1OS (FEAT_TLBIRANGE), 00010000101101->RVALE1OS (FEAT_TLBIRANGE), 00010000101111->RVAALE1OS (FEAT_TLBIRANGE), 10010000000010->RIPAS2E1IS (FEAT_TLBIRANGE), 10010000000110->RIPAS2LE1IS (FEAT_TLBIRANGE), 10010000100010->RIPAS2E1 (FEAT_TLBIRANGE), 10010000100110->RIPAS2LE1 (FEAT_TLBIRANGE), 10010000100011->RIPAS2E1OS (FEAT_TLBIRANGE), 10010000100111->RIPAS2LE1OS (FEAT_TLBIRANGE), 10010000110001->RVAE2 (FEAT_TLBIRANGE), 10010000110101->RVALE2 (FEAT_TLBIRANGE), 10010000010001->RVAE2IS (FEAT_TLBIRANGE), 10010000010101->RVALE2IS (FEAT_TLBIRANGE), 10010000101001->RVAE2OS (FEAT_TLBIRANGE), 10010000101101->RVALE2OS (FEAT_TLBIRANGE), 11010000110001->RVAE3 (FEAT_TLBIRANGE), 11010000110101->RVALE3 (FEAT_TLBIRANGE), 11010000010001->RVAE3IS (FEAT_TLBIRANGE), 11010000010101->RVALE3IS (FEAT_TLBIRANGE), 11010000101001->RVAE3OS (FEAT_TLBIRANGE), 11010000101101->RVALE3OS (FEAT_TLBIRANGE), 10010010000001->IPAS2E1ISNXS (FEAT_XS), 10010010000101->IPAS2LE1ISNXS (FEAT_XS), 00010010011000->VMALLE1ISNXS (FEAT_XS), 10010010011000->ALLE2ISNXS (FEAT_XS), 11010010011000->ALLE3ISNXS (FEAT_XS), 00010010011001->VAE1ISNXS (FEAT_XS), 10010010011001->VAE2ISNXS (FEAT_XS), 11010010011001->VAE3ISNXS (FEAT_XS), 00010010011010->ASIDE1ISNXS (FEAT_XS), 00010010011011->VAAE1ISNXS (FEAT_XS), 10010010011100->ALLE1ISNXS (FEAT_XS), 00010010011101->VALE1ISNXS (FEAT_XS), 10010010011101-

>VALE2ISNXS (FEAT_XS), 11010010011101->VALE3ISNXS (FEAT_XS), 10010010011110->VMALLS12E1ISNXS (FEAT_XS), 00010010011111->VAALE1ISNXS (FEAT_XS), 10010010100001->IPAS2E1NXS (FEAT_XS), 10010010100101->IPAS2LE1NXS (FEAT_XS), 00010010111000->VMALLE1NXS (FEAT_XS), 10010010111000->ALLE2NXS (FEAT_XS), 11010010111000->ALLE3NXS (FEAT_XS), 00010010111001->VAE1NXS (FEAT_XS), 10010010111001->VAE2NXS (FEAT_XS), 11010010111001->VAE3NXS (FEAT_XS), 00010010111010->ASIDE1NXS (FEAT_XS), 00010010111011->VAAE1NXS (FEAT_XS), 10010010111100->ALLE1NXS (FEAT_XS), 00010010111101->VALE1NXS (FEAT_XS), 10010010111101->VALE2NXS (FEAT_XS), 11010010111101->VALE3NXS (FEAT_XS), 10010010111110->VMALLS12E1NXS (FEAT_XS), 00010010111111->VAALE1NXS (FEAT_XS), 00010010001000->VMALLE1OSNXS (FEAT_XS), 00010010001001->VAE1OSNXS (FEAT_XS), 00010010001010->ASIDE1OSNXS (FEAT_XS), 00010010001011->VAAE1OSNXS (FEAT_XS), 00010010001101->VALE1OSNXS (FEAT_XS), 00010010001111->VAALE1OSNXS (FEAT_XS), 10010010100000->IPAS2E1OSNXS (FEAT_XS), 10010010100100->IPAS2LE1OSNXS (FEAT_XS), 10010010001001->VAE2OSNXS (FEAT_XS), 10010010001101->VALE2OSNXS (FEAT_XS), 10010010001110->VMALLS12E1OSNXS (FEAT_XS), 11010010001001->VAE3OSNXS (FEAT_XS), 11010010001101->VALE3OSNXS (FEAT_XS), 10010010001000->ALLE2OSNXS (FEAT_XS), 10010010001100->ALLE1OSNXS (FEAT_XS), 11010010001000->ALLE3OSNXS (FEAT_XS), 00010010110001->RVAE1NXS (FEAT_XS), 00010010110011->RVAAE1NXS (FEAT_XS), 00010010110101->RVALE1NXS (FEAT_XS), 00010010110111->RVAALE1NXS (FEAT_XS), 00010010010001->RVAE1ISNXS (FEAT_XS), 00010010010011->RVAAE1ISNXS (FEAT_XS), 00010010010101->RVALE1ISNXS (FEAT_XS), 00010010010111->RVAALE1ISNXS (FEAT_XS), 00010010101001->RVAE1OSNXS (FEAT_XS), 00010010101011->RVAAE1OSNXS (FEAT_XS), 00010010101101->RVALE1OSNXS (FEAT_XS), 00010010101111->RVAALE1OSNXS (FEAT_XS), 10010010000010->RIPAS2E1ISNXS (FEAT_XS), 10010010000110->RIPAS2LE1ISNXS (FEAT_XS), 10010010100010->RIPAS2E1NXS (FEAT_XS), 10010010100110->RIPAS2LE1NXS (FEAT_XS), 10010010100011->RIPAS2E1OSNXS (FEAT_XS), 10010010100111->RIPAS2LE1OSNXS (FEAT_XS), 10010010110001->RVAE2NXS (FEAT_XS), 10010010110101->RVALE2NXS (FEAT_XS), 10010010010001->RVAE2ISNXS (FEAT_XS), 10010010010101->RVALE2ISNXS (FEAT_XS), 10010010101001->RVAE2OSNXS (FEAT_XS), 10010010101101->RVALE2OSNXS (FEAT_XS), 11010010110001->RVAE3NXS (FEAT_XS), 11010010110101->RVALE3NXS (FEAT_XS), 11010010010001->RVAE3ISNXS (FEAT_XS), 11010010010101->RVALE3ISNXS (FEAT_XS), 11010010101001->RVAE3OSNXS (FEAT_XS), 11010010101101->RVALE3OSNXS (FEAT_XS), 11010000001100->PAALLOS (FEAT_RME), 11010000100011->RPAOS (FEAT_RME), 11010000100111->RPALOS (FEAT_RME), 11010000111100->PAALL (FEAT_RME).

Where:

<tlbi_op> Is a TLBI instruction name, as listed for the TLBI system instruction group, encoded in “op1:CRn:CRm:op2”:

op1	CRn	CRm	op2	<tlbi_op>	Architectural Feature
000	1000	0001	000	VMALLE1OS	
000	1000	0001	001	VAE1OS	
000	1000	0001	010	ASIDE1OS	
000	1000	0001	011	VAAE1OS	
000	1000	0001	101	VALE1OS	
000	1000	0001	111	VAALE1OS	
000	1000	0010	001	RVAE1IS	
000	1000	0010	011	RVAAE1IS	
000	1000	0010	101	RVALE1IS	
000	1000	0010	111	RVAALE1IS	
000	1000	0011	000	VMALLE1IS	
000	1000	0011	001	VAE1IS	
000	1000	0011	010	ASIDE1IS	
000	1000	0011	011	VAAE1IS	
000	1000	0011	101	VALE1IS	
000	1000	0011	111	VAALE1IS	
000	1000	0101	001	RVAE1OS	
000	1000	0101	011	RVAAE1OS	
000	1000	0101	101	RVALE1OS	
000	1000	0101	111	RVAALE1OS	
000	1000	0110	001	RVAE1	
000	1000	0110	011	RVAAE1	
000	1000	0110	101	RVALE1	
000	1000	0110	111	RVAALE1	
000	1000	0111	000	VMALLE1	
000	1000	0111	001	VAE1	
000	1000	0111	010	ASIDE1	
000	1000	0111	011	VAAE1	
000	1000	0111	101	VALE1	
000	1000	0111	111	VAALE1	
000	1001	0001	000	VMALLE1OSNXS	
000	1001	0001	001	VAE1OSNXS	
000	1001	0001	010	ASIDE1OSNXS	
000	1001	0001	011	VAAE1OSNXS	
000	1001	0001	101	VALE1OSNXS	
000	1001	0001	111	VAALE1OSNXS	
000	1001	0010	001	RVAE1ISNXS	
000	1001	0010	011	RVAAE1ISNXS	
000	1001	0010	101	RVALE1ISNXS	
000	1001	0010	111	RVAALE1ISNXS	
000	1001	0011	000	VMALLE1ISNXS	
000	1001	0011	001	VAE1ISNXS	
000	1001	0011	010	ASIDE1ISNXS	
000	1001	0011	011	VAAE1ISNXS	
000	1001	0011	101	VALE1ISNXS	
000	1001	0011	111	VAALE1ISNXS	
000	1001	0101	001	RVAE1OSNXS	
000	1001	0101	011	RVAAE1OSNXS	
000	1001	0101	101	RVALE1OSNXS	
000	1001	0101	111	RVAALE1OSNXS	
000	1001	0110	001	RVAE1NXS	
000	1001	0110	011	RVAAE1NXS	
000	1001	0110	101	RVALE1NXS	

System instructions<at_op>

Original text: Is an AT instruction name, as listed for the AT system instruction group, encoded in the "op1:CRm<0>:op2", where 0000000->S1E1R, 0001000->S1E1RP (FEAT_PAN2), 0001010->S1E1A (FEAT_ATS1A), 1000000->S1E2R, 1100000->S1E3R, 0000001->S1E1W, 0001001->S1E1WP (FEAT_PAN2), 1000001->S1E2W, 1100001->S1E3W, 0000010->S1E0R, 0000011->S1E0W, 1000100->S12E1R, 1000101->S12E1W, 1000110->S12E0R, 1000111->S12E0W, 1001010->S1E2A (FEAT_ATS1A), 1101010->S1E3A (FEAT_ATS1A).

Where:

<at_op> Is an AT instruction name, as listed for the AT system instruction group, encoded in “op1:CRm<0>:op2”:

op1	CRm<0>	op2	<at_op>	Architectural Feature
000	0	000	S1E1R	
000	0	001	S1E1W	
000	0	010	S1E0R	
000	0	011	S1E0W	
000	1	000	S1E1RP	
000	1	001	S1E1WP	
000	1	010	S1E1A	
100	0	000	S1E2R	
100	0	001	S1E2W	
100	0	100	S12E1R	
100	0	101	S12E1W	
100	0	110	S12E0R	
100	0	111	S12E0W	
100	1	010	S1E2A	
110	0	000	S1E3R	
110	0	001	S1E3W	
110	1	010	S1E3A	

System instructions<ic_op>

Original text: Is an IC instruction name, as listed for the IC system instruction pages, encoded in "op1:CRm:op2", where 0000001000->IALLUIS, 0000101000->IALLU, 0110101001->IVAU.

Where:

<ic_op> Is an IC instruction name, as listed for the IC system instruction pages, encoded in “op1:CRm:op2”:

op1	CRm	op2	<ic_op>
000	0001	000	IALLUIS
000	0101	000	IALLU
011	0101	001	IVAU

System pair instructions<tlbip_op>

Original text: Is a TLBIP instruction name, as listed for the TLBIP system pair instruction group, encoded in "op1:CRn:CRm:op2", where 10010000000001->IPAS2E1IS (FEAT_D128), 10010000000101->IPAS2LE1IS (FEAT_D128), 00010000011001->VAE1IS (FEAT_D128), 10010000011001->VAE2IS (FEAT_D128), 11010000011001->VAE3IS (FEAT_D128), 00010000011011->VAAE1IS (FEAT_D128), 00010000011101->VALE1IS (FEAT_D128), 10010000011101->VALE2IS (FEAT_D128), 11010000011101->VALE3IS (FEAT_D128), 00010000011111->VAALE1IS (FEAT_D128), 10010000100001->IPAS2E1 (FEAT_D128), 10010000100101->IPAS2LE1 (FEAT_D128), 00010000111001->VAE1 (FEAT_D128), 10010000111001->VAE2 (FEAT_D128), 11010000111001->VAE3 (FEAT_D128), 00010000111011->VAAE1 (FEAT_D128), 00010000111101->VALE1 (FEAT_D128), 10010000111101->VALE2 (FEAT_D128), 11010000111101->VALE3 (FEAT_D128), 00010000111111->VAALE1 (FEAT_D128), 00010000001001->VAE1OS (FEAT_D128), 00010000001011->VAAE1OS (FEAT_D128), 00010000001101->VALE1OS (FEAT_D128), 00010000001111->VAALE1OS (FEAT_D128), 10010000100000->IPAS2E1OS (FEAT_D128), 10010000100100->IPAS2LE1OS (FEAT_D128), 10010000001001->VAE2OS (FEAT_D128), 10010000001101->VALE2OS (FEAT_D128), 11010000001001->VAE3OS (FEAT_D128), 11010000001101->VALE3OS (FEAT_D128), 00010000110001->RVAE1 (FEAT_D128), 00010000110011->RVAAE1 (FEAT_D128), 00010000110101->RVALE1 (FEAT_D128), 00010000110111->RVAALE1 (FEAT_D128), 00010000010001->RVAE1IS (FEAT_D128), 00010000010011->RVAAE1IS (FEAT_D128), 00010000010101->RVALE1IS (FEAT_D128), 00010000010111->RVAALE1IS (FEAT_D128), 00010000101001->RVAE1OS (FEAT_D128), 00010000101011->RVAAE1OS (FEAT_D128), 00010000101101->RVALE1OS (FEAT_D128), 00010000101111->RVAALE1OS (FEAT_D128), 10010000000010->RIPAS2E1IS (FEAT_D128), 10010000000110->RIPAS2LE1IS (FEAT_D128), 10010000100010->RIPAS2E1 (FEAT_D128), 10010000100110->RIPAS2LE1 (FEAT_D128), 10010000100011->RIPAS2E1OS (FEAT_D128), 10010000100111->RIPAS2LE1OS (FEAT_D128), 10010000110001->RVAE2 (FEAT_D128), 10010000110101->RVALE2 (FEAT_D128), 10010000010001->RVAE2IS (FEAT_D128), 10010000010101->RVALE2IS (FEAT_D128), 10010000101001->RVAE2OS (FEAT_D128), 10010000101101->RVALE2OS (FEAT_D128), 11010000110001->RVAE3 (FEAT_D128), 11010000110101->RVALE3 (FEAT_D128), 11010000010001->RVAE3IS (FEAT_D128), 11010000010101->RVALE3IS (FEAT_D128), 11010000101001->RVAE3OS (FEAT_D128), 11010000101101->RVALE3OS (FEAT_D128), 10010010000001->IPAS2E1ISNXS (FEAT_D128), 10010010000101->IPAS2LE1ISNXS (FEAT_D128), 00010010011001->VAE1ISNXS (FEAT_D128), 10010010011001->VAE2ISNXS (FEAT_D128), 11010010011001->VAE3ISNXS (FEAT_D128), 00010010011011->VAAE1ISNXS (FEAT_D128), 00010010011101->VALE1ISNXS (FEAT_D128), 10010010011101->VALE2ISNXS (FEAT_D128), 11010010011101->VALE3ISNXS (FEAT_D128), 00010010011111->VAALE1ISNXS (FEAT_D128),

10010010100001->IPAS2E1NXS (FEAT_D128), 10010010100101->IPAS2LE1NXS (FEAT_D128), 00010010111001->VAE1NXS (FEAT_D128), 10010010111001->VAE2NXS (FEAT_D128), 11010010111001->VAE3NXS (FEAT_D128), 00010010111011->VAAE1NXS (FEAT_D128), 00010010111101->VALE1NXS (FEAT_D128), 10010010111101->VALE2NXS (FEAT_D128), 11010010111101->VALE3NXS (FEAT_D128), 00010010111111->VAALE1NXS (FEAT_D128), 00010010001001->VAE1OSNXS (FEAT_D128), 00010010001011->VAAE1OSNXS (FEAT_D128), 00010010001101->VALE1OSNXS (FEAT_D128), 00010010001111->VAALE1OSNXS (FEAT_D128), 10010010100000->IPAS2E1OSNXS (FEAT_D128), 10010010100100->IPAS2LE1OSNXS (FEAT_D128), 10010010001001->VAE2OSNXS (FEAT_D128), 10010010001101->VALE2OSNXS (FEAT_D128), 11010010001001->VAE3OSNXS (FEAT_D128), 11010010001101->VALE3OSNXS (FEAT_D128), 00010010110001->RVAE1NXS (FEAT_D128), 00010010110011->RVAAE1NXS (FEAT_D128), 00010010110101->RVALE1NXS (FEAT_D128), 00010010110111->RVAALE1NXS (FEAT_D128), 00010010010001->RVAE1ISNXS (FEAT_D128), 00010010010011->RVAAE1ISNXS (FEAT_D128), 00010010010101->RVALE1ISNXS (FEAT_D128), 00010010010111->RVAALE1ISNXS (FEAT_D128), 00010010101001->RVAE1OSNXS (FEAT_D128), 00010010101011->RVAAE1OSNXS (FEAT_D128), 00010010101101->RVALE1OSNXS (FEAT_D128), 00010010101111->RVAALE1OSNXS (FEAT_D128), 10010010000010->RIPAS2E1ISNXS (FEAT_D128), 10010010000110->RIPAS2LE1ISNXS (FEAT_D128), 10010010100010->RIPAS2E1NXS (FEAT_D128), 10010010100110->RIPAS2LE1NXS (FEAT_D128), 10010010100011->RIPAS2E1OSNXS (FEAT_D128), 10010010100111->RIPAS2LE1OSNXS (FEAT_D128), 10010010110001->RVAE2NXS (FEAT_D128), 10010010110101->RVALE2NXS (FEAT_D128), 10010010010001->RVAE2ISNXS (FEAT_D128), 10010010010101->RVALE2ISNXS (FEAT_D128), 10010010101001->RVAE2OSNXS (FEAT_D128), 10010010101101->RVALE2OSNXS (FEAT_D128), 11010010110001->RVAE3NXS (FEAT_D128), 11010010110101->RVALE3NXS (FEAT_D128), 11010010010001->RVAE3ISNXS (FEAT_D128), 11010010010101->RVALE3ISNXS (FEAT_D128), 11010010101001->RVAE3OSNXS (FEAT_D128), 11010010101101->RVALE3OSNXS (FEAT_D128).

Where:

<tlbip_op> Is a TLBIP instruction name, as listed for the TLBIP system pair instruction group, encoded in “op1:CRn:CRm:op2”:

op1	CRn	CRm	op2	<tlbip_op>	Architectural Feature
000	1000	0001	001	VAE1OS	
000	1000	0001	011	VAAE1OS	
000	1000	0001	101	VALE1OS	
000	1000	0001	111	VAALE1OS	
000	1000	0010	001	RVAE1IS	
000	1000	0010	011	RVAAE1IS	
000	1000	0010	101	RVALE1IS	
000	1000	0010	111	RVAALE1IS	
000	1000	0011	001	VAE1IS	
000	1000	0011	011	VAAE1IS	
000	1000	0011	101	VALE1IS	
000	1000	0011	111	VAALE1IS	
000	1000	0101	001	RVAE1OS	
000	1000	0101	011	RVAAE1OS	
000	1000	0101	101	RVALE1OS	
000	1000	0101	111	RVAALE1OS	
000	1000	0110	001	RVAE1	
000	1000	0110	011	RVAAE1	
000	1000	0110	101	RVALE1	
000	1000	0110	111	RVAALE1	
000	1000	0111	001	VAE1	
000	1000	0111	011	VAAE1	
000	1000	0111	101	VALE1	
000	1000	0111	111	VAALE1	
000	1001	0001	001	VAE1OSNXS	
000	1001	0001	011	VAAE1OSNXS	
000	1001	0001	101	VALE1OSNXS	
000	1001	0001	111	VAALE1OSNXS	
000	1001	0010	001	RVAE1ISNXS	
000	1001	0010	011	RVAAE1ISNXS	
000	1001	0010	101	RVALE1ISNXS	
000	1001	0010	111	RVAALE1ISNXS	
000	1001	0011	001	VAE1ISNXS	
000	1001	0011	011	VAAE1ISNXS	
000	1001	0011	101	VALE1ISNXS	
000	1001	0011	111	VAALE1ISNXS	
000	1001	0101	001	RVAE1OSNXS	
000	1001	0101	011	RVAAE1OSNXS	
000	1001	0101	101	RVALE1OSNXS	
000	1001	0101	111	RVAALE1OSNXS	
000	1001	0110	001	RVAE1NXS	
000	1001	0110	011	RVAAE1NXS	
000	1001	0110	101	RVALE1NXS	
000	1001	0110	111	RVAALE1NXS	
000	1001	0111	001	VAE1NXS	
000	1001	0111	011	VAAE1NXS	
000	1001	0111	101	VALE1NXS	
000	1001	0111	111	VAALE1NXS	
100	1000	0000	001	IPAS2E1IS	
100	1000	0000	010	RIPAS2E1IS	
100	1000	0000	101	IPAS2LE1IS	
100	1000	0000	110	RIPAS2LE1IS	
100	1000	0001	001	VAE2OS	

System register move<op0>

Original text: Is an unsigned immediate, encoded in "o0", where 0->2, 1->3.

Where:

<op0> Is an unsigned immediate, encoded in “o0”:

o0	<op0>
0	2
1	3

System register pair move<op0>

Original text: Is an unsigned immediate, encoded in "o0", where 0->2, 1->3.

Where:

<op0> Is an unsigned immediate, encoded in “o0”:

o0	<op0>
0	2
1	3

Test and branch (immediate)<R>

Original text: Is a width specifier, encoded in "b5", where 1->X, 0->W. * In assembler source code an 'X' specifier is always permitted, but a 'W' specifier is only permitted when the bit number is less than 32.

Where:

<R> Is a width specifier, encoded in “b5”:

b5	<R>
0	W
1	X

In assembler source code an 'X' specifier is always permitted, but a 'W' specifier is only permitted when the bit number is less than 32.

Internal version only: isa v33.64, AdvSIMD v29.12, pseudocode no_diffs_2023_09_RC2, sve v2023-06_rel ; Build timestamp: 2023-09-18T17:56